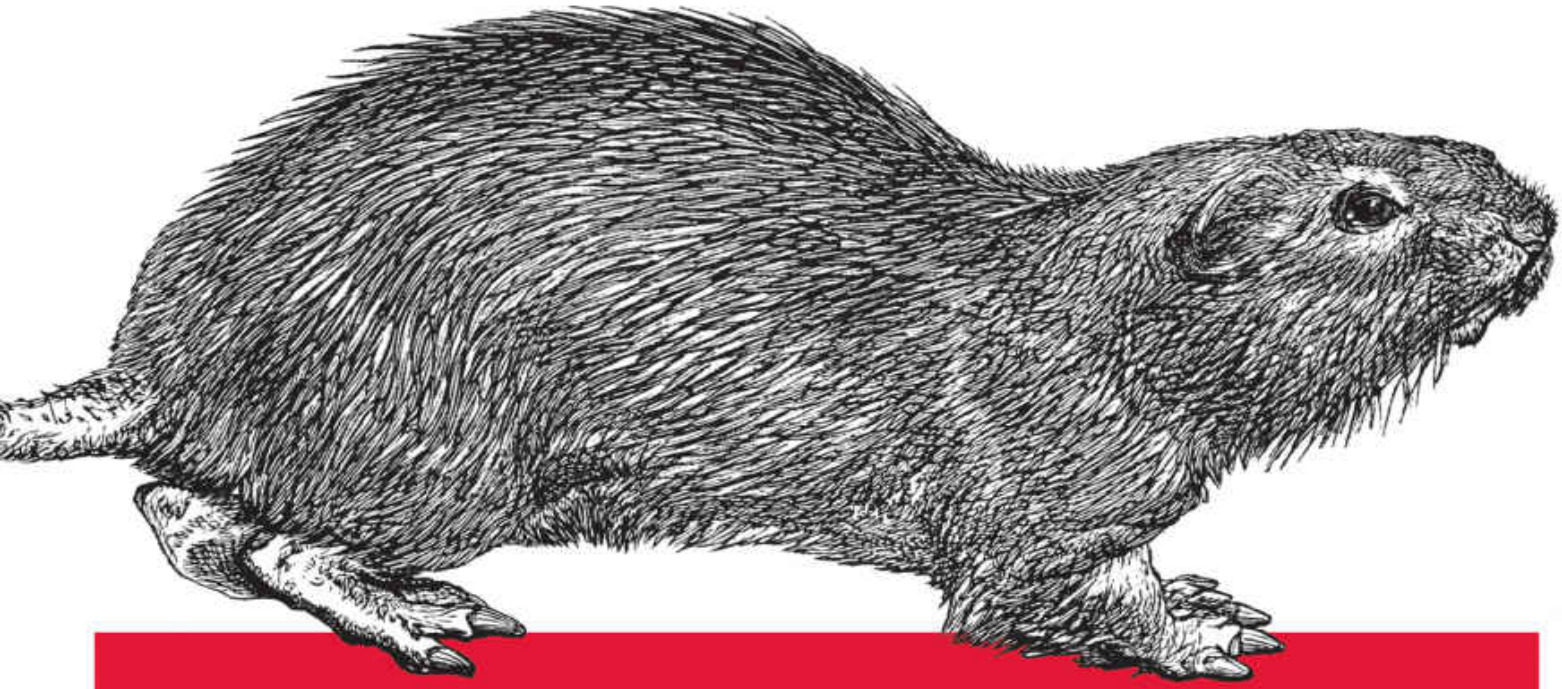
Análise de dados do Facebook, Twitter, LinkedIn e outros sites de mídia social



Mineração de Dados da Web Social

O'REILLY®
novatec

Matthew A. Russell

Elogios para o livro Mineração de dados da web social

"O livro *Mineração de dados da web social* é leitura essencial, pois cada vez mais dados são distribuídos em um ritmo frenético. Ao escrever um manual perfeito tanto para quem trabalha com APIs, quanto para fãs de mídias sociais e cientistas de dados, [Matthew] Russell habilmente aproveitou ao máximo a enorme oportunidade apresentada pela mineração de dados das mídias sociais."

– Nick Ducoff, CEO da Infochimps Inc.

"Este é um guia essencial para que você possa aproveitar a nova geração de fontes de dados on-line. Russell fez um excelente trabalho criando um manual acessível a qualquer pessoa que trabalhe com informações sociais na web, abordando desde como acessá-las, até os métodos simples para extração de insights surpreendentes de todos esses dados brutos."

– Pete Warden, Fundador do OpenHeatMap.com

"Este livro será agora meu principal recurso quando estiver trabalhando em qualquer projeto que envolva análise de dados sociais, uma vez que contém inúmeros exemplos e é altamente recomendado, independentemente do projeto de mineração de dados que você tiver em mente. Além disso, seu texto é ótimo tanto para iniciantes, quanto para leitores avançados."

– Abe Music, Diretor da Zaffra

"Este livro é evidentemente um projeto apaixonado de seu autor, combinando habilmente o uso de bibliotecas clássicas de mineração em textos e grafos com aplicações atuais de mídias sociais. Os exemplos fornecidos são concretos e concisos, mas também oferecem insights muito úteis que auxiliam o desenvolvimento e a exploração futura do leitor. Este texto é um ótimo manual para os que estão dando seus primeiros passos em busca de extração de informações das redes sociais, mas também para pesquisadores avançados, que desejam acessar as mais recentes APIs de mídias sociais."

– Chris Augeri, Fellow de pesquisa sênior, Universidade de Nebraska

"Este é um livro fenomenal para todos que querem trabalhar com mineração de dados sociais, tendo sido muito bem pesquisado e oferecendo inúmeros exemplos que podem ser postos em prática desde o primeiro capítulo. Além de fácil, sua leitura é muito prazerosa. Recomendo este livro a todos os interessados em mineração, análise e visualização de dados da web social."

– Jeffrey Humphries, PhD; Cientista da computação

"Poucas inovações terão tanto impacto em nosso dia a dia, nos anos vindouros, como o entendimento automatizado da comunicação humana por software. Trata-se de um tópico amplo e profundo que já foi assunto de milhares de estudos e de centenas de dissertações. O que Matthew criou foi algo que realmente nos faltava: uma introdução aplicada a um conjunto diverso e profundo de tecnologias e tópicos que torna acessível o conhecimento oculto na comunicação humana dentro da web social. Este é o trabalho de um poderoso tecnólogo – alguém que pode equipar programadores habilitados com novas ferramentas verdadeiramente valiosas.

Leia este livro e ele lhe abrirá portas para o futuro do software da próxima década."

– Tim Estes, Fundador e CEO da Digital Reasoning

"O livro *Mineração de dados da web social* é um ótimo recurso para quem deseja obter o máximo da API do Twitter."

– Raffi Krikorian, Grupo de Platform Services do Twitter

"Matthew aborda um grupo interessante e eclético de fontes de dados, técnicas de análise, ferramentas de gerenciamento de dados e visualizações que fornecem uma análise das ideias mais recentes sobre como obter insight da web social. Seus exemplos são vívidos e servem como ótimos pontos de partida para maiores explorações. Matthew evidentemente faz questão de que o leitor compreenda o material; seu livro é repleto de dicas e conselhos pontuais, inteligentes e verdadeiramente úteis, e fez com que eu me animasse a pesquisar mais acerca dessa rica área de análise."

– Roger Magoulas, Diretor de pesquisa de mercado da O'Reilly Media

Mineração de dados da web social

Matthew A. Russell


O'REILLY®
Novatec

Nota sobre a tradução

Ainda que, em suas páginas brasileiras, o Facebook adote o termo "aplicativo" como tradução para a palavra "application", neste livro como um todo adotamos a palavra "aplicação" por considerá-la mais adequada ao significado da expressão em inglês, sem prejuízo do entendimento do texto.

Tradução em português autorizada da edição em inglês da obra *Mining the Social Web*, First Edition ISBN 9781449388348 © 2010, Matthew Russell. Esta tradução é publicada e vendida com a permissão da O'Reilly Media, Inc., detentora de todos os direitos para publicação e venda desta obra.

Editor: Rubens Prates
Tradução: Rafael Zanolli
Revisão gramatical: Jeferson Ferreira
Revisão técnica: Leonardo Naressi (Direct Performance)
Editoração eletrônica: Camila Kuwabata / Carolina Kuwabata

ISBN: 978-85-7522-748-0

Histórico de edições impressas:

Agosto/2011 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Sumário

Prefácio à edição brasileira

Conhecimento puro. É assim que podemos definir este livro. Quem sempre estudou Data Mining, Text Mining, ou outras modalidades de descoberta de conhecimento pode ficar perplexo com a aplicação prática que o autor traz em cada capítulo. E quem começou agora a se aventurar no mundo das redes e mídias sociais e no universo de dados chamado internet, mal pode imaginar a profusão de possibilidades que se dá ao juntarmos este conhecimento acadêmico à aplicação prática nos negócios e no dia-a-dia das pessoas e, agora, também usuários conectados.

Matthew Russel liga brilhantemente estas duas pontas: pesquisas acadêmicas, algoritmos e teorias com implementação e exemplificação no "mundo real" do Twitter, Facebook, Google Buzz, Linkedin e, como se não bastasse, visita a disciplina cada vez mais popular de visualização da informação.

E se você acha que todo esse conhecimento puro, explanações acadêmicas e teorias existem só no papel, fique felizmente enganado, pois todas as explicações acompanham exemplos práticos e com código-fonte disponível. Nada como compartilhar código-aberto e ajudar a criar o mundo open-source, não é?

O que você aprender aqui com certeza lhe será útil em todo o futuro próximo da internet e das relações humanas, cada vez mais mediadas por computadores. Basta chegar ao último capítulo e ver que o universo da mineração e processamento computacional do conhecimento humano está apenas no começo de seu potencial e já apresenta bons exemplos mediante algoritmos de inferência lógica e inteligência artificial.

Detalhes técnicos para nós, brasileiros

Todos os exemplos deste livro foram preparados para o idioma inglês, usando a codificação UTF-8. O que isso significa na prática? Significa que ao tentar executar estes exemplos com conteúdos em outros idiomas, como nosso português do Brasil (pt-BR), alguns problemas podem

ocorrer, principalmente com caracteres estendidos como cedilhas, sinais de acentuação, entre outras notações.

Muitas vezes, nem percebemos, mas nossos feeds no Twitter, Google Buzz, Linkedin e outras fontes de dados usadas nos exemplos deste livro estão cheias destes caracteres.

Então, caso você queira usar os códigos destes exemplos para conteúdo em português, algumas adaptações deverão ser feitas nos códigos. Reunimos aqui alguns casos comuns para guiá-lo nesta rápida tarefa.

1. Codificar todo texto consumido para UTF-8, evitando erros de conversão ASCII.

   Exemplo: ao ler o conteúdo de um arquivo texto com caracteres estendidos, pode ser necessário codificar para UTF-8 antes de carregar um objeto JSON. Veja o exemplo original:

   ```
   blog_data = json.loads(open(BLOG_DATA).read())
   ```

   E veja como fica com a correção:

   ```
   blog_data = json.loads(open(BLOG_DATA).read().encode('utf-8'))
   ```

   Os métodos em Python para lidar com codificações diferentes, e em especial Unicode, são muito eficientes e completos. Recomendo a leitura da documentação, para estar pronto para todos os casos, ou se precisar adaptar algum trecho dos códigos. Veja: *http://docs.python.org/howto/unicode.html*.

2. Elementos de processamento léxico também devem ser adaptados para o português. Alguns exemplos do livro utilizam bibliotecas de processamento de linguagem natural e o código exemplifica o uso em inglês. Ao usar a biblioteca NLTK, por exemplo, é necessário adaptar o código para usar stopwords (palavras de parada) para o português. Veja o código usado nos exemplos originais:

   ```
   nltk.stopwords('english')
   ```

   Neste caso, a adaptação é tão simples quanto mudar a string do idioma. Veja como fica:

   ```
   nltk.stopwords('portuguese')
   ```

   Outro problema encontrado na aplicação do NLTK nos exemplos deste livro é com relação ao método de tokenização padrão. Os exemplos aplicam o método word_tokenize, que não considera caracteres acentuados ou cedilhas como parte de uma palavra válida.

Veja a chamada original:

```
nltk.tokenize.word_tokenize(s)
```

Neste caso, recomendo a troca do método de tokenização para o método `wordpunct_tokenize`, que é mais simples e não ignora os caracteres estendidos. Veja como fica:

```
nltk.tokenize.wordpunct_tokenize(s)
```

Recomendo também a leitura do artigo "Examples for Portuguese Processing", da documentação do NLTK, caso queira se aprofundar: *http://nltk.googlecode.com/svn/trunk/doc/howto/portuguese_en.html*

– Leonardo Naressi

Sócio-diretor da Direct Performance (*www.directperformance.com.br*), consultoria
brasileira de Business Intelligence e Web Analytics com foco em marketing digital.

Prefácio

> *A web é mais uma criação social do que uma técnica. Eu a projetei em busca de um efeito social – ajudar as pessoas a trabalharem juntas – e não como um brinquedo técnico. O objetivo maior da Web é sustentar e aprimorar nossa existência interconectada no mundo. Nela, nos reunimos em famílias, associações e empresas, desenvolvendo confiança, mesmo separados por muitos quilômetros, e desconfiança de nossos próprios vizinhos.*
>
> *– Tim Berners-Lee, Weaving the Web (Harper)*

Quem deve ler este livro?

Se você tem experiência básica como programador e está interessado em insights a respeito das oportunidades que surgem com mineração e análise de dados da web social, veio ao lugar certo. Daqui a apenas algumas páginas colocaremos a mão na massa. Serei franco, entretanto, e direi que uma das principais queixas que você provavelmente terá quanto a este livro é a de que todos os seus capítulos são curtos demais. Infelizmente, isso sempre ocorre quando tentamos capturar algo que evolui diariamente, e que é tão rico e abundante em oportunidades. Mesmo assim, confio no "princípio 80-20" (*http://en.wikipedia.org/wiki/Pareto_principle*), e acredito sinceramente que este livro é uma tentativa razoável de apresentar os 20% mais interessantes do que você deseja explorar com 80% de seu tempo disponível.

Este livro é curto, mas aborda muitos tópicos. Falando em termos gerais, há mais amplitude que profundidade. Ainda assim, quando a situação permite e o tópico tratado é suficientemente complexo para merecer uma discussão mais detalhada, fazemos alguns mergulhos mais profundos em técnicas interessantes de mineração e análise. Este livro foi escrito de modo que você possa lê-lo de capa a capa, como um guia sobre o trabalho com dados da web social, ou escolhendo individualmente capítulos que lhe interessem. Em outras palavras, ainda que cada capítulo seja projetado como independente, tivemos a atenção especial de introduzir material em uma ordem específica, para que o livro como um todo fosse

uma leitura agradável.

Sites de redes sociais como o Facebook, o Twitter e o LinkedIn passaram de novidades a fenômeno global nos últimos anos. No primeiro quarto de 2010, o Facebook, popular site de rede social, ultrapassou o Google em número de visitas às suas páginas[1], confirmando uma mudança evidente na forma como as pessoas passam seu tempo on-line. Declarar que esse evento indica que a web agora é mais um meio social do que uma ferramenta para pesquisa e informação pode ser um tanto indefensável. Entretanto, esses dados indicam inegavelmente que sites de redes sociais estão satisfazendo alguns desejos humanos muito básicos, em uma escala considerável, e de modos que mecanismos de busca nunca foram projetados para fazer. Redes sociais realmente estão alterando a forma como vivemos nossas vidas dentro e fora da web[2], e permitindo à tecnologia mostrar o que temos de melhor (e às vezes de pior). A explosão das redes sociais é somente uma das formas pelas quais a distância entre o mundo real e o ciberespaço continua a diminuir.

Em termos gerais, cada capítulo deste livro entrelaça elementos da web social com técnicas de mineração, análise e visualização de dados para responder às seguintes questões:

- Quem conhece quem, e quais amigos eles têm em comum?
- Com que frequência determinadas pessoas estão se comunicando?
- Quão simétrica é a comunicação entre as pessoas?
- Quem são as pessoas mais quietas/conversadoras em uma rede?
- Quem são as pessoas mais influentes/populares em uma rede?
- Sobre o que as pessoas estão conversando (e por acaso isso é interessante)?

A resposta a essas questões geralmente conecta duas ou mais pessoas e aponta para um contexto que indica por que existe a conexão. O trabalho envolvido nisso é somente o início de processos analíticos mais complexos, mas você deve iniciar em algum ponto e essa é uma ótima oportunidade, graças às APIs bem projetadas das redes sociais e às bibliotecas open-source.

Podemos dizer que este livro trata a web social[3] como um grafo de pessoas, atividades, eventos, conceitos etc. Líderes do mercado, como o Google e o Facebook, começam a dar preferência a uma terminologia

centrada em grafos, em vez de uma centrada na web, conforme promovem simultaneamente APIs com base em grafos. De fato, Tim Berners-Lee sugeriu que talvez devesse ter utilizado o termo Giant Global Graph (Grafo Global Gigante, *http://dig.csail.mit.edu/breadcrumbs/node/215*, GGG) em vez de World Wide Web (WWW), pois os termos "web" e "grafo" podem ser livremente intercambiados no contexto da definição de uma topologia para a Internet. Se um dia a totalidade da visão original de Tim Berners-Lee será alcançada, ainda não podemos dizer, mas a web que conhecemos está se tornando cada vez mais rica em dados sociais. Quando olharmos para trás, daqui a alguns anos, talvez pareça óbvio que os efeitos de segundo e terceiro níveis criados por uma web inerentemente social foram elementos necessários para alcançarmos uma web verdadeiramente semântica. A distância entre essas duas realidades parece estar diminuindo.

Quem não deve ler este livro?

Atividades como criar do zero seu próprio processador de linguagem natural, avançar muito além do uso típico das bibliotecas de visualização ou construir algo com tecnologia de ponta não estão dentro do escopo deste livro. Você ficará muito desapontado se o comprar interessado em algo do tipo. Porém, simplesmente por não ser realista, nem nossa meta, ter a pretensão de capturar em poucas centenas de páginas o Santo Graal das ferramentas analíticas de texto ou da correspondência de registros, não significa que este livro não permitirá que você alcance soluções razoáveis para problemas complicados, que aplique essas soluções à web social como um domínio, e que se divirta muito nesse processo. Também não significa que ter um interesse significativo por essas fascinantes áreas de pesquisa não seja potencialmente uma ótima ideia a ser considerada. Um livro curto como este não pode fazer muito mais do que despertar seu interesse e fornecer insights suficientes para que você siga em frente e crie algo novo munido de sua recém-descoberta paixão pelo hacking de dados.

Talvez seja óbvio, mas outro item importante que merece destaque é que este livro em geral presume que você esteja conectado à Internet. Este não seria um bom livro para você levar em suas férias a algum lugar remoto, pois contém muitas referências em hiperlinks, e todos os exemplos de

código estão diretamente ligados ao GitHub (*http://github.com*), um repositório Git (*http://git-scm.com*) de caráter social que sempre apresentará o exemplo de código mais atualizado disponível. A esperança é que, fornecendo o código em um ambiente social, estimularemos a colaboração entre pessoas de interesses semelhantes que, como nós, desejam trabalhar juntas para estender os exemplos e explorar problemas interessantes. O que esperamos é que você modifique, estenda e aprimore o código-fonte – talvez até mesmo fazendo alguns novos amigos nesse processo. Fontes facilmente acessíveis de informação on-line, como a documentação das APIs, também têm seus links fornecidos, e presumimos que você preferirá examiná-las on-line a confiar em cópias inevitavelmente desatualizadas neste livro impresso.

O repositório oficial no GitHub que mantém o código-fonte mais recente e corrigido para este livro é *http://github.com/ptwobrussell/Mining-the-Social-Web*. A conta oficial no Twitter para este livro é *@SocialWebMining*.

Este livro também não é recomendado se você busca uma referência que o mantenha atualizado quanto a plataformas de computação distribuída como clusters MySQL compartilhados ou tecnologias NoSQL (*http://en.wikipedia.org/wiki/NoSQL*), como Hadoop ou Cassandra. De fato, utilizamos algumas tecnologias pouco convencionais de armazenamento, como o CouchDB (*http://couchdb.apache.org*) e o Redis (*http://code.google.com/p/redis*), mas sempre dentro do contexto da execução em uma única máquina, e somente por que funcionam bem para o problema em questão. Entretanto, não é difícil portar os exemplos para tecnologias distribuídas desde que você tenha motivação suficiente e necessidade de escalabilidade horizontal. Uma recomendação importante é a de que você domine primeiro o fundamental e prove sua tese em um ambiente um pouco menos complexo antes de migrar para um sistema distribuído inerentemente mais complexo. Então, estará pronto para fazer grandes ajustes em seus algoritmos para que eles funcionem quando o acesso aos dados não for mais local. Uma boa opção a ser investigada, se você deseja seguir essa rota, é o Dumbo (*http://github.com/klbostee/dumbo/wiki/*). Fique atento à conta deste livro no Twitter (*@SocialWebMining*) para acessar exemplos estendidos que envolvem o Dumbo.

Este livro não fornece nenhum conselho sobre as ramificações legais do que pode ser feito com os dados obtidos a partir dos sites de redes sociais,

ainda que sempre busquemos obedecer ao texto e ao espírito dos termos que governam os sites específicos mencionados. Pode parecer desafortunado que muitos dos sites mais populares de redes sociais tenham termos de licença que proíbam o uso de seus dados fora de suas plataformas, mas, no momento, isso é o normal. A maioria desses sites se parece com jardins cercados, mas de seu ponto de vista (e do ponto de vista de seus investidores) muito do valor que eles oferecem depende do controle que têm de suas plataformas e da proteção da privacidade de seus usuários; não é fácil manter esse equilíbrio e isso provavelmente não será solucionado tão cedo.

Uma questão final, e muito menos significativa, é a de que este livro privilegia um ambiente *nix[4], por isso há algumas visualizações que podem ser problemáticas para usuários do Windows. Sempre que tivermos ciência desse problema, forneceremos conselhos quanto a alternativas e "gambiarras", como o uso de um VirtualBox (*http://www.virtualbox.org*) para rodar o exemplo em um ambiente Linux. Felizmente, isso não ocorre com frequência e, quando ocorre, você pode ignorar essas seções e avançar sem qualquer perda significativa do prazer de sua leitura.

Ferramentas e pré-requisitos

Os únicos verdadeiros pré-requisitos deste livro são: você precisa estar suficientemente motivado a aprender um pouco de Python e ter o desejo de realmente se envolver com os dados sociais. Nenhuma das técnicas ou exemplos do livro exige conhecimento prévio significativo de análise de dados, computação de alta performance, sistemas distribuídos, aprendizagem de máquina, ou de qualquer outro elemento específico. Alguns exemplos envolvem construções que você talvez nunca tenha utilizado, como *thread pools* (*http://en.wikipedia.org/wiki/Thread_pool_pattern*), mas não se preocupe – estamos programando em Python. Sua sintaxe intuitiva, seu incrível ecossistema de pacotes para manipulação de dados, e suas principais estruturas, que são praticamente JSON (*http://www.json.org*), fazem com que a linguagem Python seja uma excelente ferramenta de ensino; poderosa, mas de fácil utilização. Em outras situações, utilizamos alguns pacotes que realizam ações consideravelmente avançadas, como o

processamento de linguagem natural, mas abordamos esses tópicos do ponto de vista do uso da tecnologia como um programador de aplicações. Dada a probabilidade de que existam referências semelhantes em outras linguagens de programação, deve ser algo prático portar os exemplos de código, caso você o queira (com sorte, isso é exatamente o que acontecerá no GitHub!). Para além dessa explicação prévia, este livro não procura justificar a preferência pela linguagem Python ou se desculpar por ter feito essa escolha, uma vez que ela é uma ferramenta perfeitamente adequada para a função. Caso você esteja dando seus primeiros passos em programação, ou nunca tenha visto a sintaxe Python, ler rapidamente algumas das próximas páginas deve fornecer-lhe toda a confirmação de que necessita. Há documentação excelente disponível on-line, e o tutorial oficial da linguagem (*http://docs.python.org/tutorial/*) é um ótimo ponto de partida se você está em busca de uma introdução completa.

Este livro procura apresentar uma grande gama de visualizações úteis, mostrando várias ferramentas e kits de ferramentas de visualização, indo dos favoritos dos consumidores, como planilhas eletrônicas, a referências do mercado, como o Graphviz (*http://www.graphviz.org*), e tecnologias HTML5 de ponta (*http://en.wikipedia.org/wiki/HTML5*) como o Protovis (*http://vis.stanford.edu/protovis*). Procuramos apresentar visualizações novas a cada capítulo, mas de uma forma que seja natural e que faça sentido. Você deve sentir-se à vontade com a construção de protótipos leves a partir dessas ferramentas. Dito isso, a maioria das visualizações deste livro é pouco mais do que pequenas mutações de exemplos convencionais ou de projetos que exercitam minimamente as APIs. Assim, desde que você esteja disposto a aprender, não deverá ter dificuldades.

Convenções usadas neste livro

As seguintes convenções tipográficas são utilizadas neste livro:

Itálico

Indica novos termos, URLs, endereços de e-mail, nomes e extensões de arquivos.

`Largura constante`

Indica listagens de programas e é utilizada dentro de parágrafos para indicar elementos de programas, como variáveis ou nomes de funções,

bancos de dados, tipos de dados, variáveis de ambiente, instruções e palavras-chave.

`Constante com negrito`
: Mostra comandos ou outros textos que devem ser digitados literalmente pelo usuário. Também é utilizada ocasionalmente para dar ênfase em listagens de código.

`Constante com itálico`
: Mostra texto que deve ser substituído por valores fornecidos pelo usuário ou valores determinados pelo contexto.


Este ícone significa uma dica, sugestão ou nota geral.


Este ícone indica um aviso ou alerta.

Uso de exemplos de código

A maioria dos exemplos numerados nos capítulos seguintes está disponível para download no GitHub, em *https://github.com/ptwobrussell/Mining-the-Social-Web* – o repositório de código oficial para este livro. Encorajamos você a monitorar esse endereço sempre, para ter acesso a código mais atualizado, assim como a exemplos estendidos pelo autor e pelo restante da comunidade social que trabalha no código.

O intuito deste livro é ajudá-lo a fazer seu trabalho. Em geral, você pode usar o código do livro em seus programas e em sua documentação. Não é necessário entrar em contato conosco para obter permissão, a não ser que você esteja reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que usa diversas partes do código presente neste livro não requer nenhum tipo de permissão. No entanto, para vender ou distribuir um CD-ROM com exemplos extraídos dos livros da O'Reilly é necessário obter permissão. Responder a uma pergunta citando este livro e o código de exemplo não requer permissão. Incorporar uma quantidade significativa de código de exemplo deste livro na documentação de seu produto requer permissão.

Agradecemos, mas não exigimos, a atribuição de créditos. Uma atribuição

geralmente inclui o título, o autor, a editora e o ISBN. Por exemplo, "Mining the Social Web por Matthew A. Russell. Copyright 2011 Matthew Russell, 978-1-449-38834-8".

Se acreditar que o uso de exemplos de código ultrapassa o permitido ou a permissão concedida aqui, sinta-se à vontade para entrar em contato direto com a O'Reilly pelo e-mail *permissions@oreilly.com* ou com a Novatec (*novatec@novatec.com.br*).

Como entrar em contato conosco

Dirija seus comentários e questões sobre este livro à editora, escrevendo para:

novatec@novatec.com.br.

Temos uma página na web para este livro, onde incluímos a lista de erratas, exemplos e qualquer outra informação adicional.

- Página da edição em português:
  http://www.novatec.com.br/livros/mining
- Página da edição original em inglês:
  http://www.oreilly.com/catalog/9781449388348/

Leitores podem solicitar o auxílio do autor por meio do GetSatisfaction em:

http://getsatisfaction.com/oreilly

Leitores também podem registrar tickets acerca dos exemplos de código – bem como de qualquer outro elemento do livro – por meio do sistema de gerenciamento de incidentes (ou de acompanhamento de questões) do GitHub em:

http://github.com/ptwobrussell/Mining-the-Social-Web/issues

Para obter mais informações sobre livros da Novatec, acesse nosso site em:

http://www.novatec.com.br

Agradecimentos

Para dizer o mínimo, escrever um livro técnico exige um enorme sacrifício. No aspecto pessoal, deixei de passar mais tempo do que

gostaria de admitir com minha esposa, Baseeret, e minha filha, Lindsay Belle. Agradeço acima de tudo a vocês duas por me amarem, apesar de minhas ambições de talvez um dia conquistar o mundo (é apenas uma fase e estou sinceramente tentando sair dela – eu juro).

Honestamente, gosto de crer que a soma de nossas decisões nos conduz à posição que ocupamos na vida (especialmente no aspecto profissional), mas ninguém é capaz de completar a jornada sozinho e é uma honra dar crédito a quem merece. Tive a benção de ter a companhia de algumas das pessoas mais inteligentes do mundo enquanto trabalhava neste livro, incluindo um editor técnico inteligente como Mike Loukides, uma equipe de produção talentosa, como o pessoal da O'Reilly, e um grupo enorme de revisores incríveis como todos que me ajudaram a completar este livro. Gostaria de agradecer especialmente a Abe Music, Pete Warden, Tantek Celik, J. Chris Anderson, Salvatore Sanfilippo, Robert Newson, DJ Patil, Chimezie Ogbuji, Tim Golden, Brian Curtin, Raffi Krikorian, Jeff Hammerbacher, Nick Ducoff, e Cameron Marlowe, por revisarem o material e fazerem comentários especialmente úteis que ajudaram significativamente a produzir um melhor resultado. Também gostaria de agradecer a Tim O'Reilly por me permitir analisar alguns de seus dados do Twitter e do Google Buzz nos capítulos 4, 5 e 7; isso definitivamente tornou a leitura desses capítulos muito mais interessante. Seria impossível mencionar todas as outras pessoas que, direta ou indiretamente, moldaram minha vida ou o resultado deste livro.

Por fim, obrigado a você por dar uma chance a este livro. Se você está lendo este texto, está ao menos pensando em comprar um exemplar. Se comprar, provavelmente encontrará algo de errado, apesar de todo meu esforço; ainda assim, acredito sinceramente que, mesmo com os inevitáveis erros, você gostará de passar algumas noites ou fins de semana acompanhado do texto, e acabará por aprender algo novo no processo.

---

1 Veja o parágrafo de abertura do capítulo 9.

2 Mark Zuckerberg, o criador do Facebook, foi nomeado a personalidade do ano de 2010 pela revista Time (*http://www.time.com/time/specials/packages/article/0,28804,2036683_2037183_2037185,00.html*).

3 Consulte *http://journal.planetwork.net/article.php?lab=reed0704* para ter acesso a outra perspectiva sobre a web social, concentrando-se em identidades digitais.

4 *nix é um termo utilizado para se referir a um ambiente Linux/Unix, basicamente

sinônimo de ambiente não-Windows.

CAPÍTULO 1

Introdução: hacking de dados do Twitter

Ainda que fosse possível iniciar este capítulo com uma discussão prolongada sobre as APIs específicas das redes sociais, sobre bancos de dados schemaless ou NoSQL[1], ou sobre muitos outros tópicos, vamos, em vez disso, mergulhar diretamente em alguns exemplos introdutórios que ilustram como é simples coletar e analisar dados da web social. Este capítulo é um tutorial rápido que busca motivá-lo e fazer com que pense em algumas das questões que o restante do livro abordará com maiores detalhes. Iniciaremos preparando nosso ambiente de desenvolvimento e, então, rapidamente avançaremos para a coleta e análise de alguns dados do Twitter.

Instalação de ferramentas de desenvolvimento em Python

O código de exemplo deste livro é escrito em Python, assim, se você já tem uma versão recente da linguagem e do `easy_install` em seu sistema, obviamente sabe o que fazer e pode provavelmente ignorar o restante desta seção. Se ainda não tem a Python instalada, a má notícia é que você provavelmente ainda não deve ser um hacker nessa linguagem. Mas não se preocupe, você se tornará um em breve; a Python é capaz de fazer isso com as pessoas, pois é muito fácil aprendê-la na prática. Usuários de todas as plataformas podem encontrar instruções para download e instalação da Python em *http://www.python.org/download/*, mas é altamente recomendado que usuários do Windows instalem o ActivePython (*http://www.activestate.com/activepython*), que automaticamente adiciona a Python ao seu path no prompt de comando do Windows (daqui em diante chamado de "terminal") e vem com o `easy_install`, que discutiremos em breve. Os exemplos neste livro foram criados e testados com o branch mais recente da Python 2.7, mas também devem funcionar com versões atualizadas da linguagem. Quando da

redação deste livro, a Python Version 2 ainda era o status quo na comunidade Python (*http://wiki.python.org/moin/Python2orPython3*), e é recomendado que você a utilize, a menos que esteja confiante em que todas as dependências de que necessita foram portadas para a Version 3, estando disposto a realizar o debug de eventuais idiossincrasias envolvidas na mudança.

Uma vez instalada a linguagem, você deve ser capaz de digitar `python` no terminal para iniciar um interpretador. Experimente realizar o que mostramos no exemplo 1.1.

Exemplo 1.1 – Sua primeira sessão no interpretador Python

```
>>> print "Hello World"
Hello World
>>> #isto é um comentário
...
>>> for i in range(0,10): # um loop
... print i, # a vírgula omite quebras de linha
...
0 1 2 3 4 5 6 7 8 9
>>> numbers = [ i for i in range(0,10) ] # uma compreensão de
lista
>>> print numbers
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> if 10 in numbers: # lógica condicional
... print True
... else:
... print False
...
False
```

Outra ferramenta que você deverá ter em mãos é a `easy_install`[2], semelhante a um gerenciador de pacotes em sistemas Linux, que permite a você instalar sem dificuldades pacotes Python, em vez de ter de baixá-los, compilá-los e instalá-los a partir do código-fonte. Você pode baixar a versão mais recente do `easy_install` em *http://pypi.python.org/pypi/setuptools*, onde também encontrará instruções específicas para cada plataforma. Em termos gerais, usuários *nix deverão utilizar `sudo easy_install` para que os módulos sejam gravados nos diretórios da instalação global da Python. Presume-se que usuários do Windows sigam nosso conselho e utilizem o ActivePython, que inclui

automaticamente o `easy_install` como parte da instalação.

Usuários do Windows também podem verificar o post "Installing easy_install... could be easier" (*http://blog.sadphaeton.com/2009/01/20/python-development-windows-part-2-installing-easyinstallcould-be-easier.html*). Esse texto discute alguns dos problemas comuns relacionados à compilação do código C e que podem ser encontrados ao executar o `easy_install`.

Estando devidamente configurado o `easy_install`, você deverá ser capaz de executar o comando seguinte para instalar o NetworkX – pacote que utilizaremos durante todo o livro para criar e analisar grafos – e observar uma saída semelhante:

```
$ easy_install networkx
Searching for networkx
...saída truncada...
Finished processing dependencies for networkx
```

Com o NetworkX instalado, talvez você pense que basta importá-lo do interpretador para iniciar seu trabalho, mas ocasionalmente alguns pacotes poderão surpreendê-lo. Por exemplo, suponha que ocorra o seguinte:

```
>>> import networkx
Traceback (most recent call last):
... saída truncada ...
ImportError: No module named numpy
```

Sempre que ocorre um `ImportError`, isso quer dizer que está faltando um pacote. Nesta ilustração, o módulo que instalamos, `networkx`, tem uma dependência faltante de nome `numpy` (*http://numpy.scipy.org*), uma coleção altamente otimizada de ferramentas para computação científica. Geralmente uma nova invocação do `easy_install` será capaz de corrigir o problema, o que pode ser feito aqui. Apenas feche o interpretador e instale a dependência, digitando `easy_install numpy` no terminal:

```
$ easy_install numpy
Searching for numpy
...saída truncada...
Finished processing dependencies for numpy
```

Agora que a `numpy` está instalada, você deve ser capaz de abrir um novo interpretador, `import networkx`, e utilizá-lo para criar grafos (exemplo 1.2).

Exemplo 1.2 – Uso do NetworkX para criar um grafo de nós (nodes) e arestas (edges)

```
>>> import networkx
>>> g=networkx.Graph()
>>> g.add_edge(1,2)
>>> g.add_node("spam")
>>> print g.nodes()
[1, 2, 'spam']
>>> print g.edges()
[(1, 2)]
```

Com isso, você tem instaladas algumas das principais ferramentas de desenvolvimento em Python, e está pronto para avançar a tarefas mais interessantes. Se a maioria do que mostramos nesta seção foi novidade para você, pode valer a pena revisar o tutorial on-line oficial da Python (*http://docs.python.org/tutorial/*) antes de avançar.

Coleta e manipulação de dados do Twitter

Mesmo sendo extremamente improvável, caso você ainda não conheça o Twitter, saiba que ele é um serviço de microblogging de caráter altamente social que permite que você poste mensagens com 140 caracteres ou menos; essas mensagens são chamadas de *tweets*. Diferentemente de redes sociais como o Facebook e o LinkedIn, em que as conexões são bidirecionais, o Twitter tem uma infraestrutura de servidor assimétrica, composta de "amigos" e "seguidores". Presumindo que você tenha uma conta no Twitter, seus amigos são as contas que você está seguindo, e seus seguidores, aquelas que seguem a sua. Ainda que você possa seguir todos os usuários que são seus seguidores, isso geralmente não ocorre, pois você deseja que o Histórico (Timeline[3]) de sua Página Inicial inclua tweets apenas das contas que você considera interessantes. O Twitter é um fenômeno importante do ponto de vista de seu número incrivelmente alto de usuários, assim como de seu uso como dispositivo de marketing e como camada de transporte para serviços de troca de mensagens de terceiros. Ele oferece uma ampla coleção de APIs e, ainda que você possa utilizar muitas delas sem se registrar, é muito mais interessante construir e minerar sua própria rede. Não tenha pressa e leia os termos de serviço do Twitter (*http://twitter.com/tos*), a documentação de sua API (*http://apiwiki.twitter.com*), e suas regras (*http://twitter.com/apirules*), que permitem que você faça praticamente tudo que seria de se esperar com dados do Twitter antes de iniciar um desenvolvimento mais avançado. O

restante deste livro presume que você tenha uma conta no Twitter e um número suficiente de amigos/seguidores dos quais possa minerar dados.

A conta oficial no Twitter para este livro é *@SocialWebMining*.

Experimentando com a API do Twitter

Uma biblioteca simples para a API web do Twitter está disponível por meio de um pacote chamado `twitter` (*http://github.com/sixohsix/twitter*) que pode ser instalado com o `easy_install` de forma normal:

```
$ easy_install twitter
Searching for twitter
...saída truncada...
Finished processing dependencies for twitter
```

O pacote também inclui um prático utilitário de linha de comando e um bot IRC, assim, depois de instalar o módulo, você deve ser capaz de simplesmente digitar `twitter` em um shell para ver uma tela explicando como utilizar o utilitário de linha de comando. Entretanto, nos concentraremos no trabalho dentro do interpretador Python interativo. Veremos alguns exemplos, mas note: você pode sempre consultar a documentação executando `pydoc` no terminal. Usuários *nix podem simplesmente digitar `pydoc twitter.Twitter` para visualizar a documentação sobre a classe Twitter, enquanto usuários do Windows têm de digitar `python -mpydoc twitter.Twitter`. Se você notar que precisa consultar frequentemente a documentação em busca de determinados módulos, pode escolher passar a opção `-w` para `pydoc` e produzir uma página HTML que poderá ser salva ou marcada como favorita em seu navegador. Saiba também que executar `pydoc` em um módulo ou classe exibe a documentação embutida da mesma forma que executar `help()` no interpretador. Experimente digitar `help(twitter.Twitter)` no interpretador e veja o resultado.

Feito isso, vamos agora descobrir o que as pessoas estão conversando, inspecionando as *trends*[4] disponíveis por meio da API de busca do Twitter (*http://dev.twitter.com/doc/get/search*). Inicie o interpretador e faça uma pesquisa. Tente acompanhar o exemplo 1.3 e utilize a função `help()`, quando necessário, para responder às suas dúvidas antes de avançar.

Exemplo 1.3 – Recuperação das trends de pesquisa do Twitter

```
>>> import twitter
>>> twitter_search = twitter.Twitter(domain="search.twitter.com")
>>> trends = twitter_search.trends()
>>> [ trend['name'] for trend in trends['trends'] ]
[u'#ZodiacFacts', u'#nowplaying', u'#ItsOverWhen',
u'#Christoferdrew',
u'Justin Bieber', u'#WhatwouldItBeLike', u'#Sagittarius', u'SNL',
u'#SurveySays',
u'#iDoit2']
```

Como você deve estar se perguntando, o padrão para uso do módulo `twitter` é simples e previsível: instancie a classe `Twitter` com um URL de base e, então, invoque métodos no objeto que correspondam a contextos do URL. Por exemplo, `twitter_search.trends()` inicia uma chamada HTTP para fazer o GET de *http://search.twitter.com/trends.json*, que pode ser digitada em seu navegador para obter o mesmo conjunto de resultados. Esteja ciente de que este capítulo foi originalmente escrito em uma noite de sábado, por isso não é coincidência que o trend SNL (Saturday Night Live, popular programa de comédia apresentado nos Estados Unidos nas noites de sábado) surja na lista. Este seria um bom momento para marcar como favorita a documentação oficial da API do Twitter (*http://dev.twitter.com/doc*), uma vez que você a consultará com frequência.

Dado que o SNL está entre as tendências, o próximo passo lógico pode ser pegar alguns resultados de busca sobre ele, utilizando a API de busca para pesquisar por tweets que contenham esse texto, e então reproduzi-los de forma legível como uma estrutura JSON (*http://json.org*) (exemplo 1.4).

Exemplo 1.4 – Paginação de resultados de busca do Twitter

```
>>> search_results = []
>>> for page in range(1,6):
... search_results.append(twitter_search.search(q="SNL", rpp=100,
page=page))
```

O código busca e armazena cinco lotes consecutivos (`pages`) de resultados para uma consulta (`q`) do termo SNL, com 100 resultados por página (`rpp`). Também vale a pena observar que a consulta REST[5] equivalente, que executamos no loop, tem a seguinte forma *http://search.twitter.com/search.json?&q=SNL&rpp=100&page=1*. O

mapeamento simples entre a API REST e o módulo `twitter` faz com que seja simples escrever código Python que interaja com serviços do Twitter. Depois de executar a busca, a lista `search_results` contém cinco objetos, sendo que cada um é um lote de 100 resultados. Você pode imprimir esses resultados de forma legível para inspeção utilizando o pacote `json` que vem integrado na Python Version 2.6 (exemplo 1.5).

Exemplo 1.5 – Impressão de dados do Twitter como JSON

```
>>> import json
>>> print json.dumps(search_results, sort_keys=True, indent=1)
[
 {
  "completed_in": 0.088122000000000006,
  "max_id": 11966285265,
  "next_page": "?page=2&max_id=11966285265&rpp=100&q=SNL",
  "page": 1,
  "query": "SNL",
  "refresh_url": "?since_id=11966285265&q=SNL",
  "results": [
   {
    "created_at": "Sun, 11 Apr 2010 01:34:52 +0000",
    "from_user": "bieber_luv2",
    "from_user_id": 106998169,
    "geo": null,
    "id": 11966285265,
    "iso_language_code": "en",
    "metadata": {
     "result_type": "recent"
    },
    "profile_image_url":
"http://a1.twimg.com/profile_images/809471978/DSC00522...",
    "source": "&lt;a
href=&quot;http://twitter.com/&quot;&gt;web&lt;/a&gt;",
    "text": " ...truncated... im nt gonna go to sleep happy
unless i see @justin...",
    "to_user_id": null
   }
            ... saída truncada - mais 99 tweets ...
  ],
  "results_per_page": 100,
  "since_id": 0
 },
```

```
    ... saída truncada - mais 4 páginas ...
]
```

Saiba que, até o final de 2010, o campo `from_user_id`, presente em cada resultado de busca, não corresponde ao verdadeiro id do autor do tweet no Twitter. Consulte o Issue #214 da API do Twitter (*http://code.google.com/p/twitter-api/issues/detail?id=214*) para mais detalhes. Esse defeito não afeta o código de exemplo deste capítulo, mas é importante conhecê-lo, caso você esteja experimentando com seu código (o que é altamente encorajado).

Apenas mais adiante no livro é que examinaremos muitos dos detalhes dessa consulta (capítulo 5). A observação mais importante neste momento é que os tweets são indicados por `results` na resposta. Podemos transformar o texto dos 500 tweets em uma lista com a seguinte abordagem. O exemplo 1.6 ilustra uma compreensão de lista dupla, endentada de modo a ilustrar que sua estrutura subjacente não é nada mais do que um loop aninhado.

Exemplo 1.6 – Uma simples compreensão de lista em Python

```
>>> tweets = [ r['text'] \
... for result in search_results \
... for r in result['results'] ]
```

Compreensões de lista são utilizadas com frequência neste livro. Ainda que possam parecer confusas se escritas em uma única linha, exibi-las como loops aninhados esclarece seu significado. O resultado de `tweets` neste caso específico é equivalente a definir uma lista vazia, `tweets`, e invocar `tweets.append(r['text'])` no mesmo tipo de loop aninhado exibido aqui. Consulte a seção "Data Structures" (*http://docs.python.org/tutorial/datastructures.html*) no tutorial oficial da Python para maiores detalhes. Compreensões de lista são particularmente poderosas, pois geralmente resultam em ganhos significativos de desempenho quando comparadas a listas aninhadas, e fornecem uma sintaxe intuitiva (desde que você esteja habituado a ela), ainda que concisa.

Análise de frequência e diversidade léxica

Uma das medições mais intuitivas que podem ser aplicadas a um texto não estruturado é uma métrica chamada diversidade léxica. Na prática, ela é uma expressão do número de *tokens*[6] individuais que existem no texto, dividido pelo número total de tokens; são métricas elementares, mas importantes, mesmo quando consideradas independentemente. A diversidade léxica pode ser computada como mostra o exemplo 1.7.

Exemplo 1.7 – Cálculo da diversidade léxica para tweets

```
>>> words = []
>>> for t in tweets:
... words += [ w for w in t.split() ]
...
>>> len(words) # palavras no total
7238
>>> len(set(words)) # palavras únicas
1636
>>> 1.0*len(set(words))/len(words) # diversidade léxica
0.22602928985907708
>>> 1.0*sum([ len(t.split()) for t in tweets ])/len(tweets) #
média de palavras por tweet
14.476000000000001
```

Em versões anteriores à Python 3.0, o operador de divisão aplica a função `floor` e retorna um valor inteiro (a menos que um dos operandos seja um valor em ponto flutuante). Multiplique o numerador ou o denominador por 1.0 para evitar erros de truncamento.

Uma forma de interpretar uma diversidade léxica de valor 0,23 seria dizer que cerca de uma em cada quatro palavras nos tweets agregados é única. Dado que o número médio de palavras em um tweet é 14, isso significa que temos mais de 3 palavras únicas por tweet. Sem introduzir nenhuma informação adicional, isso pode ser interpretado como significando que cada tweet carrega cerca de 20% de informações únicas. Nesse ponto, o que seria interessante saber é qual o nível de "ruído" nos tweets provocado pelas abreviações incomuns que os usuários podem empregar para permanecerem dentro do limite de 140 caracteres, assim como quais os termos mais frequentes e os mais incomuns utilizados. Ter acesso a uma distribuição das palavras e à sua frequência seria útil. Ainda que não seja difícil computar esses dados, seria preferível instalar uma ferramenta que oferecesse de forma integrada a distribuição de frequência e muitas outras ferramentas para análise de texto.

O Natural Language Toolkit (*http://www.nltk.org*), ou NLTK, é um módulo popular que utilizaremos em todo este livro e que traz consigo uma grande gama de ferramentas para diversos tipos de análise de texto, incluindo cálculo de métricas comuns, extração de informações e processamento de linguagem natural (*natural language processing*, ou NLP). Ainda que o NLTK não seja necessariamente o que há de mais avançado quando comparado às opções comerciais e aos recursos do meio acadêmico, ele oferece uma estrutura sólida e ampla – especialmente se

esta é sua primeira experiência tentando processar linguagem natural. Caso seu projeto seja tão sofisticado que a qualidade e a eficiência do NLTK não satisfaçam suas necessidades, você terá aproximadamente três opções, dependendo da quantidade de tempo e dinheiro que estiver disposto a investir: explorar as opções de código aberto em busca de uma alternativa mais adequada realizando experimentos e testes de desempenho comparativos, pesquisar textos técnicos e produzir sua própria biblioteca, ou adquirir um produto comercial. Nenhuma dessas opções é barata (presumindo que você acredite que tempo é dinheiro) ou fácil.

O NLTK pode ser instalado da forma habitual utilizando o `easy_install`, mas você terá de reiniciar o interpretador para poder utilizá-lo. Você pode utilizar o módulo `cPickle` para salvar seus dados antes de sair de sua sessão atual (Exemplo 1.8).

Exemplo 1.8 – Salvando os dados

```
>>> f = open("myData.pickle", "wb")
>>> import cPickle
>>> cPickle.dump(words, f)
>>> f.close()
>>>
$ easy_install nltk
Searching for nltk
...saída truncada...
Finished processing dependencies for nltk
```

Se você encontrar um problema do tipo "ImportError: No module named yaml" quando tentar importar o nltk, pode executar `easy_install pyYaml` para solucionar a questão.

Depois de instalar o NLTK, seria interessante visitar seu site oficial (*http://www.nltk.org*), onde você poderá analisar sua documentação, incluindo o texto completo de Natural Language Processing with Python (O'Reilly), escrito por Steven Bird, Ewan Klein, e Edward Loper, uma referência definitiva sobre o NLTK.

Sobre o que as pessoas estão conversando neste exato instante?

Dentre as razões mais significativas que justificam a mineração de dados do Twitter está tentarmos descobrir sobre o que as pessoas estão conversando agora. Uma das técnicas mais simples que você pode aplicar para responder a essa questão é a análise básica de frequência. O NLTK

simplifica essa tarefa fornecendo uma API para análise de frequência, por isso vamos deixar que ele cuide desses detalhes. O exemplo 1.9 demonstra o que descobrimos ao criar uma distribuição de frequência e examinar os 50 termos mais e menos frequentes.

Exemplo 1.9 – Uso do NLTK para realizar análise básica de frequência

```
>>> import nltk
>>> import cPickle
>>> words = cPickle.load(open("myData.pickle"))
>>> freq_dist = nltk.FreqDist(words)
>>> freq_dist.keys()[:50] # 50 tokens mais frequentes
[u'snl', u'on', u'rt', u'is', u'to', u'i', u'watch', u'justin',
u'@justinbieber',
u'be', u'the', u'tonight', u'gonna', u'at', u'in', u'bieber',
u'and', u'you',
u'watching', u'tina', u'for', u'a', u'wait', u'fey', u'of',
u'@justinbieber:',
u'if', u'with', u'so', u"can't", u'who', u'great', u'it',
u'going',
u'im', u':)', u'snl...', u'2nite...', u'are', u'cant', u'dress',
u'rehearsal',
u'see', u'that', u'what', u'but', u'tonight!', u':d', u'2',
u'will']
>>> freq_dist.keys()[-50:] # 50 tokens menos frequentes
[u'what?!', u'whens', u'where', u'while', u'white', u'whoever',
u'whoooo!!!!',
u'whose', u'wiating', u'wii', u'wiig', u'win...', u'wink.',
u'wknd.', u'wohh', u'won',
 u'wonder', u'wondering', u'wootwoot!', u'worked', u'worth',
u'xo.', u'xx', u'ya',
u'ya&lt;3miranda', u'yay', u'yay!', u'ya\u2665', u'yea', u'yea.',
u'yeaa', u'yeah!',
u'yeah.', u'yeahhh.', u'yes,', u'yes;)', u'yess', u'yess,',
u'you!!!!!',
u"you'll", u'you+snl=', u'you,', u'youll', u'youtube??',
u'youu&lt;3',
u'youuuuu', u'yum', u'yumyum', u'~', u'\xac\xac']
```

A Python 2.7 adicionou uma classe collections.Counter (*http://docs.python.org/library/collections.html#collections.Counter*) que facilita operações de contagem e que pode ser útil se você estiver em uma situação em que não possa instalar facilmente o NLTK, ou se simplesmente quiser realizar testes com as mais recentes e melhores classes da biblioteca-padrão da Python.

Um rápido exame dos resultados do exemplo 1.9 mostra que encontramos

informações muito mais úteis nos tokens mais frequentes do que nos tokens não frequentes. Ainda que seja necessário certo trabalho para fazer com que uma máquina reconheça isso, os tokens frequentes se referem a entidades, como pessoas, dados e atividades, enquanto os termos não frequentes representam basicamente ruído a partir do qual nenhuma conclusão significativa pode ser formulada.

O primeiro ponto que você pode ter notado sobre os tokens mais frequentes é que "snl" está no topo da lista. Dado que ele é a base da consulta original, isso não deve surpreendê-lo. O cenário fica mais interessante quando avançamos para os próximos tokens: aparentemente, há muita conversa sobre um rapaz chamado Justin Bieber, como fica evidente pelos tokens `@justinbieber`, `justin` e `bieber`. Qualquer pessoa que conheça o SNL também sabe que a presença dos tokens "tina" e "fey" não é coincidência, dada a longa participação de Tina Fey no programa. Com sorte, não deve ser tão difícil (como ser humano) visualizar os tokens e concluir que Justin Bieber é um rapaz popular e que muitas pessoas estavam empolgadas com sua participação no programa na noite de sábado em que a consulta foi executada.

Você deve estar se perguntando: "E daí? Eu mesmo poderia pesquisar alguns tweets e deduzir essas informações." Ainda que isso possa ser verdade, por acaso você gostaria de fazê-lo 24 horas por dia, sete dias por semana; ou de pagar a alguém para que o fizesse ininterruptamente? E se você estivesse trabalhando em um domínio diferente que não fosse tão adequado à análise de exemplos randômicos de breves comentários? O que queremos dizer é que a análise de frequência é uma ferramenta muito simples, ainda que poderosa, que não deve ser desconsiderada simplesmente por sua simplicidade ou obviedade. Assim, uma lição preliminar que aprendemos aqui é a de que a aplicação de uma técnica simples pode produzir ótimos resultados no sentido de respondermos à questão: "Sobre o que as pessoas estão conversando agora?"

Como observação final, a presença de "rt" também é uma pista muito importante sobre a natureza das conversas que estão ocorrendo. O token `RT` é um símbolo especial, frequentemente anexado antes da mensagem para indicar que você está fazendo um *retweet* em nome de alguém. Dada a frequência desse token, é razoável deduzir que houve uma grande quantidade de tweets duplicados, ou praticamente duplicados,

envolvendo o assunto de que estamos tratando. De fato, essa observação é a base de nossa próxima análise.

O token RT pode ser pré-anexado a uma mensagem para indicar que ela está sendo repassada, como um "retweet" no linguajar do Twitter. Por exemplo, um tweet "RT *@SocialWebMining* Justin Bieber is on SNL 2nite. w00t?!?" indicaria que o remetente está retransmitindo informações obtidas por meio do usuário *@SocialWebMining* (*http://twitter.com/SocialWebMining*). Uma forma equivalente do retweet seria "Justin Bieber is on SNL 2nite. w00t?!? Ummm...(via *@SocialWebMining*)".

Extração de relacionamentos a partir dos tweets

Como a web social trata principalmente das conexões entre pessoas no mundo real, um grafo é um formato muito conveniente para armazenar dados. Vamos utilizar o NetworkX para criar um grafo conectando usuários do Twitter que fizeram retweets de informações. Incluiremos direcionalidade no grafo para indicar a direção na qual a informação está fluindo. Assim, nosso grafo seria mais corretamente chamado de dígrafo, ou grafo orientado (*http://en.wikipedia.org/wiki/Directed_graph*). Ainda que as APIs do Twitter ofereçam algumas capacidades que determinam e analisam os status que foram transmitidos por retweets, elas não são adequadas ao nosso uso, pois teríamos de fazer muitas chamadas de ida e volta, o que seria um desperdício do limite de chamadas à API incluído em nossa quota.

Quando da redação deste livro, o Twitter impunha um limite de 350 chamadas à API por hora para requisições autenticadas; requisições anônimas são limitadas a 150 chamadas por hora. Você pode ler mais sobre dados específicos dessa natureza em *http://dev.twitter.com/pages/rate-limiting*. Nos capítulos 4 e 5, discutiremos técnicas para que você obtenha o máximo dentro desses limites, além de outras opções para coleta de dados.

Além disso, podemos utilizar dicas apresentadas nos próprios tweets para extrair de forma confiável informações de retweets com uma simples expressão regular. Por convenção, os nomes de usuário no Twitter iniciam com o símbolo @ e podem incluir somente letras, números e sublinhados. Assim, dadas as convenções para retweets, temos de buscar somente os seguintes padrões:

- `RT` seguido por um nome de usuário;
- `via` seguido por um nome de usuário.

Ainda que o capítulo 5 introduza um módulo especificamente projetado para realizar o parsing de entidades a partir dos tweets, o exemplo 1.10 demonstra que você pode utilizar o módulo `re` para compilar[7] um padrão e extrair a origem de um tweet sem dificuldades e sem precisar de nenhuma biblioteca especial.

Exemplo 1.10 – Uso de expressões regulares para encontrar retweets

```
>>> import re
>>> rt_patterns = re.compile(r"(RT|via)((?:\b\W*@\w+)+)",
re.IGNORECASE)
>>> example_tweets = ["RT @SocialWebMining Justin Bieber is on
SNL 2nite. w00t?!?",
... "Justin Bieber is on SNL 2nite. w00t?!? (via
@SocialWebMining)"]
>>> for t in example_tweets:
... rt_patterns.findall(t)
...
[('RT', ' @SocialWebMining')]
[('via', ' @SocialWebMining')]
```

Caso não seja óbvio, a chamada a `findall` retorna uma lista de tuplas[8] na qual cada uma contém o texto correspondente, ou uma string vazia para cada grupo no padrão. Note que a regex deixa um espaço em branco à esquerda nas entidades extraídas, mas isso pode ser facilmente corrigido com uma chamada à `strip()`, como demonstra o exemplo 1.11. Como nenhum dos tweets de exemplo contém ambos os grupos encerrados nas expressões entre parênteses, uma string está vazia em cada uma das tuplas.

Expressões regulares são um conceito básico de programação cuja explanação está fora do escopo deste livro. A documentação do módulo re (*http://docs.python.org/library/re.html*) é um ótimo ponto de partida para aprender mais sobre elas, e você pode sempre consultar o clássico *Mastering Regular Expressions* (*http://oreilly.com/catalog/9780596528126/*), da O'Reilly, escrito por Jeffrey E.F. Friedl, se quiser aprender até mais do que o necessário sobre o assunto.

Uma vez que a estrutura de dados do tweet, como retornada pela API, fornece o nome de usuário daquele que faz o tweet, e que acabamos de descobrir como podemos extrair o responsável por um retweet, é simples carregar essa informação em um grafo do NetworkX. Vamos criar um grafo em que os nós representem nomes de usuários e uma aresta direcionada (ou seta) entre dois nós indique que há um relacionamento de retweet. A aresta em si carregará um *payload*[9] com o ID do tweet e seu texto em si.

O exemplo 1.11 demonstra o processo de geração de tal grafo. Os passos básicos envolvidos generalizam uma rotina que extrai nomes de usuários em retweets, organiza as páginas dos tweets em uma lista plana, para facilitar seu processamento em um loop, e, por fim, itera os tweets e adiciona arestas a um grafo. Mesmo sabendo que geraremos uma imagem desse grafo mais adiante, vale notar que você pode obter muitos insights

valiosos simplesmente analisando as características dos grafos sem necessariamente visualizá-los.

Exemplo 1.11 – Criação e análise de um grafo descrevendo quem fez o retweet de quem

```
>>> import networkx as nx
>>> import re
>>> g = nx.DiGraph()
>>>
>>> all_tweets = [ tweet
... for page in search_results
... for tweet in page["results"] ]
>>>
>>> def get_rt_sources(tweet):
... rt_patterns = re.compile(r"(RT|via)((?:\b\W*@\w+)+)",
re.IGNORECASE)
... return [ source.strip()
... for tuple in rt_patterns.findall(tweet)
... for source in tuple
... if source not in ("RT", "via") ]
...
>>> for tweet in all_tweets:
... rt_sources = get_rt_sources(tweet["text"])
... if not rt_sources: continue
... for rt_source in rt_sources:
... g.add_edge(rt_source, tweet["from_user"], {"tweet_id" :
tweet["id"]})
...
>>> g.number_of_nodes()
160
>>> g.number_of_edges()
125
>>> g.edges(data=True)[0]
(u'@ericastolte', u'bonitasworld', {'tweet_id': 11965974697L})
>>> len(nx.connected_components(g.to_undirected()))
37
>>> sorted(nx.degree(g))
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 6, 6, 9, 37]
```

As operações integradas que o NetworkX oferece são um ótimo ponto de partida para formular conclusões a partir dos dados, mas é importante considerar que estamos analisando apenas uma pequena amostra das conversas totais que ocorrem no Twitter sobre o SNL – 500 tweets de potencialmente dezenas de milhares (ou mais). Por exemplo, o número de nós no grafo nos diz que, dos 500 tweets, houve 160 usuários envolvidos em relacionamentos de retweets, com 125 arestas conectando esses nós. A proporção de 160/125 (aproximadamente 1,28) é uma pista importante que nos diz que o grau médio (*http://en.wikipedia.org/wiki/Degree_(graph_theory)*) de um nó é aproximadamente um – o que significa que ainda que alguns nós estejam conectados a mais de um nó, a média é de aproximadamente uma conexão por nó.

A chamada a `connected_components` mostra que o grafo consiste de 37 subgrafos e não está totalmente conectado. A saída de `degree` pode parecer um tanto obscura a princípio, mas confirma algo que já tínhamos notado: pense nela como uma forma de compreender a essência de quão bem conectados estão os nós do grafo sem que para isso tenhamos de renderizá-lo de fato. Neste caso, a maioria dos valores é 1, o que significa que todos esses nós têm um grau de 1 e estão conectados somente a um outro nó no grafo. Alguns valores estão entre 2 e 9, indicando que esses nós estão conectados a 2 e 9 outros nós. O valor mais alto é um grau de 37. Em essência, isso demonstra que o grafo é composto principalmente de nós separados, mas que há um nó altamente conectado. A figura 1.1 ilustra a distribuição de graus como um gráfico de coluna. A linha de tendência mostra que a distribuição acompanha com precisão uma Lei de Potência (Power Law, *http://en.wikipedia.org/wiki/Power_law*) e tem uma cauda longa, ou pesada. Ainda que as características de distribuições com caudas longas não sejam tratadas com rigor neste livro, você perceberá que muitas das distribuições que encontraremos exibem essa propriedade, e encorajamos você a tomar a iniciativa e pesquisar mais, se estiver interessado. Um bom ponto de partida é a Lei de Zipf (Zipf's law, *http://en.wikipedia.org/wiki/Zipf's_law*).

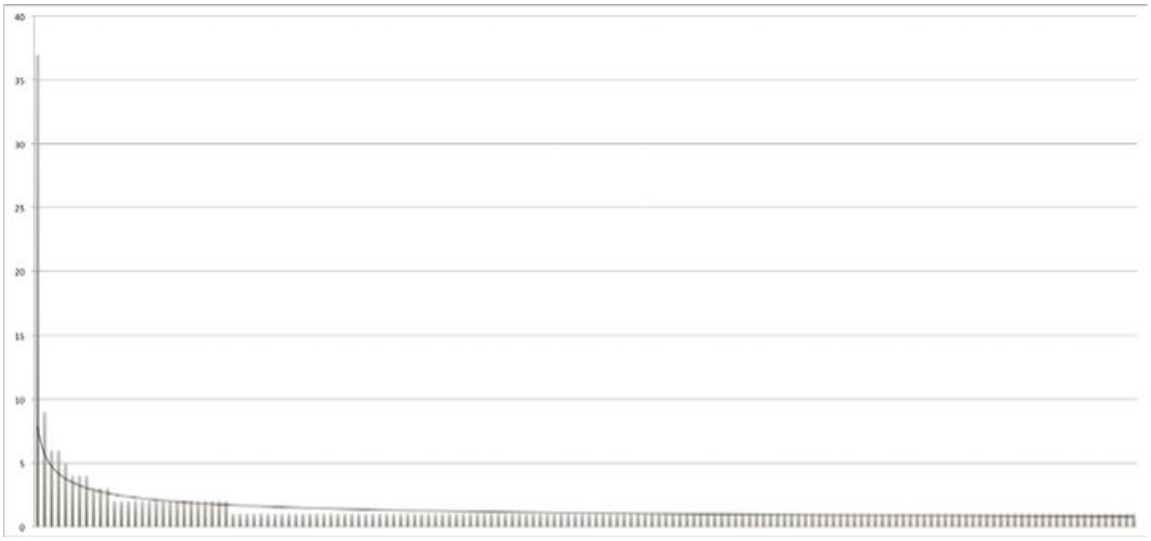


Figura 1.1 – Uma distribuição ilustrando o grau de cada nó no grafo, fornecendo informações quanto à sua conectividade.

Ainda dedicaremos muito tempo neste livro ao uso de heurísticas automatizáveis para formular conclusões a partir dos dados. Este capítulo pretende ser simplesmente uma introdução, estimulando seu interesse e fazendo com que você pense em como pode explorar os dados com os recursos mostrados. Entretanto, antes de concluirmos, vamos visualizar o gráfico apenas para termos certeza de que nossa intuição nos está conduzindo na direção certa.

Visualização de grafos de tweets

O Graphviz é uma referência na comunidade de visualização. Esta seção apresenta uma abordagem possível para visualização de grafos com dados de tweets: sua exportação para a linguagem DOT (*http://www.graphviz.org/doc/info/lang.html*), um simples formato com base em texto que o Graphviz aceita. Você pode fazer o download de binários do Graphviz para todas as plataformas em seu site oficial (*http://www.graphviz.org*); sua instalação é simples independentemente da plataforma escolhida. Uma vez instalado o Graphviz, usuários *nix devem ser capazes de utilizar o comando `easy_install pygraphviz` sem dificuldades, satisfazendo à dependência `PyGraphviz` (*http://networkx.lanl.gov/pygraphviz/*) que o NetworkX requer para emitir DOT. Usuários do Windows provavelmente terão dificuldades na instalação do PyGraphviz[10], mas isso não deve ser um problema, uma vez que não é difícil formular algumas linhas de código para gerar a saída em

linguagem DOT de que necessitamos nesta seção.

O exemplo 1.12 ilustra uma abordagem que funciona em ambas as plataformas.

Exemplo 1.12 – Gerar a saída em linguagem DOT não é difícil, independentemente da plataforma

```
OUT = "snl_search_results.dot"
try:
    nx.drawing.write_dot(g, OUT)
except ImportError, e:
# Ajuda para usuários do Windows:
# Este não se trata de um método de propósito geral, mas é
representativo da
# mesma saída que write_dot fornece para este grafo, se instalado
e implementado
dot = ['"%s" -> "%s" [tweet_id=%s]' % (n1, n2, g[n1][n2]
['tweet_id']) \
    for n1, n2 in g.edges()]
f = open(OUT, 'w')
f.write('strict digraph {\n%s\n}' % (';\n'.join(dot),))
f.close()
```

A saída em DOT gerada tem a forma mostrada no exemplo 1.13.

Exemplo 1.13 – Exemplo de saída em linguagem DOT

```
strict digraph {
"@ericastolte" -> "bonitasworld" [tweet_id=11965974697];
"@mpcoelho" -> "Lil_Amaral" [tweet_id=11965954427];
"@BieberBelle123" -> "BELIEBE4EVER" [tweet_id=11966261062];
"@BieberBelle123" -> "sabrina9451" [tweet_id=11966197327];
}
```

Agora que temos a saída em linguagem DOT, o próximo passo deve ser convertê-la em uma imagem. O Graphviz em si fornece diversos algoritmos de layout para visualizar o grafo exportado; o `circo`, uma ferramenta para renderização de grafos em um layout de estilo circular, deve satisfazer à nossa necessidade, uma vez que os dados sugerem que o grafo deverá exibir a forma de um *ego graph* (*http://networkx.lanl.gov/examples/drawing/ego_graph.html*), apresentando uma topologia de estilo "eixo e raios", com um nó central altamente conectado a muitos nós com um grau de 1. Em uma plataforma *nix, o comando que veremos a seguir converte o arquivo

snl_search_results.dot, exportado do NetworkX, em um arquivo *snl_search_results.dot*.png, que você pode abrir em um visualizador de imagens (o resultado da operação é exibido na figura 1.2):

```
$ circo -Tpng -Osnl_search_results snl_search_results.dot
```

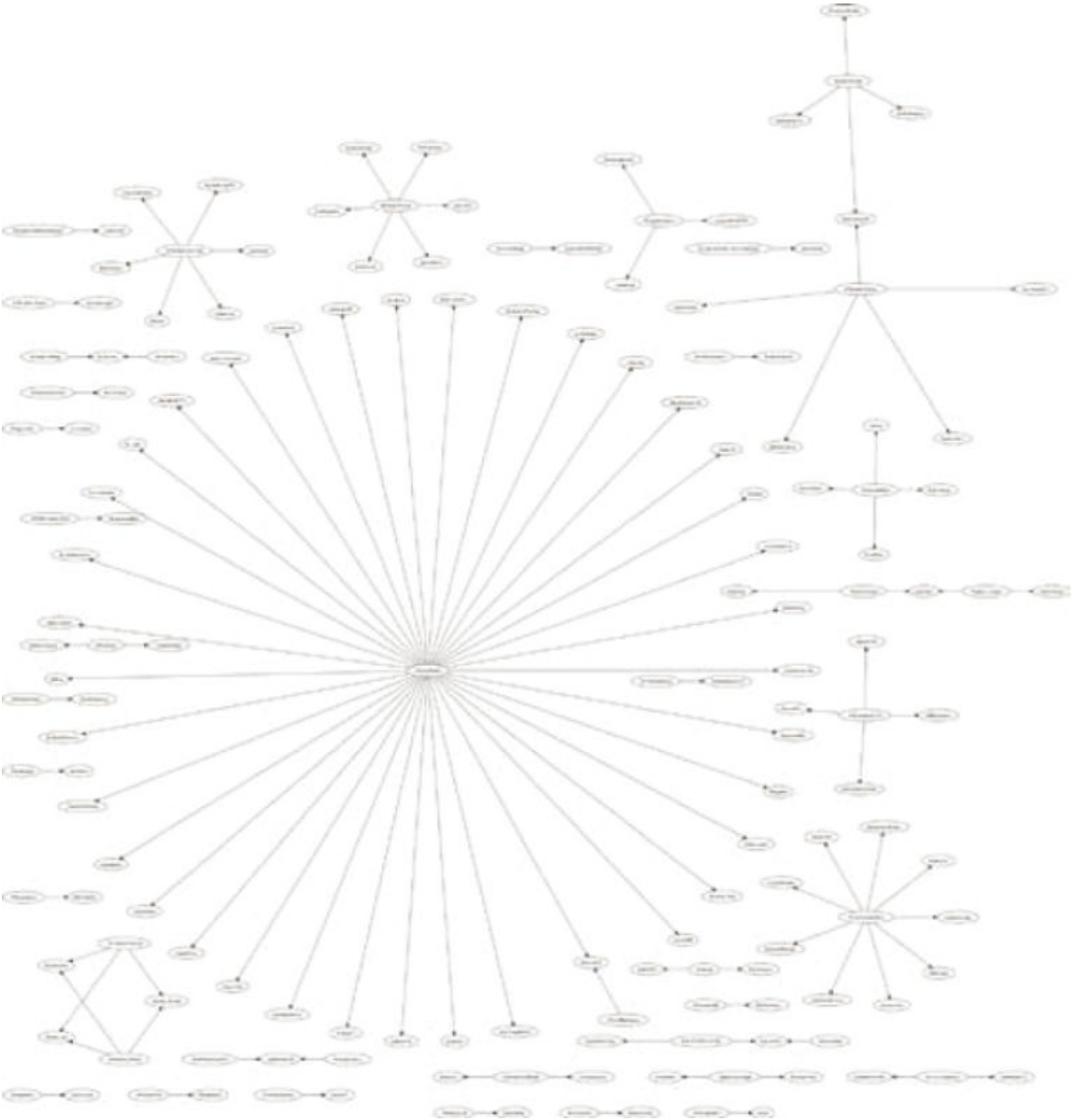


Figura 1.2 – Nossos resultados de busca renderizados em um layout circular com o Graphviz.

Usuários do Windows podem utilizar a aplicação GVedit para renderizar o arquivo, como mostrado na figura 1.3. Você pode ler mais sobre as várias opções do Graphviz em sua documentação on-line

(*http://www.graphviz.org/Documentation.php*). Uma inspeção visual do arquivo gráfico inteiro confirma que as características do grafo se alinham à nossa análise prévia, e podemos visualmente confirmar que o nó com grau mais elevado é `@justinbieber`, assunto de muita discussão (e, caso você não tenha assistido, apresentador convidado desse episódio do programa). Lembre-se de que, se tivéssemos coletado mais tweets, é bem provável que víssemos muito mais subgrafos interconectados do que os evidenciados pela amostra de 500 tweets que estamos analisando. Uma análise mais demorada do grafo fica como exercício facultativo para o leitor, uma vez que o objetivo principal deste capítulo é preparar seu ambiente de desenvolvimento e despertar seu apetite para tópicos mais interessantes.

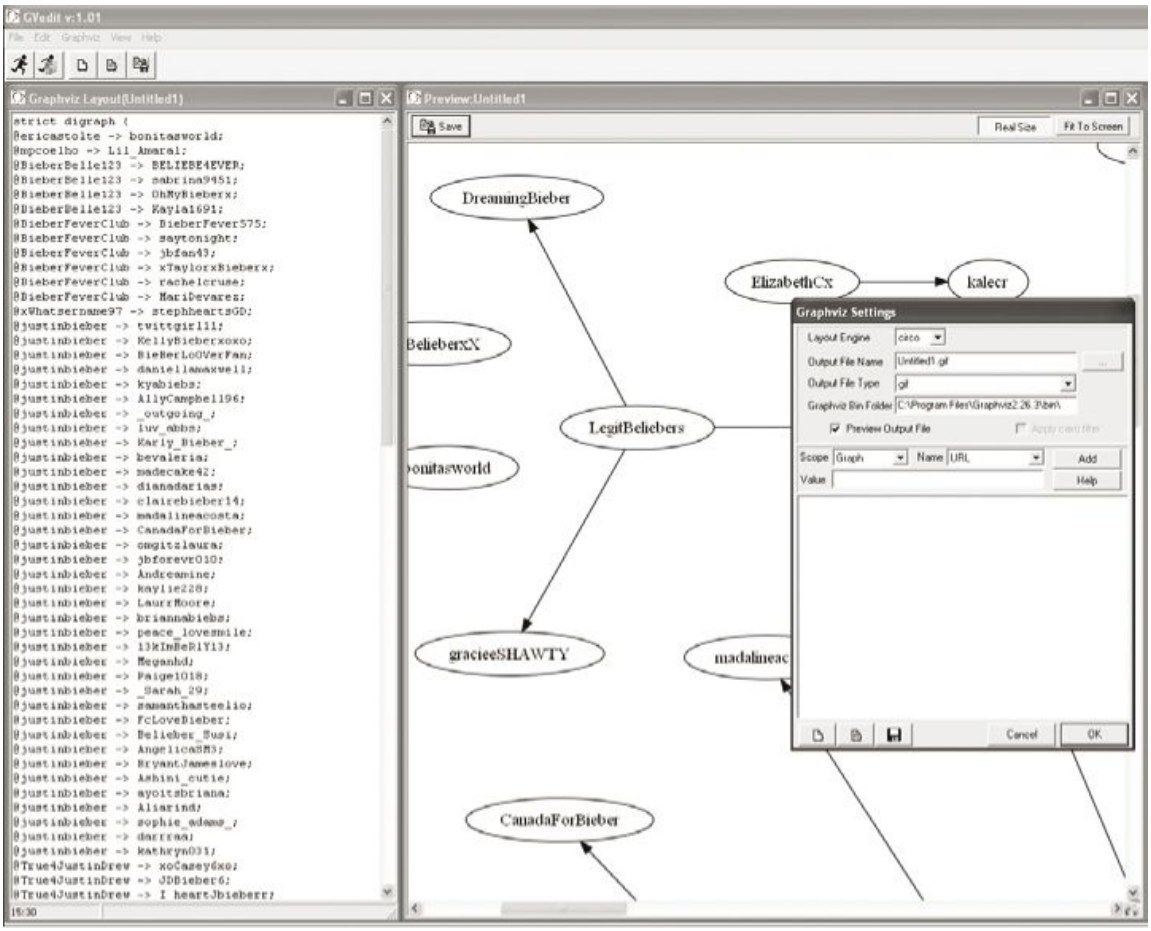


Figura 1.3 – Usuários do Windows podem utilizar o GVedit em vez de interagir com o Graphviz no prompt de comando.

O Graphviz voltará a ser mencionado neste livro e, se você se considera um cientista de dados (ou aspira a se tornar um), esta é uma ferramenta que deverá ser dominada. Dito isso, ainda veremos muitas outras

abordagens úteis para visualização de grafos. Nos capítulos seguintes, abordaremos saídas alternativas para dados da web social e técnicas de análise.

Síntese: visualização de retweets com o Protovis

Encerraremos este capítulo com um script de exemplo pronto para uso que sintetiza grande parte do conteúdo visto e adiciona uma visualização. Além de enviar algumas informações úteis para o console, ele aceita um termo de busca como parâmetro na linha de comando, busca, faz o parsing e abre seu navegador web para visualizar os dados como um grafo interativo com base em HTML5. Você pode encontrá-lo no repositório oficial de código para este livro em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/introduction__retweet_visualization.py*.

Encorajamos você a experimentá-lo. Voltaremos a falar do Protovis (*http://vis.stanford.edu/protovis/*), a biblioteca de visualização utilizada neste exemplo, em diversos capítulos na sequência do livro. A figura 1.4 ilustra a saída do Protovis para este script. O texto de exemplo no script de amostra é somente o início – muito mais pode ser feito!

Figura 1.4 – Um grafo interativo do Protovis, com layout direcionado, que visualiza relacionamentos de retweets para uma consulta "JustinBieber".

Comentários finais

Este capítulo, em uma rápida introdução, ilustrou como é fácil utilizar o interpretador interativo da Python para explorar e visualizar dados do Twitter. Antes que você avance para outros capítulos, é importante que se sinta à vontade com seu ambiente de desenvolvimento Python, e é altamente recomendado que passe algum tempo trabalhando com as APIs do Twitter e com o Graphviz. Se você está disposto a explorar algo diferente, pode verificar o canviz (*http://code.google.com/p/canviz/*), projeto que objetiva desenhar grafos do Graphviz no elemento `<canvas>` de um navegador web. Também pode ser interessante investigar o IPython (*http://ipython.scipy.org/moin/*), interpretador Python "melhorado" que oferece conclusão automática, monitoramento de histórico e muito mais. A maior parte do trabalho que você fará neste livro, daqui em diante, envolverá script executáveis, mas é importante que você seja o mais produtivo possível quando estiver experimentando novas ideias,

realizando debugs etc.

---

1 N.R.: NoSQL (entenda-se "Not only SQL") é um um termo genérico para uma classe definida de banco de dados não-relacionais (fonte: Wikipedia).

2 Ainda que os exemplos neste livro utilizem o renomado `easy_install`, a comunidade Python vem lentamente migrando para o `pip` (*http://pip.openplans.org/*), outra ferramenta de compilação que você deve conhecer e que, em geral, funciona adequadamente com qualquer pacote que possa ser instalado com o `easy_install.`

3 *http://support.twitter.com/entries/164083-what-is-a-timeline*

4 N.T.: os Trending Topics, ou tópicos populares, são uma lista em tempo real dos nomes mais postados no Twitter em todo o mundo. Valem para essa lista as hashtagas (#) e os nomes próprios (fonte: Wikipédia).

5 Se você não está familiarizado com REST, consulte a seção “Web services RESTful”, no capítulo 7, para uma breve explicação.

6 N.T.: token, em computação, é um segmento de texto ou símbolo que pode ser manipulado por um parser, que fornece um significado ao texto. Em outras palavras, é um conjunto de caracteres (de um alfabeto, por exemplo) com um significado coletivo (fonte: Wikipédia).

7 No presente contexto, compilar uma expressão regular significa transformá-la em bytecode para que possa ser executada por um engine correspondente escrito em C.

8 N.T.: cada linha de um banco de dados, formada por um conjunto de colunas, representa um registro (ou tupla). Os registros não têm de, necessariamente, conter dados em todas as colunas, os seus valores podem ser nulos (fonte: Wikipédia).

9 N.T.: payload, ou carga útil, em protocolos de comunicação, refere-se ao dado real sendo transmitido. Ele é seguido por um cabeçalho que identifica o transmissor e o receptor do dado transportado (fonte: Wikipédia).

10 Consulte o Ticket #117 do NetworkX (*https://networkx.lanl.gov/trac/ticket/117*), que demonstra que este é um problema antigo, que ainda não recebeu atenção devida para que fosse corrigido, mesmo depois de muitos anos de frustração. A questão subjacente está relacionada à necessidade de compilar código C durante o processo do *easy_install*. A habilidade de contornar esse problema com facilidade, por meio da geração de saída em linguagem DOT, pode ser parcialmente responsável por não haver até agora uma solução para o problema.

CAPÍTULO 2

Microformatos: quando marcação semântica e senso comum se encontram

Nos termos da evolução constante da web, os microformatos são um importante passo adiante, pois fornecem um mecanismo efetivo capaz de incorporar "dados mais inteligentes" às páginas, além de serem de fácil implementação para autores de conteúdo. Em poucas palavras, microformatos (*http://www.microformats.org*) são simplesmente padrões para inclusão de dados estruturados em páginas web, utilizando valores agregados. Este capítulo inicia com um panorama desse universo, para depois mergulhar diretamente em alguns exemplos que envolvem o uso específico dos microformatos XFN (XHTML Friends Network) (*http://gmpg.org/xfn/*), geo (*http://microformats.org/wiki/geo*), hRecipe (*http://microformats.org/wiki/hrecipe*), e hReview (*http://microformats.org/wiki/hReview*). Mais especificamente, neste capítulo mineraremos relacionamentos humanos a partir de blogrolls (ou listas de blogs), extrairemos coordenadas de páginas web, faremos o parsing de receitas do foodnetwork.com (*http://foodnetwork.com*) e analisaremos críticas de algumas delas. Os exemplos mostrados nas listagens de código não foram implementados com a intenção de serem "parsers completos", mas devem ser o suficiente para oferecer um ponto de partida.

Ainda que possa parecer um exagero chamar de "dados sociais" os dados complementados por microformatos, como os de tipo geo ou hRecipe, essa nomenclatura ainda é interessante e inevitavelmente será cada vez mais presente em mashups[1] de dados sociais. Quando da redação deste livro, praticamente metade de todos os desenvolvedores web diz já ter utilizado microformatos de alguma forma (*http://www.webdirections.org/sotw10/markup/*). Além disso, a comunidade do microformats.org acabou de celebrar seu quinto aniversário (*http://microformats.org/2010/07/08/microformats-org-at-5-hcards-rich-*

snippets), e o Google reportou que, em 94% dos casos, microformatos estão envolvidos em seus Rich Snippets (resultados de busca com informações adicionais, *http://microformats.org/2010/07/08/microformats-org-at-5-hcards-rich-snippets*). Se pudermos confiar no Google, os microformatos devem ter um crescimento significativo no futuro; de fato, segundo o ReadWriteWeb, o Google deseja que ao menos 50% das páginas web tenha alguma forma de marcação semântica, e encoraja diferentes empresas a influenciar suas concorrentes para que também empreguem tais iniciativas (*http://www.readwriteweb.com/archives/google_semantic_web_push_rich_snippets_usage_grow.php*). Independentemente de sua perspectiva, certamente veremos mais dos microformatos no futuro se ficarmos atentos ao espaço da web, por isso, mãos à obra.

XFN e amigos

Entusiastas da web semântica proclamam que tecnologias como a FOAF (*http://en.wikipedia.org/wiki/FOAF_(software)*), sigla de Friend of a Friend, ou amigo de um amigo – ontologia que descreve relacionamentos entre pessoas, suas atividades etc. –, podem um dia vir a ser o catalisador que impulsionará o surgimento de redes sociais descentralizadas, construídas como a antítese de plataformas rigidamente controladas, como o Facebook. O fato das chamadas tecnologias de web semântica, como a FOAF, ainda não terem atingido o ponto crucial que fará com que se tornem amplamente disseminadas não deve surpreendê-lo. Se você conhece um pouco a curta história da web, sabe que inovações são constantes e que a natureza altamente descentralizada na qual a web opera não é muito conducente a revoluções que ocorram de um dia para o outro (consulte o capítulo 10). Em vez disso, a mudança parece ocorrer continuamente, de modo fluido, e muito evolucionário. A forma como os microformatos evoluíram, preenchendo gradualmente a lacuna dos "dados inteligentes" na web, é um exemplo especialmente adequado da união de tecnologias existentes a padrões recentes que ainda não estão completos. Neste caso específico, trata-se da história do avanço de uma web consideravelmente ambígua, com base principalmente no padrão HTML 4.01 (*http://www.w3.org/TR/REC-html40/*) legível por seres humanos, em direção a uma web mais semântica, na qual informações são menos ambíguas e mais adequadas à interpretação por máquinas.

A beleza dos microformatos está no fato de permitirem não apenas a incorporação de dados relacionados às redes sociais, mas também a calendários, currículos, e favoritos compartilhados, que podem já estar contidos na marcação HTML atualmente existente, de forma a possibilitar uma compatibilidade retroativa completa. O ecossistema geral é bem diverso, apresentando alguns microformatos já estabelecidos, como o geo, ao lado de outros que ganham terreno e rapidamente alcançam popularidade entre mecanismos de busca, sites de mídias sociais e plataformas para blogs. Quando da redação deste livro, alguns acontecimentos notáveis estavam em curso, como o anúncio de que o Google passou a utilizar o hRecipe como parte de seus Rich Snippets (*http://googlewebmastercentral.blogspot.com/2010/04/better-recipes-on-web-introducing.html*). A tabela 2.1 fornece uma sinopse com alguns dos mais populares microformatos, e suas respectivas ações, que podem ser encontrados em pesquisas na web. Para mais exemplos, consulte *http://microformats.org/wiki/examples-in-the-wild*.

Tabela 2.1 – Algumas tecnologias populares para incorporação de dados estruturados a páginas web

Tecnologia	Propósito	Popularidade	Especificação da marcação	Tipo
XFN	Representa relacionamentos legíveis por seres humanos em hiperlinks	Amplamente utilizado, especialmente em plataformas de blogs	HTML semântico, XHTML	Microformato
geo	Incorpora coordenadas geográficas para pessoas e objetos	Amplamente utilizado, especialmente por sites como o MapQuest e a Wikipédia	HTML semântico, XHTML	Microformato
hCard	Identifica pessoas, empresas e outras informações de contato	Amplamente utilizado	HTML semântico, XHTML	Microformato
hCalendar	Incorpora dados no formato iCalendar	Gradualmente ganhando tração (http://microformats.org/2010/04/28/google-adds-support-for-hcalendar-and-hrecipe-rich-snippets)	HTML semântico, XHTML	Microformato
hResume	Incorpora informações de currículos	Amplamente utilizando por sites como o LinkedIn[a]	HTML semântico, XHTML	Microformato
hRecipe	Identifica receitas	Amplamente utilizado por sites de nicho como o foodnetwork.com	HTML semântico, XHTML	Microformato
Microdata	Incorpora pares nome/valor a páginas web criadas em HTML5	Tecnologia emergente, mas que vem ganhando tração (http://googlewebmastercentral.blogspot.com/2010/03/microdata-support-for-rich-snippets.html)	HTML5	Iniciativa do W3C

Tecnologia	Propósito	Popularidade	Especificação da marcação	Tipo
RDFa	Incorpora fatos inequívocos a páginas XHTML de acordo com vocabulários especializados criados por especialistas nessas áreas	Seu resultado pode ser positivo ou negativo, dependendo do vocabulário específico; vocabulários como FOAF estão gradualmente ganhando popularidade, enquanto outros permanecem obscuros	XHTML[b]	Iniciativa do W3C
Porotocolo Open Graph	Incorpora perfis de elementos da vida real a páginas XHTML	Gradualmente ganhando popularidade, com potencial tremendo dado o alcance da plataforma Facebook (http://www.facebook.com/press/info.php?statistics)	XHTML (com base em RDFa)	Iniciativa da plataforma Facebook

(a) O LinkedIn apresenta currículos públicos em formato hResume (*http://steve.ganz.name/blog/2007/01/linkedin-launches-hresume.html*) para seus mais de 75 milhões de usuários no mundo todo (*http://press.linkedin.com*).

(b) Quando da redação deste livro, a incorporação do RDFa à marcação semântica e ao HTML5 ainda estava em andamento. Consulte o rascunho de trabalho do W3C sobre HTML+RDFa 1.1 (*http://www.w3.org/TR/rdfa-in-html/*).

Há muitos outros microformatos, mas uma boa regra prática é observar o que os principais líderes do mercado – como o Google, o Yahoo! e o Facebook – estão fazendo. Quanto mais suporte um microformato tiver de um desses grandes nomes, maior a probabilidade de que ele obtenha êxito e se torne útil para a mineração de dados.

Marcação semântica?

Dado que o propósito dos microformatos é incorporar conhecimento semântico a páginas web, faz sentido que XHTML ou HTML semântico sejam utilizados. Mas qual é exatamente a diferença entre HTML semântico, ou marcação semântica, e XHTML? Uma forma de analisar esta questão é partir da perspectiva da separação entre conteúdo e apresentação. HTML semântico é marcação que enfatiza o significado da informação na página, com tags que se concentram no conteúdo, e não na apresentação. Infelizmente, o HTML, na forma como foi definido originalmente e de acordo com sua evolução inicial, fez muito pouco para promover a adoção de uma marcação semântica. O XHTML, por outro lado, representa XML *bem formado* (ou seja, em conformidade com regras rígidas de sintaxe), tendo sido desenvolvido no final da década de 1990 para solucionar o importante problema da separação entre conteúdo e apresentação, questão que havia surgido com o HTML. A intenção era a de que conteúdo original pudesse ser criado em XML sem nenhum foco na apresentação, mas que ferramentas fossem capazes de transformá-lo em algo que pudesse ser consumido com facilidade e renderizado por navegadores. A meta para o conteúdo transformado passou a ser conhecida como XHTML, formato praticamente idêntico ao HTML, exceto pelo fato de que é XML válido e, como tal, requer que todos os elementos sejam encerrados com as devidas tags, ou que sejam de autoencerramento, e também corretamente aninhados e definidos em letras minúsculas.

Em termos de design, parecia que o XHTML era exatamente aquilo de que a web necessitava. Havia muito a ser ganho e virtualmente nada a perder, de acordo com sua proposta: utilizar conteúdo XHTML *bem formado*, que poderia ser *validado* contra um esquema XML, e aproveitar todos os benefícios do formato, como atributos personalizados utilizando namespaces (dispositivo do qual dependem tecnologias da web semântica como RDFa). O problema é que o formato simplesmente não alcançou o êxito esperado. Seja por culpa do Internet Explorer, pela confusão entre desenvolvedores quanto à entrega dos tipos MIME corretos aos navegadores, pela qualidade das ferramentas disponíveis para desenvolvedores em XML, ou pelo fato de que não era razoável esperar que toda a web simplesmente fizesse uma pausa para realizar a conversão; isso é tópico para uma discussão que não nos diz respeito. A realidade é que o resultado não foi o esperado. Como consequência, vivemos em um mundo em que marcação semântica com base no padrão HTML 4.01, criado

há mais de uma década, continua presente, enquanto tecnologias com base em XHTML, como RDFa, permanecem à margem. A maioria dos desenvolvedores web aguarda ansiosa e torce para que o HTML5 crie a convergência esperada.

Exploração de conexões sociais com o XFN

Tendo estabelecido um pouco de contexto sobre como os microformatos encaixam-se no cenário geral da web, vamos agora nos voltar a algumas aplicações práticas do XFN, de longe o microformato mais popular que você encontrará. Como você já deve saber, o XFN é um modo de identificar relacionamentos entre pessoas, incluindo algumas palavras-chave no atributo `rel` de uma tag de âncora. O XFN é comumente utilizando em blogs, e especialmente em plug-ins de "blogrolls", como os oferecidos pelo WordPress[2]. Considere o conteúdo HTML a seguir (Exemplo 2.1), que poderia estar presente em um blogroll.

Exemplo 2.1 – Exemplo de marcação XFN

```
<div>
    <a href="http://example.org/matthew" rel="me">Matthew</a>
    <a href="http://example.com/users/jc" rel="friend met">J.C.
</a>
    <a href="http://example.com/users/abe" rel="friend met co-
worker">Abe</a>
    <a href="http://example.net/~baseeret" rel="spouse
met">Baseeret</a>
    <a href="http://example.net/~lindsaybelle" rel="child
met">Lindsay Belle</a>
    </div>
```

Ao ler o conteúdo das tags `rel`, deve ficar óbvio quais são os relacionamentos existentes entre as várias pessoas. Mais especificamente, podemos ver que um rapaz de nome Matthew tem alguns amigos (`friend`s), uma esposa (`spouse`) e uma filha (`child`). Ele também trabalha com um desses amigos (`co-worker`) e conheceu todos (`met`) na "vida real" (o que não seria o caso se estivéssemos tratando de um associado estritamente on-line). Fora o uso de um vocabulário bem definido (*http://gmpg.org/xfn/1*), isso é tudo que você tem de saber sobre o XFN. A boa notícia é que, apesar de ser aparentemente simples, ele é incrivelmente poderoso quando empregado em uma escala considerável, pois se trata de informação estruturada e deliberadamente autorada. A menos que você encontre algum conteúdo incrivelmente desatualizado – como no caso de

melhores amigos que se tornaram arqui-inimigos, mas se esqueceram de atualizar devidamente seus blogrolls – o XFN oferece um indicador muito preciso da forma como duas pessoas estão conectadas. Como a maioria das plataformas oferece suporte ao XFN, há muita informação que pode ser analisada. A má notícia é que o XFN não lhe diz nada além desses dados básicos, por isso você terá de utilizar outras fontes de informação e técnicas para avançar além de conclusões simples, como "Matthew e Baseeret são casados e têm uma filha chamada Lindsay Belle.". Porém, temos de iniciar nossa jornada em algum ponto, não é mesmo?

Vamos criar um simples script que colete dados do XFN de modo semelhante ao serviço oferecido pelo rubhub (*http://rubhub.com/*), um engine de busca social que rastreia e indexa um grande número de sites utilizando o XFN. Talvez seja interessante conferir também uma das muitas ferramentas on-line do XFN (*http://gmpg.org/xfn/creator*) se você quiser explorar a especificação completa antes de avançar à próxima seção.

Rastreamento em largura-primeiro de dados do XFN

Vamos abordar o aspecto social minerando alguns dados do XFN e criando um grafo social de largura-primeiro[3] a partir deles. Como o XFN pode ser incorporado a qualquer página web concebível, a má notícia é que teremos de realizar um pouco de raspagem de dados (*web scraping*)[4]. A boa notícia, entretanto, é que esta será provavelmente a raspagem mais trivial que você já realizou, e o pacote `BeautifulSoup` minimiza muito o esforço necessário. O código no exemplo 2.2 utiliza o Ajaxian (*http://ajaxian.com*), um popular blog sobre desenvolvimento web moderno, como base para o grafo. Antes de tentar executar o `BeautifulSoup` não se esqueça de instalá-lo com o `easy_install`.

Exemplo 2.2 – Raspagem de conteúdo XFN de uma página web (microformats__xfn_scrape.py)

```
# -*- coding: utf-8 -*-

import sys
import urllib2
import HTMLParser
from BeautifulSoup import BeautifulSoup

# Experimente http://ajaxian.com/
URL = sys.argv[1]
```

```
XFN_TAGS = set([
    'colleague',
    'sweetheart',
    'parent',
    'co-resident',
    'co-worker',
    'muse',
    'neighbor',
    'sibling',
    'kin',
    'child',
    'date',
    'spouse',
    'me',
    'acquaintance',
    'met',
    'crush',
    'contact',
    'friend',
])
try:
    page = urllib2.urlopen(URL)
except urllib2.URLError:
    print 'Failed to fetch ' + item
try:
    soup = BeautifulSoup(page)
except HTMLParser.HTMLParseError:
    print 'Failed to parse ' + item
anchorTags = soup.findAll('a')
for a in anchorTags:
    if a.has_key('rel'):
        if len(set(a['rel'].split()) & XFN_TAGS) > 0:
            tags = a['rel'].split()
            print a.contents[0], a['href'], tags
```

A partir da versão 3.1.x, o BeautifulSoup passou a utilizar o HTMLParser (*http://docs.python.org/library/htmlparser.html*) em vez do SGMLParser (*http://docs.python.org/library/sgmllib.html*) e, como resultado, uma queixa usual de seus usuários é a de que ele parece menos robusto do que a versão anterior. O autor do BeautifulSoup recomenda outras opções, se você estiver entre os descontentes (*http://www.crummy.com/software/BeautifulSoup/3.1-problems.html*), sendo a mais óbvia simplesmente continuar utilizando a versão 3.0.x, caso a 3.1.x não atenda a todas as suas necessidades. Como alternativa, você pode capturar o HTMLParseError, como mostra o exemplo 2.2, e descartar o conteúdo.

Executar o código mostrado contra um URL que inclui informações XFN retornará o nome, o tipo do relacionamento e um URL para cada um dos amigos da pessoa. Um exemplo de saída pode ser visto a seguir:

```
Dion Almaer http://www.almaer.com/blog/ [u'me']
Ben Galbraith http://weblogs.java.net/blog/javaben/ [u'co-
worker']
Rey Bango http://reybango.com/ [u'friend']
Michael Mahemoff http://softwareas.com/ [u'friend']
Chris Cornutt http://blog.phpdeveloper.org/ [u'friend']
Rob Sanheim http://www.robsanheim.com/ [u'friend']
Dietrich Kappe http://blogs.pathf.com/agileajax/ [u'friend']
Chris Heilmann http://wait-till-i.com/ [u'friend']
Brad Neuberg http://codinginparadise.org/about/ [u'friend']
```

Supondo que o URL de cada amigo inclui o XFN ou outras informações úteis, é bem simples seguir os links e criar mais informações em grafos sociais de modo sistemático. Essa abordagem pode ser vista no próximo exemplo de código: a construção de um grafo em largura-primeiro, algo semelhante ao que descrevemos no exemplo 2.3 em pseudocódigo.

Exemplo 2.3 – Pseudocódigo para uma busca em largura-primeiro

```
Crie um grafo vazio
Crie uma fila vazia para monitorar os nós que serão processados

Adicione um ponto de partida para o grafo como o nó raiz
Adicione o nó raiz a uma fila para processamento

Repita até que alguma profundidade máxima seja alcançada ou que a
fila esteja vazia:
 Remova um nó da fila
 Para cada um dos vizinhos do nó:
    Se o vizinho ainda não foi processado:
      Adicione-o à fila
      Adicione-o ao grafo
      Crie uma aresta no grafo conectando o nó ao seu vizinho
```

Perceba que essa abordagem tem a vantagem de naturalmente criar arestas em ambas as direções entre os nós, desde que elas existam, sem que seja necessária nenhuma atividade adicional. Isso é perfeito para situações sociais, pois é naturalmente adequado para identificação de amigos mútuos sem nenhuma lógica adicional. Refinando o exemplo 2.2 alcançamos o exemplo 2.4, em que utilizamos a abordagem de largura-primeiro para criar um grafo NetworkX, seguindo os hiperlinks caso eles

contenham dados XFN. Executar o exemplo de código, mesmo para uma profundidade pequena como dois, pode produzir um grafo de tamanho considerável, dependendo da popularidade do XFN na comunidade de nicho abordada, como podemos ver ao gerar uma imagem do grafo e inspecioná-la, ou ao aplicar métricas a ela (Figura 2.1).

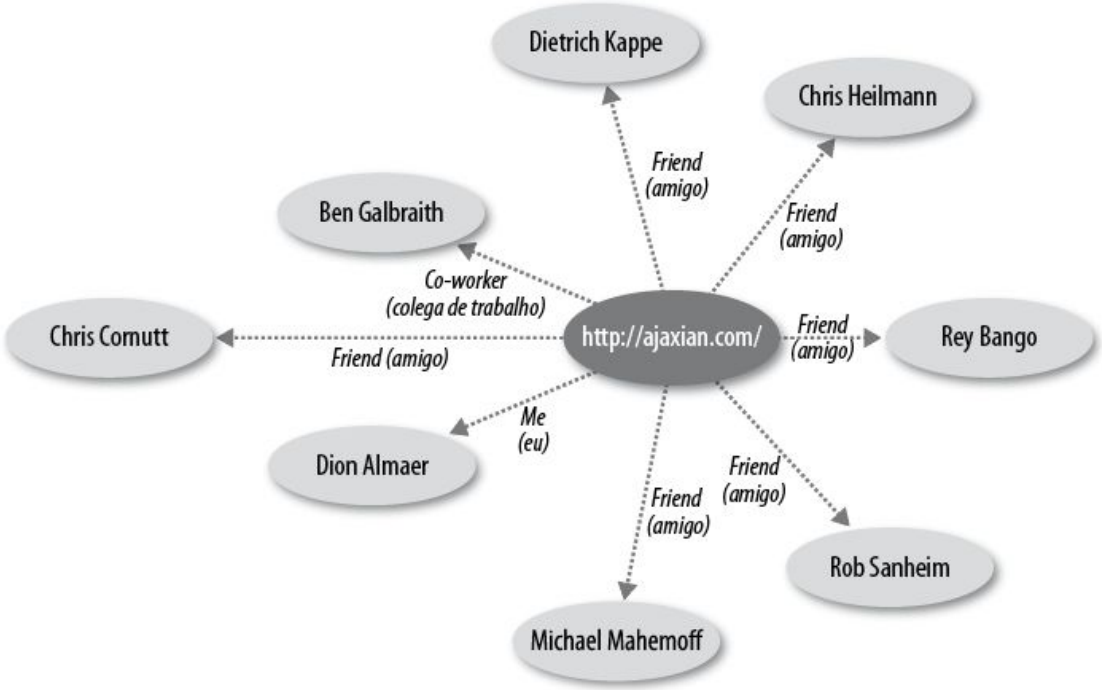


Figura 2.1 – Dígrafo de dados XFN do Ajaxian – note que o label "Me" da seta entre http://ajaxian.com e Dion Almaer indica que Dion é dono do blog.

Usuários do Windows podem consultar a seção "Visualização de grafos do Twitter", no capítulo 1, para que entendam a necessidade da "gambiarra" utilizada para o `nx.drawing.write_dot` no exemplo 2.4.

Exemplo 2.4. Uso de uma busca em largura-primeiro para rastrear links XFN (microformats__xfn_crawl.py)

```
# -*- coding: utf-8 -*-
import sys
import os
import urllib2
from BeautifulSoup import BeautifulSoup
import HTMLParser
import networkx as nx
ROOT_URL = sys.argv[1]
if len(sys.argv) > 2:
    MAX_DEPTH = int(sys.argv[2])
```

```
else:
    MAX_DEPTH = 1
XFN_TAGS = set([
    'colleague',
    'sweetheart',
    'parent',
    'co-resident',
    'co-worker',
    'muse',
    'neighbor',
    'sibling',
    'kin',
    'child',
    'date',
    'spouse',
    'me',
    'acquaintance',
    'met',
    'crush',
    'contact',
    'friend',
    ])
OUT = "graph.dot"
depth = 0
g = nx.DiGraph()
next_queue = [ROOT_URL]
while depth < MAX_DEPTH:
    depth += 1
    (queue, next_queue) = (next_queue, [])
    for item in queue:
        try:
            page = urllib2.urlopen(item)
        except urllib2.URLError:
            print 'Failed to fetch ' + item
            continue
        try:
            soup = BeautifulSoup(page)
        except HTMLParser.HTMLParseError:
            print 'Failed to parse ' + item
            continue
```

```
        anchorTags = soup.findAll('a')
        if not g.has_node(item):
            g.add_node(item)
        for a in anchorTags:
            if a.has_key('rel'):
                if len(set(a['rel'].split()) & XFN_TAGS) > 0:
                    friend_url = a['href']
                    g.add_edge(item, friend_url)
                    g[item][friend_url]['label'] =
a['rel'].encode('utf-8')
                    g.node[friend_url]['label'] =
a.contents[0].encode('utf-8')
                    next_queue.append(friend_url)
        # Uma análise mais completa do grafo poderia ser
realizada aqui
if not os.path.isdir('out'):
    os.mkdir('out')
try:
    nx.drawing.write_dot(g, os.path.join('out', OUT))
except ImportError, e:
    # Ajuda para usuários do Windows:
    # Este não se trata de um método de propósito geral, mas é
representativo da
    # mesma saída que write_dot fornece para este grafo, se
instalado e implementado
    dot = []
    for (n1, n2) in g.edges():
        dot.append('"%s" [label="%s"]' % (n2, g.node[n2]
['label']))
        dot.append('"%s" -> "%s" [label="%s"]' % (n1, n2, g[n1]
[n2]['label']))
    f = open(os.path.join('out', OUT), 'w')
    f.write('''strict digraph {
    %s
    }''' % (';\n'.join(dot), ))
    f.close()
# Usuários *nix podem produzir, a partir de um terminal,
# um arquivo de imagem com um bom layout da seguinte maneira:
# $ circo -Tpng -Ograph graph.dot
# Usuários do Windows podem utilizar as mesmas opções com o
circo.exe
```

```
# ou utilizar a aplicação de desktop GVedit
```

O uso do atributo `label` para os nós e setas no `DiGraph` não é arbitrário. Atributos incluídos em nós e setas são serializados na linguagem DOT e as ferramentas do Graphviz reconhecem `label` como um atributo especial (*http://www.graphviz.org/doc/info/attrs.html#d:label*).

Apesar de sua simplicidade, o grafo é muito interessante. Ele conecta oito pessoas, tendo um senhor de nome Dion Almaer como elo de ligação. Note, entretanto, que rastrear um ou mais níveis adicionais poderia introduzir outros nós no grafo, que talvez estivessem conectados a todos os outros. Analisando apenas este grafo, não fica claro se Dion tem um relacionamento mais próximo com Ben Galbraith do que com as outras pessoas, como indicado pela diferença entre relacionamentos dos tipos "colega de trabalho" e "amigo", mas poderíamos rastrear as informações de Ben no XFN, se ele as tivesse fornecido no destino identificado por este hiperlink, e buscar outras tags identificando colegas de trabalho. A partir dessas tags, poderíamos construir uma rede social determinando quem trabalha com quem. Consulte o capítulo 6 para mais informações sobre mineração de dados no quesito "colegas de trabalho".

Uma breve análise de técnicas de largura-primeiro

Geralmente não faremos uma pausa tão longa apenas para analisar uma abordagem empregada, mas como este exemplo é o primeiro algoritmo *de verdade* que escrevemos, e esta não será a última vez que o encontraremos no livro, vale a pena analisá-lo mais atentamente. Em geral, há dois critérios que devemos sempre considerar em um algoritmo: sua eficiência e sua efetividade. Ou, em outras palavras: seu desempenho e sua qualidade.

A análise-padrão do desempenho de qualquer algoritmo geralmente envolve um exame de sua complexidade de tempo e espaço no pior caso – em outras palavras, o tempo necessário para o programa executar, e a quantidade de memória necessária para a execução de um grande conjunto de dados. A abordagem de largura-primeiro que empregamos é essencialmente uma busca de largura-primeiro, ou de amplitude, exceto pelo fato de que não estamos de fato buscando nada específico, uma vez que não há critérios de saída para além da expansão do grafo até uma profundidade máxima, ou até que acabem os nós. Se estivéssemos buscando algo específico em vez de simplesmente rastrear links indefinidamente, esta seria considerada uma verdadeira busca de largura-

primeiro. Na prática, entretanto, quase todas as buscas impõem algum tipo de critério de saída, em razão das limitações dos recursos computacionais. Assim, uma variação mais usual de uma busca em largura-primeiro é conhecida como busca em *largura-primeiro limitada*, que impõe um limite à profundidade máxima da busca, como fizemos neste exemplo.

Para uma busca em largura-primeiro (ou rastreamento em largura-primeiro), tanto a complexidade do tempo quanto a do espaço podem ser limitadas no pior caso por b^d, em que b é o fator de ramificação do grafo e d é a profundidade. Se você esboçar um exemplo no papel e pensar um pouco sobre ele, verá que essa análise faz muito sentido. Se cada nó em um grafo tiver cinco vizinhos, e você utilizar uma profundidade de somente um, acabará com seis nós no total: o nó raiz e seus cinco vizinhos. Se todos esses cinco vizinhos também tiverem cinco vizinhos, e você expandir mais um nível, terminará com 31 nós no total: o nó raiz, seus cinco vizinhos, e cinco vizinhos para cada um dos vizinhos do nó raiz. Tal análise pode parecer pedante, mas na era emergente em que vivemos, a dos *big data*[5], ser capaz de realizar ao menos um nível mínimo de análise com base no que você sabe sobre o conjunto de dados é mais importante do que nunca. A tabela 2.2 fornece um resumo de como b^d cresce para alguns exemplos de b e d.

Tabela 2.2 – Exemplos do cálculo de fatores de ramificação para grafos de profundidades diferente

Fator de ramificação	Nós para profundidade = 1	Nós para profundidade = 2	Nós para profundidade = 3	Nós para profundidade = 4	Nós para profundidade = 5
2	3	7	15	31	63
3	4	13	40	121	364
4	5	21	85	341	1365
5	6	31	156	781	3906
6	7	43	259	1555	9331

Enquanto os comentários prévios trataram primariamente dos limites teóricos do algoritmo, uma consideração final que merece nossa atenção diz respeito ao seu desempenho prático para um conjunto de dados de tamanho fixo. Uma breve análise do código revela que ele é

primariamente *I/O bound*[6], no sentido em que a maior parte do tempo é gasta aguardando até que o método `urlopen` retorne conteúdo a ser processado. Um exemplo futuro, que veremos na seção "Threading de conversas", no capítulo 3, introduz e demonstra o uso de um *thread pool* para melhorar o desempenho nesses casos, em troca de um mínimo de complexidade.

Uma consideração final refere-se à qualidade geral dos resultados. Do ponto de vista da qualidade, uma inspeção visual básica do grafo revela que há de fato um grafo que conecta as pessoas. Missão cumprida? Sim, mas sempre há espaço para melhorias. Neste caso, pequenas variações nos URLs podem fazer com que nós múltiplos apareçam no grafo para a mesma pessoa. Por exemplo, se Matthew for mencionado em um hiperlink com o URL *http://example.com/~matthew*, mas também em um como *http://www.example.com/~matthew* em outro nó, esses dois nós permanecerão distintos no grafo, mesmo que apontem provavelmente para o mesmo recurso na web. Felizmente, o XFN define um valor especial `rel="me"` que pode ser utilizado para consolidação de identidade (*http://gmpg.org/xfn/and/#idconsolidation*). A API Social Graph do Google (*http://code.google.com/apis/socialgraph/*) adota essa abordagem para conectar os vários perfis de um usuário, e existem muitos exemplos de serviços que utilizam o `rel="me"` para permitir que os usuários conectem perfis a diversos sites externos. Outra questão (muito menos significativa) na resolução de URLs é a utilização ou omissão de uma barra invertida no final do endereço. A maioria dos sites bem projetados redirecionará automaticamente essas versões diferentes, por isso essa observação é praticamente desnecessária.

Felizmente, outras pessoas também encontraram tais problemas e decidiram fazer algo para solucioná-los. O SocialGraph Node Mapper (*http://code.google.com/p/google-sgnodemapper/*) é um interessante projeto de código aberto que padroniza URLs no que se refere às barras invertidas, à presença de "www" etc., mas que também reconhece que vários sites de redes sociais podem expor URLs diferentes, oferecendo links para a mesma pessoa. Por exemplo, *http://blog.example.com/matthew* e *http://status.example.com?user=matthew* podem representar a mesma pessoa para determinado site de rede social.

Coordenadas geográficas: um assunto que une praticamente tudo

Omitir de uma discussão sobre microformatos o geo ou o hRecipe, por não considerá-los particularmente úteis para a mineração de dados da web social, seria um grande equívoco. Ainda que certamente seja verdade que dados geográficos sozinhos, removidos de qualquer contexto, não são necessariamente sociais, relacionamentos importantes, e muito menos óbvios, geralmente emergem de conjuntos distintos de dados conectados por um contexto geográfico comum. Dados geográficos estão em todos os lugares e desempenham um papel importante em inúmeros mashups sociais, uma vez que um ponto específico no espaço pode ser utilizado como a ligação que une muitas pessoas. A divisão entre "vida real" e vida na web continua diminuindo e praticamente qualquer tipo de dado se torna social assim que vinculado a um indivíduo específico no mundo real. Por exemplo, muito pode ser deduzido sobre uma pessoa com base em seu local de residência, no tipo de alimentação que cozinha em sua casa, e até mesmo nos ingredientes específicos de suas receitas. Esta seção analisa alguns exemplos nos quais encontraremos, faremos o parsing e visualizaremos dados de tipo geo e hRecipe. Certamente você não demorará a imaginar uma utilidade para esse tipo de informação.

Artigos da Wikipédia + Google Maps = viagem a passeio?

Um dos microformatos mais simples e disseminados para incorporar informações de localização geográfica em páginas web é adequadamente chamado de geo. Sua especificação é inspirada em uma propriedade de mesmo nome do vCard (*http://www.ietf.org/rfc/rfc2426.txt*), que fornece uma maneira de especificar uma localização. Há duas formas possíveis de incorporar um microformato geo. O trecho de código HTML no exemplo 2.5 ilustra as duas técnicas disponíveis quando utilizadas na descrição de Franklin, a mais agradável cidadezinha do Tennessee.

Exemplo 2.5 – Exemplo de marcação do geo

```
<!-- Abordagem com várias classes -->
<span style="display: none" class="geo">
    <span class="latitude">36.166</span>
    <span class="longitude">-86.784</span>
```

```
</span>
<!-- Quando utilizado como uma única classe, o separador deve ser
uma semi-vírgula -->
<span style="display: none" class="geo">36.166; -86.784</span>
```

Como podemos ver, esse microformato simplesmente encapsula os valores de latitude e longitude em tags com os nomes de classe correspondentes, e empacota ambos dentro de uma tag com a classe geo. Muitos sites populares – incluindo a Wikipédia, o Yahoo! Local e o MapQuest Local, dentre outros – utilizam o geo e outros microformatos para expor dados estruturados em suas páginas.

Uma boa prática quando você estiver utilizando o geo é ocultar do usuário a informação codificada. Há duas formas de fazê-lo com CSS tradicionais: `style="display: none"` e `style="visibility: hidden"`. A primeira remove inteiramente o posicionamento do elemento na página, de modo que o layout se comporta como se ele não estivesse presente. A segunda oculta o conteúdo, mas reserva o espaço que o elemento ocupa na página.

O exemplo 2.6 ilustra um simples programa que faz o parsing de dados do microformato geo de uma página do MapQuest Local, para mostrar como você pode extrair coordenadas do conteúdo implementando o microformato geo.

Exemplo 2.6 – Extração de dados geográficos do MapQuest Local (microformats__mapquest_geo.py)

```
# -*- coding: utf-8 -*-
import sys
import urllib2
from BeautifulSoup import BeautifulSoup
import HTMLParser
# Passe um URL como http://local.mapquest.com/franklin-tn
url = sys.argv[1]
try:
    page = urllib2.urlopen(url)
except urllib2.URLError, e:
    print 'Failed to fetch ' + url
    raise e
    exit()
try:
    soup = BeautifulSoup(page)
except HTMLParser.HTMLParseError:
    print 'Failed to parse ' + url
    exit()
```

```
geoTag = soup.find(True, 'geo')
if geoTag and len(geoTag) > 1:
    lat = geoTag.find(True, 'latitude').string
    lon = geoTag.find(True, 'longitude').string
    print 'Location is at', lat, lon
elif geoTag and len(geoTag) == 1:
    (lat, lon) = geoTag.string.split(';')
    (lat, lon) = (lat.strip(), lon.strip())
    print 'Location is at', lat, lon
else:
    print 'No location found'
```

As implicações do uso de microformatos são sutis, mas profundas: ainda que uma pessoa lendo um artigo sobre uma cidade chamada Franklin, no Tennessee, possa intuitivamente saber que o ponto no mapa mostrado na página representa a localização dessa cidadezinha, um robô não seria capaz de formular a mesma conclusão sem uma lógica especializada que direcionasse diversas possibilidades de correspondência de padrões. Uma raspagem de página desse tipo é uma proposta complexa e, geralmente, justo quando você acha que atendeu a todas as possibilidades, descobre que se esqueceu de alguma. Incorporar a semântica correta na página, de modo a marcar efetivamente dados não-estruturados de uma forma que até mesmo nosso amigo Robby, O Robô, (*http://en.wikipedia.org/wiki/Robby_the_Robot*) possa compreender, remove ambiguidades e facilita o trabalho de rastreadores e desenvolvedores como você. Trata-se de um cenário em que todos saem ganhando, tanto o produtor, quanto o consumidor, e, com sorte, o efeito final será uma maior inovação, benéfica a todos.

Representação de dados geo via microform.at e Google Maps

Quando encontrar uma página web com dados geo interessantes incorporados, seu primeiro interesse deve ser visualizá-los. Por exemplo, considere o artigo na Wikipédia "List of National Parks of the United States" (Listas de parques nacionais dos Estados Unidos, *http://en.wikipedia.org/wiki/List_of_U.S._national_parks*), que exibe uma boa visualização tabular dos parques nacionais norte-americanos, marcando-os com formatação geo. Não seria melhor carregar rapidamente esses dados em uma ferramenta interativa que permitisse inspeção visual? Bem, o microform.at (*http://microform.at*) é um fantástico pequeno

serviço que extrai diversos tipos de microformatos para um dado URL, retransmitindo-os em muitos formatos úteis e oferecendo várias opções que permitem detectar e interagir com esses dados em páginas web (Figura 2.2).

Figura 2.2 – Resultados do http://microform.at para o artigo da Wikipédia intitulado "List of National Parks of the United States".

Se você tiver a opção, uma saída em KML (*http://code.google.com/apis/kml/documentation/*), sigla de Keyhole Markup Language, será talvez a forma mais fácil de visualizar dados geo. Você pode fazer o download do Google Earth e carregar o arquivo KML localmente, ou digitar um URL contendo dados KML diretamente na barra de busca do Google Maps, para exibir o resultado sem que seja necessário nenhum esforço adicional. Nos resultados exibidos para o microform.at, clicar no link "KML" dispara o download de um arquivo que pode ser utilizado no Google Earth. Você também pode copiá-lo para a área de transferência com um clique no botão direito e passá-lo ao Google Maps. A figura 2.3 exibe a visualização do Google Maps para o endereço *http://microform.at/?type=geo&url=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FList_of_U.S._national_parks* – resultados em KML do artigo mencionado da Wikipédia, que é simplesmente o URL-base, *http://microform.at*, acompanhado de parâmetros de consulta `type` e `url`.

A habilidade de trabalhar a partir de um artigo da Wikipédia, contendo marcação semântica como dados geo, e facilmente visualizá-los é uma

poderosa capacidade analítica, pois além de fornecer insights rapidamente, requer muito pouco esforço. Extensões de navegadores, como o add-on Firefox Operator (*https://addons.mozilla.org/en-US/firefox/addon/4106/*), procuram facilitar ainda mais seu trabalho. Há um limite para o que podemos abordar neste capítulo, mas uma forma interessante de investir seu tempo seria combinar os dados dos parques nacionais desta seção com informações de contato de sua rede profissional no LinkedIn, e descobrir como você poderia se divertir um pouco mais em sua próxima viagem de negócios (quem sabe uma possivelmente premeditada). Consulte a seção "Agrupamento geográfico de sua rede", no capítulo 6, para ver um exemplo sobre como coletar e analisar dados geo aplicando a técnica das k-médias para localizar clusters e computar centróides para eles.

Figura 2.3 – Resultados no Google Maps que exibem todos os parques nacionais nos Estados Unidos quando transmitimos resultados KML a partir do microform.at.

Fatiando e picando receitas (isso faz bem à saúde)

Desde o início do projeto de Rich Snippets do Google, há uma popularização cada vez maior dos microformatos, e muitos dos mais famosos sites de gastronomia fizeram grande progresso na divulgação de receitas e críticas utilizando o hRecipe e o hReview. Considere o potencial de sucesso de um fictício serviço on-line de encontros que rastreasse blogs e comunidades sociais, buscando reunir pessoas para jantares. Seria de se esperar que o acesso a informações geo e hRecipe suficientes, conectadas às pessoas específicas, teria impacto significativo na "taxa de sucesso" dos

primeiros encontros. As pessoas poderiam ser relacionadas de acordo com dois critérios: a proximidade de suas residências e o tipo de alimentação que preferem. Por exemplo, um jantar entre duas pessoas que preferem uma dieta vegetariana com ingredientes orgânicos provavelmente teria uma chance maior de sucesso do que um encontro entre um apaixonado por churrascos e um *vegan*[7]. Preferências gastronômicas e informações sobre o uso de tipos específicos de ingredientes orgânicos e de alergênicos poderiam ser dicas muito úteis para essa proposta de negócio. Ainda que não queiramos lançar um novo serviço on-line de encontros, fica aqui nossa sugestão, caso você decida colocá-la em prática.

O Food Network (*http://www.foodnetwork.com*) é um dentre os muitos sites on-line que estão realmente empregando microformatos para melhorar a web como um todo, expondo informações de receitas gastronômicas no microformato hRecipe, e utilizando o hReview para divulgar críticas[8]. Esta seção demonstra como mecanismos de busca (ou você) podem fazer o parsing dos dados estruturados das receitas e críticas contidas nas páginas do Food Network para indexação e análise. Mesmo que agora não façamos nenhuma análise do texto livre dessas receitas ou críticas, nem armazenemos permanentemente a informação extraída, capítulos futuros demonstrarão como fazê-lo, caso você esteja interessado. Mais especificamente, o capítulo 3 apresentará o CouchDB, uma ótima forma de armazenar e compartilhar dados (e análises) extraídos de conteúdo web que emprega microformatos, e o capítulo 7 apresentará algumas informações fundamentais sobre processamento de linguagem natural (NLP) que você pode utilizar para obter um melhor entendimento do conteúdo das críticas. Coincidentemente, chegamos até a mostrar uma receita nesse capítulo.

Uma adaptação do exemplo 2.6, que faz o parsing dos dados formatados em hRecipe, pode ser vista no exemplo 2.7.

Exemplo 2.7 – Parsing de dados hRecipe para uma receita de Pad Thai (microformats__foodnetwork_hrecipe.py)

```
# -*- coding: utf-8 -*-
import sys
import urllib2
import json
import HTMLParser
```

```
import BeautifulSoup
# Passe um URL como
# http://www.foodnetwork.com/recipes/alton-brown/pad-thai-
recipe/index.html
url = sys.argv[1]
# Faz o parsing de algumas das informações pertinentes para uma
receita
# Consulte http://microformats.org/wiki/hrecipe
def parse_hrecipe(url):
    try:
        page = urllib2.urlopen(url)
    except urllib2.URLError, e:
        print 'Failed to fetch ' + url
        raise e
    try:
        soup = BeautifulSoup.BeautifulSoup(page)
    except HTMLParser.HTMLParseError, e:
        print 'Failed to parse ' + url
        raise e
    hrecipe = soup.find(True, 'hrecipe')
    if hrecipe and len(hrecipe) > 1:
        fn = hrecipe.find(True, 'fn').string
        author = hrecipe.find(True, 'author').find(text=True)
        ingredients = [i.string
                            for i in hrecipe.findAll(True,
'ingredient')
                                if i.string is not None]
        instructions = []
        for i in hrecipe.find(True, 'instructions'):
            if type(i) == BeautifulSoup.Tag:
                s = ''.join(i.findAll(text=True)).strip()
            elif type(i) == BeautifulSoup.NavigableString:
                s = i.string.strip()
            else:
                continue
            if s != '':
                instructions += [s]
        return {
            'name': fn,
            'author': author,
            'ingredients': ingredients,
```

```
            'instructions': instructions,
            }
    else:
        return {}
recipe = parse_hrecipe(url)
print json.dumps(recipe, indent=4)
```

Para um URL de exemplo, como a famosa receita de Pad Thai, de Alton Brown, você deve receber os resultados mostrados no exemplo 2.8.

Exemplo 2.8 – Resultados para a receita de Pad Thai do exemplo 2.7

```
{
   "instructions": [
       "Place the tamarind paste in the boiling water and set
aside ...",
       "Combine the fish sauce, palm sugar, and rice wine vinegar
in ...",
       "Place the rice stick noodles in a mixing bowl and cover
with ...",
       "Press the tamarind paste through a fine mesh strainer and
add ...",
       "Place a wok over high heat. Once hot, add 1 tablespoon of
the ...",
       "If necessary, add some more peanut oil to the pan and
heat until ..."
   ],
   "ingredients": [
       "1-ounce tamarind paste",
       "3/4 cup boiling water",
       "2 tablespoons fish sauce",
       "2 tablespoons palm sugar",
       "1 tablespoon rice wine vinegar",
       "4 ounces rice stick noodles",
       "6 ounces Marinated Tofu, recipe follows",
       "1 to 2 tablespoons peanut oil",
       "1 cup chopped scallions, divided",
       "2 teaspoons minced garlic",
       "2 whole eggs, beaten",
       "2 teaspoons salted cabbage",
       "1 tablespoon dried shrimp",
       "3 ounces bean sprouts, divided",
       "1/2 cup roasted salted peanuts, chopped, divided",
       "Freshly ground dried red chile peppers, to taste",
```

```
        "1 lime, cut into wedges"
    ],
    "name": "Pad Thai",
    "author": "Recipe courtesy Alton Brown, 2005"
    }
```

Mesmo que esta não seja de fato uma forma de análise social, pode ser interessante analisar variações da mesma receita e descobrir se há correlações entre o uso, ou a falta, de certos ingredientes e entre avaliações e críticas para as receitas. Por exemplo, você pode acessar algumas receitas diferentes de Pad Thai e determinar quais ingredientes estão presentes em todas e quais são menos usuais.

Reunindo críticas de restaurantes

Esta seção conclui nossos estudos sobre microformatos – e sobre culinária tailandesa – apresentando brevemente o hReview. O Yelp é um serviço popular que implementa o hReview para expor avaliações que clientes postam sobre restaurantes. O exemplo 2.9 demonstra como extrair informações de tipo hReview na forma como elas são implementadas pelo Yelp. Um URL de exemplo que você pode experimentar está no código, e representa um restaurante tailandês muito recomendado, se você um dia tiver a oportunidade de visitá-lo.

Ainda que as especificações do hReview sejam estáveis, implementações diferentes parecem variar e incluir divergências arbitrárias. Mais especificamente, o exemplo 2.9 não faz o parsing do autor da crítica como um hCard, pois a implementação do Yelp não inclui essa informação dessa forma.

Exemplo 2.9 – Parsing de dados de tipo hReview para uma receita de Pad Thai (microformats__yelp_hreview.py)

```
# -*- coding: utf-8 -*-
import sys
import re
import urllib2
import json
import HTMLParser
from BeautifulSoup import BeautifulSoup

# Passe um URL contendo informações hReview tal como
# http://www.yelp.com/biz/bangkok-golden-fort-washington-2
url = sys.argv[1]

# Faz o parsing de algumas das informações pertinentes para uma
crítica no Yelp
```

```
# Infelizmente, a qualidade das implementações do hReview varia
muito,
# por isso seu resultado pode ser diferente. Este código *não* é
um parser
# amplo de qualquer especificação. Consulte
http://microformats.org/wiki/hreview
def parse_hreviews(url):
    try:
        page = urllib2.urlopen(url)
    except urllib2.URLError, e:
        print 'Failed to fetch ' + url
        raise e
    try:
        soup = BeautifulSoup(page)
    except HTMLParser.HTMLParseError, e:
        print 'Failed to parse ' + url
        raise e
    hreviews = soup.findAll(True, 'hreview')
    all_hreviews = []
    for hreview in hreviews:
        if hreview and len(hreview) > 1:
            # A partir de 1º de janeiro de 2010, o Yelp não
implementa o autor da crítica como
            # um hCard, de acordo com a especificação
            reviewer = hreview.find(True, 'reviewer').text
            dtreviewed = hreview.find(True, 'dtreviewed').text
            rating = hreview.find(True, 'rating').find(True,
'value-title')['title']
            description = hreview.find(True, 'description').text
            item = hreview.find(True, 'item').text
            all_hreviews.append({
                'reviewer': reviewer,
                'dtreviewed': dtreviewed,
                'rating': rating,
                'description': description,
                })
    return all_hreviews
reviews = parse_hreviews(url)
# Faça algo interessante, como representar as críticas de forma
cronológica
# ou minerar o texto nas descrições...
```

```
print json.dumps(reviews, indent=4)
```

Um exemplo de resultados truncados para o exemplo 2.9 pode ser visto no exemplo 2.10, incluindo a informação sobre o autor da crítica, cujo parsing foi feito a partir dos nós microformatados do hCard, como seu próprio objeto.

Exemplo 2.10 – Exemplo de resultados do hReview, correspondentes ao exemplo 2.9

```
[
   {
       "reviewer": "Nick L.",
       "description": "Probably the best Thai food in the metro
area...",
       "dtreviewed": "4/27/2009",
       "rating": "5"
   },
   ...truncado...
]
```

Infelizmente, nem o Yelp, nem o Food Network fornecem informações suficientemente específicas que permitam conectar autores de críticas de forma significativa, mas com sorte isso mudará em breve, abrindo possibilidades adicionais para a web social. Nesse meio tempo, você pode aproveitar ao máximo os dados disponíveis e verificar a avaliação média de um restaurante, de acordo com dados cronológicos, para descobrir se ela melhorou ou piorou. Outra sugestão seria minerar os campos de descrição no texto. Consulte os capítulos 7 e 8 para mais informações sobre como isso pode ser feito.

Por brevidade, não é mostrado aqui o trabalho bruto da preparação dos dados JSON em uma única lista e seu carregamento em uma planilha. Porém, uma planilha dos dados está disponível para download, caso você esteja se sentindo preguiçoso.

Não há limite para as inovações que podem ocorrer quando você combina geeks de informática e dados gastronômicos, como podemos notar pela popularidade do recém-publicado livro *Cooking for Geeks* (*http://www.cookingforgeeks.com*), também da O'Reilly. À medida que evoluírem as capacidades dos sites gastronômicos em fornecer APIs adicionais, também evoluirão as inovações que vemos nesse meio.

Resumo

Se você tiver de recordar somente um ponto deste capítulo, lembre-se de

que microformatos são uma forma de decorar sua marcação e de expor tipos específicos de informações estruturadas, como receitas, informações para contato, e relacionamentos humanos. Microformatos têm vasto potencial, pois nos permitem pegar conteúdo existente e tornar seus dados explícitos, de acordo com um padrão previsível. Espere encontrar um crescimento significativo no interesse por essa área nos meses futuros. Você também verá inovações incríveis com base nos microdados do HTML5 (*http://www.w3.org/TR/html5/microdata.html*), à medida que esse formato for adquirindo market share. Caso tenha muito tempo livre, confira a API Social Graph do Google (*http://code.google.com/apis/socialgraph/*), sem confundi-la com o protocolo Open Graph do Facebook (*http://developers.facebook.com/docs/opengraph*), ou com a API Graph abordada no capítulo 9, que inclui um índice do XFN (*http://gmpg.org/xfn/*), do FOAF (*http://www.foaf-project.org*) e de outras conexões declaradas publicamente.

---

1 N.T.: um mashup é um site ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo (fonte: Wikipédia).

2 Consulte *http://codex.wordpress.org/Links_Add_New_SubPanel* para observar um exemplo de um plug-in popular.

3 N.T.: na teoria dos grafos, a busca em largura (busca em largura-primeiro, ou busca em amplitude) é um algoritmo de procura de árvore usado para realizar uma busca ou travessia em uma árvore ou grafo. Intuitivamente, você começa pelo nó raiz e explora todos os nós vizinhos primeiro pela largura (fonte: Wikipédia).

4 N.T.: a raspagem de dados da web (também chamada de extração de dados da web, ou web scraping) é uma técnica para extração de informações de sites que se concentra na transformação de conteúdo web não-estruturado, tipicamente em formato HTML, em dados estruturados que podem ser armazenados e analisados em um banco de dados central local ou em uma planilha de dados (fonte: Wikipédia).

5 N.T.: big data são conjuntos de dados tão grandes que se torna difícil administrá-los utilizando as ferramentas existentes para gerenciamento de bancos de dados. As dificuldades incluem a captura, o armazenamento, a busca, o compartilhamento, a análise e a visualização desses dados (fonte: Wikipédia).

6 N.T.: I/O bound refere-se a uma condição em que o tempo necessário para completar uma computação é determinado principalmente pelo período de tempo gasto na espera da conclusão de operações de entrada/saída (I/O). Trata-se do oposto de uma tarefa CPU bound (fonte: Wikipédia).

7 N.T.: o veganismo é uma filosofia de vida motivada por convicções éticas com base nos direitos dos animais; procura evitar a exploração ou abuso desses seres, boicotando atividades e produtos que ofendam esses preceitos (fonte: Wikipédia).

8 Em meados de 2010, o Food Network implementou o hReview basicamente da mesma

forma que o Yelp, que será apresentado na próxima seção. Entretanto, desde o início de janeiro de 2011, a implementação do Food Network foi alterada para incluir somente o formato hreview-aggregate (*http://microformats.org/wiki/hreview-aggregate*).

CAPÍTULO 3

Caixas de e-mail: elas nunca saem de moda

Este capítulo apresenta algumas ferramentas e técnicas fundamentais para análise de seus e-mails (componentes essenciais da Internet que, apesar de todos os avanços nas redes sociais, ainda estarão presentes por muitos anos), permitindo-nos responder a questões como:

- Quem envia o maior número de e-mails?
- Por acaso há um momento específico do dia (ou um dia específico na semana) em que é mais provável se obter uma resposta para uma questão?
- Quais pessoas enviam o maior número de mensagens entre si?
- Quais os assuntos dos tópicos mais populares de discussão?

Ainda que sites de mídia social coletem petabytes de dados sociais praticamente em tempo real, existe o problema de que, diferentemente dos e-mails, dados de redes sociais são gerenciados de modo centralizado por um provedor de serviços que pode criar regras para seu acesso e sobre o que pode ser feito com os dados[1]. Dados de e-mails, por outro lado, são em grande parte descentralizados e estão espalhados por toda a web na forma de muitas discussões em listas de mensagens que tratam de uma enormidade de tópicos interessantes. Mesmo que provedores de serviços como o Google e o Yahoo! restrinjam o uso de dados desse tipo, quando obtidos utilizando seus serviços, há formas menos formidáveis de minerar tal conteúdo, que chegam mesmo a ter uma probabilidade maior de êxito: você mesmo pode facilmente coletar os dados assinando uma lista e aguardando a chegada das mensagens em sua caixa de entrada, ou pedindo ao proprietário da lista que forneça um arquivo de seus dados etc. Outro ponto interessante é que, diferentemente dos sites de mídia social, empresas geralmente têm controle total sobre suas caixas de entrada e podem realizar análises agregadas capazes de identificar certos tipos de tendências.

Como você pode imaginar, nem sempre é fácil (ou às vezes sequer

possível) encontrar conjuntos de dados sociais adequados para ilustração, mas, felizmente, este capítulo utiliza alguns dos dados mais realistas concebíveis: o conjunto de dados públicos da Enron[2] (*http://www.cs.cmu.edu/~enron/*).

mbox: noções básicas sobre caixas de e-mail Unix

Caso você nunca tenha encontrado um arquivo mbox, saiba que ele é simplesmente um grande arquivo de texto com mensagens de e-mail concatenadas, facilmente acessíveis por ferramentas baseadas em texto. O início de cada mensagem é sinalizado por uma linha especial *From_*, formatada de acordo com o padrão `"From user@example.com Fri Dec 25 00:06:42 2009"`, em que a marcação de data/hora tem formato asctime (*http://opengroup.org/onlinepubs/007908775/xsh/asctime.html*), uma representação padronizada de largura-fixa de um timestamp.

O formato mbox é muito conhecido, e a maioria dos clientes de e-mail oferece uma opção "export" ou "save as" para exportar dados nesse formato, independentemente da implementação subjacente.

Em um arquivo mbox, a linha divisória entre as mensagens é determinada por uma linha *From_* precedida (exceto por sua primeira ocorrência) por exatamente duas novas linhas. O exemplo 3.1 mostra um mbox fictício contendo duas mensagens.

Exemplo 3.1 – Exemplo de arquivo mbox

```
From santa@northpole.example.org Fri Dec 25 00:06:42 2009
Message-ID: <16159836.1075855377439@mail.northpole.example.org>
References: <88364590.8837464573838@mail.northpole.example.org>
In-Reply-To: <194756537.0293874783209@mail.northpole.example.org>
Date: Fri, 25 Dec 2009 00:06:42 -0000 (GMT)
From: St. Nick <santa@northpole.example.org>
To: rudolph@northpole.example.org
Subject: RE: FWD: Tonight
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Sounds good. See you at the usual location.

Thanks,
-S

-----Original Message-----
From: Rudolph
```

```
Sent: Friday, December 25, 2009 12:04 AM
To: Claus, Santa
Subject: FWD: Tonight

Santa -

Running a bit late. Will come grab you shortly. Standby.

Rudy

Begin forwarded message:
> Last batch of toys was just loaded onto sleigh.
>
> Please proceed per the norm.
>
> Regards,
> Buddy
>
> --
> Buddy the Elf
> Chief Elf
> Workshop Operations
> North Pole
> buddy.the.elf@northpole.example.org

From buddy.the.elf@northpole.example.org Fri Dec 25 00:03:34 2009
Message-ID: <88364590.8837464573838@mail.northpole.example.org>
Date: Fri, 25 Dec 2009 00:03:34 -0000 (GMT)
From: Buddy <buddy.the.elf@northpole.example.org>
To: workshop@northpole.example.org
Subject: Tonight
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

Last batch of toys was just loaded onto sleigh.

Please proceed per the norm.

Regards,
Buddy

--
Buddy the Elf
Chief Elf
Workshop Operations
North Pole
buddy.the.elf@northpole.example.org
```

No exemplo 3.1, vemos duas mensagens, ainda que haja evidência de ao

menos mais uma no mbox. Cronologicamente, a primeira mensagem tem como autor um rapaz, Buddy, e acabou de ser enviada para *workshop@northpole.example.org*, uma oficina no Polo Norte, comunicando que os brinquedos acabaram de ser carregados. A outra mensagem no mbox é a resposta de Santa (Papai Noel) para Rudolph. Oculta em nosso exemplo está uma mensagem intermediária, em que Rudolph encaminha a mensagem de Buddy para o Papai Noel e envia uma nota informando seu atraso. Ainda que seja possível descobrir essas informações simplesmente lendo o texto das mensagens, também temos dicas importantes nos cabeçalhos Message-ID, References, e In-Reply-To, que são bem intuitivos e fornecem a base para algoritmos de exibição das discussões em threads e outros recursos dessa natureza. Veremos em breve um algoritmo popular que utiliza esses campos para criar threads de mensagens, mas o que vale notar agora é que cada mensagem tem um ID único, contém uma referência à exata mensagem que está sendo respondida (caso seja uma resposta), e pode fazer referência a várias outras mensagens na cadeia de respostas, que são parte da thread de discussão maior em questão.

Além do uso de alguns módulos Python em nosso trabalho, não discutiremos particularidades das mensagens de e-mail, como conteúdo multipart, MIME (*http://en.wikipedia.org/wiki/MIME*), codificação em 7 bits de transferência de conteúdo etc. Não é necessário fazer muita pesquisa para encontrar ótimas referências que abordam em profundidade esses tópicos, caso você esteja interessado em um exame mais detalhado.

Esses cabeçalhos são de importância vital. Mesmo neste simples exemplo, fica claro como nosso trabalho pode se complicar ao realizar o parsing do corpo em si de uma mensagem: o cliente utilizado por Rudolph destacou o conteúdo encaminhado com caracteres >, enquanto o cliente que o Papai Noel utilizou para responder aparentemente não destacou nada, mas incluiu um cabeçalho de mensagem legível por seres humanos. Tentar o parsing do fluxo exato da comunicação a partir de dados da caixa de e-mail pode ser algo muito complicado, se não completamente impossível, em razão das ambiguidades envolvidas. A maioria dos clientes de e-mail oferece uma opção que exibe cabeçalhos estendidos além dos normalmente exibidos, caso você esteja interessado em uma técnica um pouco mais acessível, que não o faça mergulhar em detalhes brutos para analisar esse tipo de informação. O exemplo 3.2 ilustra o fluxo de mensagem do exemplo 3.1, e a figura 3.1 mostra cabeçalhos de exemplo, como exibidos pelo Apple Mail.

Exemplo 3.2 – Fluxo de mensagem do exemplo 3.1

```
Fri, 25 Dec 2009 00:03:34 -0000 (GMT) - Buddy sends a message to
the workshop
Friday, December 25, 2009 12:04 AM - Rudolph forwards Buddy's
message to Santa with an additional note
Fri, 25 Dec 2009 00:06:42 -0000 (GMT) - Santa replies to Rudolph
```

Felizmente, muito pode ser feito sem que você tenha de essencialmente reimplementar um cliente de e-mail. Além disso, se o seu desejo é analisar a caixa de e-mail, basta importá-la para um cliente, não é mesmo? Ainda que isso seja um exercício rotineiro, vale a pena fazer uma pausa e verificar se o seu cliente de e-mail permite importar/exportar dados no formato mbox, para que você possa utilizar as ferramentas deste capítulo em sua análise.

A Python oferece algumas ferramentas básicas para parsing de dados mbox, e o script no exemplo 3.3 introduz uma técnica básica para conversão de dados mbox em um array de objetos JSON.

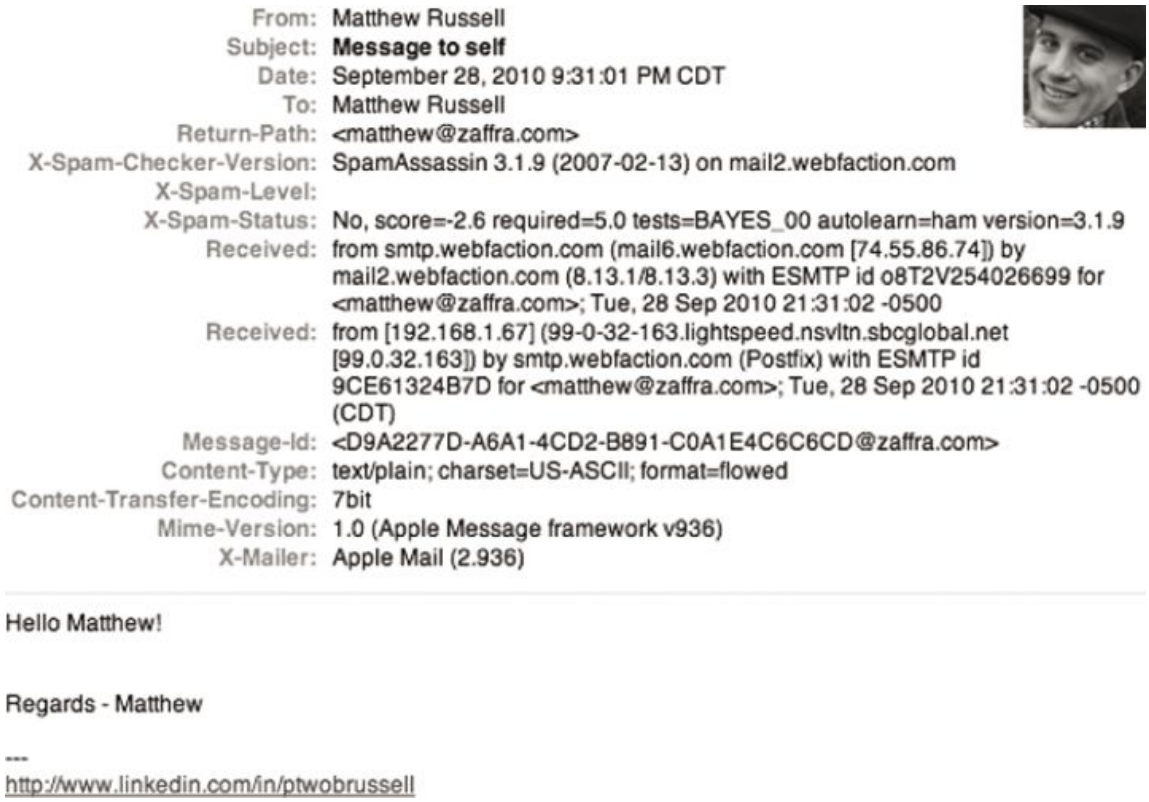


Figura 3.1 – Maioria dos clientes de e-mail permite que você visualize cabeçalhos estendidos por meio de um menu de opções.

Note que o exemplo 3.3 inclui a função `decode('utf-8', 'ignore')` em vários pontos. Quando estiver trabalhando com dados baseados em texto, como e-mails e páginas web, não será de todo

incomum encontrar o infame `UnicodeDecodeError`, provocado por erros de codificação de caracteres, e nem sempre será evidente qual sua causa ou como corrigir o problema. A solução mais simples é executar a função `decode` em qualquer valor de string e passá-la como um segundo argumento, especificando o que deve ser feito caso ocorra um `UnicodeDecodeError`. Seu valor padrão é `'strict'`, o que resulta no levantamento de uma exceção, mas você também pode utilizar `'ignore'` ou `'replace'`, dependendo de suas necessidades.

Exemplo 3.3 – Conversão de um mbox para uma estrutura JSON mais conveniente (mailboxes__jsonify_mbox.py)

```
# -*- coding: utf-8 -*-
import sys
import mailbox
import email
import quopri
from BeautifulSoup import BeautifulSoup
try:
    import jsonlib2 as json # muito mais rápido que o stdlib da
Python 2.6
except ImportError:
    import json
MBOX = sys.argv[1]
def cleanContent(msg):
    # Decodifica a mensagem do formato "quoted printable"
    msg = quopri.decodestring(msg)
    # Remove as tags HTML, se estiverem presentes
    soup = BeautifulSoup(msg)
    return ''.join(soup.findAll(text=True))
def jsonifyMessage(msg):
    json_msg = {'parts': []}
    for (k, v) in msg.items():
        json_msg[k] = v.decode('utf-8', 'ignore')
    # Os campos To, CC, e Bcc, se presentes, podem ter itens
múltiplos
    # Note que nem todos esses campos estão necessariamente
definidos
    for k in ['To', 'Cc', 'Bcc']:
         if not json_msg.get(k):
             continue
         json_msg[k] = json_msg[k].replace('\n',
'').replace('\t', '').replace('\r'
                  , '').replace(' ', '').decode('utf-8',
'ignore').split(',')
```

```
    try:
        for part in msg.walk():
            json_part = {}
            if part.get_content_maintype() == 'multipart':
                continue
            json_part['contentType'] = part.get_content_type()
            content = part.get_payload(decode=False).decode('utf-
8', 'ignore')
            json_part['content'] = cleanContent(content)
            json_msg['parts'].append(json_part)
    except Exception, e:
       sys.stderr.write('Skipping message - error encountered
(%s)' % (str(e), ))
    finally:
       return json_msg
# Nota: recomenda-se a abertura em modo binário
mbox = mailbox.UnixMailbox(open(MBOX, 'rb'),
email.message_from_file)
json_msgs = []
while 1:
    msg = mbox.next()
    if msg is None:
        break
    json_msgs.append(jsonifyMessage(msg))
print json.dumps(json_msgs, indent=4)
```

Desde a Python 2.6.x, módulos de terceiros escritos em C, como o `jsonlib2` (disponível via `easy_install`), são significativamente mais rápidos do que o módulo da biblioteca-padrão, mas a Python 2.7 vem com um update (*http://bugs.python.org/issue4136*) no mesmo nível desses módulos. A título de ilustração, a diferença entre o desempenho da biblioteca-padrão e o do `jsonlib2` para estruturas JSON de tamanho considerável (acima de 100 MB) pode estar na casa das dezenas de segundos.

Esse curto script faz um ótimo trabalho no parsing das informações mais pertinentes de um e-mail, e cria um objeto JSON portável. Poderíamos fazer mais, mas o que fizemos é suficiente para satisfazer algumas das questões mais usuais, incluindo um mecanismo primitivo para decodificar texto no formato *quoted-printable* e a remoção de tags HTML. O módulo `quopri` é utilizado para manipular o formato *quoted-printable*, codificação utilizada para transferir conteúdo de 8 bits em um canal de 7 bits[3]. Um exemplo de saída abreviada para o exemplo 3.3 pode ser visto no exemplo 3.4.

Exemplo 3.4 – Exemplo de saída em JSON como produzida pelo exemplo 3.3, a partir do mbox no

exemplo 3.1

```
[
   {
       "From": "St. Nick <santa@northpole.example.org>",
       "Content-Transfer-Encoding": "7bit",
       "To": [
           "rudolph@northpole.example.org"
       ],
       "parts": [
           {
               "content": "Sounds good. See you at the usual
location.\n\nThanks,...",
               "contentType": "text/plain"
           }
       ],
       "References": "
<88364590.8837464573838@mail.northpole.example.org>",
       "Mime-Version": "1.0",
       "In-Reply-To": "
<194756537.0293874783209@mail.northpole.example.org>",
       "Date": "Fri, 25 Dec 2001 00:06:42 -0000 (GMT)",
       "Message-ID": "
<16159836.1075855377439@mail.northpole.example.org>",
       "Content-Type": "text/plain; charset=us-ascii",
       "Subject": "RE: FWD: Tonight"
   },
   {
       "From": "Buddy <buddy.the.elf@northpole.example.org>",
       "Content-Transfer-Encoding": "7bit",
       "To": [
           "workshop@northpole.example.org"
       ],
       "parts": [
           {
               "content": "Last batch of toys was just loaded
onto sleigh. \n\n...",
               "contentType": "text/plain"
           }
       ],
       "Mime-Version": "1.0",
       "Date": "Fri, 25 Dec 2001 00:03:34 -0000 (GMT)",
       "Message-ID": "
<88364590.8837464573838@mail.northpole.example.org>",
```

```
        "Content-Type": "text/plain; charset=us-ascii",
        "Subject": "Tonight"
    }
    ]
```

Com as suas recém-adquiridas habilidades no parsing de dados de e-mails em um formato acessível, é natural que você não veja a hora de analisar seus dados. O restante deste capítulo utilizará os dados de e-mails da Enron, disponíveis ao público, como downloads mbox. O arquivo *enron.mbox.gz* é um mbox construído a partir de mensagens que constavam das pastas de "inbox" (caixas de entrada) do arquivo original da Enron, enquanto o arquivo *enron.mbox.json.gz* representa esses mesmos dados convertidos para JSON utilizando o script do exemplo 3.3. Ainda que não seja mostrado como exemplo, você pode fazer o download do script que realizou a conversão dos dados brutos para o formato mbox com o script *mailboxes__convert_enron_inbox_to_mbox.py* (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/mailboxes__convert_enron_inbox_to_mbox.py*). Note que, se você pesquisar on-line, perceberá que, além dos dados oficiais da Enron disponibilizados em *http://www.cs.cmu.edu/~enron/*, o que há de mais próximo disso são dados disponíveis apenas em formato não-padronizado, mais adequados para pesquisa, que incluem informações de agendamentos, notas etc. O script utilizado na conversão das partes desse conjunto de dados explicitamente marcadas como "Inbox data" para o formato mbox está disponível em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/mailboxes__convert_enron_inbox_to_mbox.py*. O restante deste capítulo presume que você esteja utilizando os dados mbox fornecidos como downloads.

mbox + CouchDB = Análise descontraída de e-mails

Utilizar a ferramenta adequada pode diminuir significativamente o esforço envolvido na análise de dados. Ainda que a abordagem mais óbvia para analisar dados estruturados seja criar um esquema a priori, importar dados para ele, realizar as modificações necessárias (uma vez que provavelmente vamos nos esquecer de algo), e então repetir esses passos mais algumas vezes, todo esse processo certamente não seria nada relaxante[4]. Dado o mapeamento natural que há entre a natureza centrada

em documentos de uma mensagem de e-mail, e uma estrutura de dados JSON, importaremos nossos dados mbox para o CouchDB (*http://couchdb.apache.org*), um banco de dados orientado a documentos que fornece recursos de mapeamento e redução (map/reduce) perfeitos para construir índices de dados e realizar análise de frequência agregada, respondendo a questões como: "Quantas mensagens foram enviadas por fulano?" ou "Quantas mensagens foram enviadas em tal data?".

Uma discussão completa sobre o CouchDB, incluindo sua posição no panorama geral do armazenamento de dados, e muitas outras de suas capacidades únicas, além daquelas mais pertinentes à análise básica, está além do escopo deste livro. Entretanto, não deve ser complicado acompanhar esta seção, mesmo que você nunca tenha ouvido falar no CouchDB.

Outro benefício agradável quanto ao uso do CouchDB para análise de documentos é o fato de que ele fornece uma interface inteiramente baseada em REST (*http://en.wikipedia.org/wiki/Representational_State_Transfer*) que pode ser integrada a qualquer arquitetura web-based, além de capacidades de replicação fáceis de utilizar, muito adequadas caso você queira permitir que outros clonem seus bancos de dados (e análises).

Web services RESTful

O REST (*http://en.wikipedia.org/wiki/Representational_State_Transfer*), sigla de Representational State Transfer, é um estilo de arquitetura de renome graças à dissertação de PhD de Roy Fielding (*http://www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf*), mas você não tem de ler esse texto para entendê-lo. Ainda que livros inteiros tenham sido escritos sobre REST, compreender alguns de seus princípios fundamentais deve ser o suficiente para nosso propósito: clientes e servidores devem realizar comunicação *stateless* (sem monitoramento de estado), e URIs devem descrever recursos que podem ser utilizados por verbos HTTP como GET, PUT, POST, HEAD e DELETE. Por exemplo, o URL hierárquico *http://example.com/blog* pode descrever um recurso de "blog". Um GET nesse recurso o buscaria, enquanto um PUT, no contexto /blog/foo, criaria um novo blog para o usuário "foo". Um DELETE, nesse mesmo contexto, excluiria o blog, e um POST /blog/foo poderia anexar um novo post ao blog de foo. O livro *RESTful Web Services* (*http://oreilly.com/catalog/9780596529260*), da O'Reilly, é um ótimo recurso caso você esteja interessado em uma análise mais detalhada a respeito do REST, ou queira ver muitos exemplos práticos de situações não tão óbvias.

O restante desta seção presume que você seja capaz de encontrar e instalar um binário do CouchDB para seu sistema, ou de compilar e instalá-lo a partir do código-fonte se preferir. Talvez valha a pena conferir a CouchOne (*http://www.couchone.com*), empresa que tem como CEO o criador do CouchDB e que emprega seus principais colaboradores, fornecendo downloads de binários para a maioria das plataformas, além de algumas opções gratuitas de hospedagem que podem ser úteis. A Cloudant (*https://cloudant.com*) é outra opção interessante de

hospedagem on-line.

De início, você pode pensar no CouchDB como um local de armazenamento de chaves e valores, em que as chaves são identificadores arbitrários e os valores são documentos baseados em JSON criados por você. A figura 3.2 ilustra um conjunto de documentos além de um documento individual, utilizando a Futon, interface administrativa baseada na web do CouchDB. Você pode acessar a Futon em sua instalação local, em *http://localhost:5984/_utils/*.

Figura 3.2 – Visualização agregada do banco de dados da Enron e de um de seus documentos individuais.

Assim que o CouchDB tiver sido instalado, sua próxima tarefa administrativa deverá ser instalar o módulo `couchdb` do cliente Python, executando `easy_install couchdb`. Você pode ler mais sobre o `couchdb` via `pydoc`, ou on-line em *http://packages.python.org/CouchDB/*. Tendo o

CouchDb e um cliente Python instalados, é hora de relaxar e preparar algum código que consuma nossos dados mbox transformados em JSON – fique à vontade para fazer testes com a Futon, caso este seja seu primeiro encontro com o CouchDB. Se agir de improviso não lhe deixa relaxar, faça uma pausa e verifique os primeiros capítulos do livro *CouchDB: The Definitive Guide* (O'Reilly), disponível em *http://books.couchdb.org/relax*.

Carregamento em massa de documentos no CouchDB

Executar o script do exemplo 3.3 no arquivo não comprimido `enron.mbox` e redirecionar sua saída para um arquivo, resulta em uma estrutura JSON de tamanho considerável (aproximadamente 200 MB). O script do exemplo 3.5 demonstra como você pode carregar os dados nesse estado no CouchDB utilizando o `couchdb-python`, que você já deve ter instalado via `easy_install couchdb` (você tem de admitir, o código parece bem relaxante quando comparado às alternativas, que exigiriam o carregamento de um esquema relacional). Note que pode demorar alguns minutos até que o script conclua, caso você esteja utilizando um hardware típico, como um computador laptop[5]. De início, uma demora de alguns minutos pode parecer excessiva, mas quando você considera que a estrutura JSON contém mais de 40.000 objetos que serão gravados como documentos CouchDB, isso representa mais de 300 transações de documentos por segundo, o que de fato é bem razoável para o desempenho de um simples laptop. Utilizar um utilitário de monitoramento de desempenho, como `top` em um sistema *nix, ou o Task Manager no Windows, deve revelar que um dos núcleos (*http://en.wikipedia.org/wiki/Multi-core_processor*) de sua CPU está sendo completamente utilizado, o que indicaria uma taxa de transferência máxima. Durante o desenvolvimento, vale a pena trabalhar com um conjunto menor de documentos do que com o conjunto completo de dados. Você pode facilmente selecionar cerca de uma dúzia das mensagens da Enron convertidas em JSON, e utilizá-las para acompanhar os exemplos deste capítulo.

Ainda que a Erlang, linguagem de programação por trás do CouchDB, seja reconhecida como uma linguagem que oferece suporte a alta concorrência, isso não significa que o CouchDB, ou qualquer outra plataforma de aplicação criada utilizando essa linguagem, possa utilizar todos os núcleos de todas as CPUs disponíveis. Por exemplo, no CouchDB, operações de escrita (write) em um banco de dados são operações somente de acréscimo (append) que são serializadas para a Árvore B no disco, e mesmo grandes volumes de gravações em lotes não podem ser distribuídos em mais de um núcleo.

Nesse sentido, há um mapeamento de um-para-um entre bancos de dados e núcleos para operações de escrita. Entretanto, você poderia realizar gravações em lotes para bancos de dados múltiplos ao mesmo tempo e maximizar sua taxa de transferência em mais de um núcleo.

Exemplo 3.5 – Pequeno script que demonstra o carregamento de dados JSON no CouchDB (mailboxes__load_json_mbox.py)

```
# -*- coding: utf-8 -*-
import sys
import os
import couchdb
try:
    import jsonlib2 as json
except ImportError:
    import json
JSON_MBOX = sys.argv[1] # por exemplo: enron.mbox.json
DB = os.path.basename(JSON_MBOX).split('.')[0]
server = couchdb.Server('http://localhost:5984')
db = server.create(DB)
docs = json.loads(open(JSON_MBOX).read())
db.update(docs, all_or_nothing=True)
```

Tendo carregado os dados, talvez pareça tentador simplesmente relaxar um pouco, mas, ao verificar parte deles com a Futon, podem surgir algumas questões sobre a natureza de quem está se comunicando, com qual frequência e quando isso ocorre. Vamos colocar em prática a funcionalidade de mapeamento/redução do CouchDB para que possamos responder a algumas dessas questões.

Se você não está em dia quanto às funcionalidades de mapeamento/redução (*http://en.wikipedia.org/wiki/MapReduce*), não se preocupe: ensiná-las é parte do propósito desta seção.

Em resumo, funções de mapeamento tomam um conjunto de documentos e *mapeiam* um novo par de chave/valor para cada documento, enquanto funções de redução tomam um conjunto de documentos, *reduzindo-o* de alguma forma. Por exemplo, a computação da soma aritmética dos quadrados, $f(x) = x1^2 + x2^2 + ... + xn^2$, poderia ser expressa como uma função de mapeamento que calcula o quadrado de cada valor, produzindo uma correspondência de um-para-um para cada valor de entrada, enquanto o redutor simplesmente soma a saída dos mapeadores, e a reduz a um único valor. Esse padrão de programação é muito adequado a problemas paralelizáveis triviais, mas de fato não apresenta sempre um bom desempenho em termos de performance. Deve ter ficado

claro que utilizar o CouchDB para solucionar um problema exige necessariamente que se verifique primeiro o cabimento desse paradigma com relação à tarefa apresentada.

Ordenação inteligente

Talvez ainda não pareça óbvio, mas se você analisar atentamente a figura 3.2, verá que os documentos inseridos no CouchDB estão ordenados por uma chave. Por padrão, essa chave é o valor especial `_id`, atribuído automaticamente pelo próprio CouchDB. Assim, neste momento, a ordem de classificação é praticamente sem significado. Para visualizar os dados na Futon de uma forma mais inteligente, além de realizar consultas por intervalo, será interessante realizar uma operação de mapeamento, criando uma visualização que classifique os elementos de acordo com outra chave. A ordenação por datas parece uma boa sugestão e permite também certos tipos de análises cronológicas, por isso vamos iniciar por ela e ver o resultado. Primeiro, entretanto, teremos de realizar uma pequena alteração de configuração para que possamos escrever nossas funções de mapeamento e redução, que realizarão essa tarefa em Python.

O CouchDB é especialmente intrigante, pois foi escrito em Erlang, linguagem projetada para oferecer suporte a alta concorrência[6] e tolerância a falhas. Na prática, a linguagem utilizada para consultar e transformar seus dados via funções de mapeamento/redução é JavaScript. Note que poderíamos certamente optar por escrever funções de mapeamento/redução nessa linguagem e obter alguns dos benefícios das funções JavaScript integradas oferecidas pelo CouchDB – como `_sum`, `_count`, e `_stats`. Porém, o benefício obtido pela verificação/realce da sintaxe de seu ambiente de desenvolvimento (IDE) pode se provar mais útil e agradável aos olhos do que a visualização das funções JavaScript encapsuladas como valores em strings entre aspas triplas, dentro do código Python. Além disso, presumindo que você tenha o `couchdb` instalado, basta um pequeno ajuste de configuração para especificar um servidor de visualização (*view server*) Python, permitindo que você escreva seu código em Python. Mais precisamente, no linguajar do CouchDB, diz-se que as funções de mapeamento/redução são funções de *visualização* (*view*) que residem em um documento especial, o *documento de design* (*http://guide.couchdb.org/draft/design.html*).

Simplesmente insira a linha a seguir (ou um equivalente compatível no Windows) na seção apropriada do arquivo de configuração *local.ini* do CouchDB, em que o executável `couchpy` é um servidor de visualização automaticamente instalado com o módulo `couchdb`. Não se esqueça de reiniciar o CouchDB depois de efetuar a alteração:

```
[query_servers] python = /path/to/couchpy
```

Se você está tendo dificuldades em encontrar o executável `couchpy` em um ambiente *nix, experimente utilizar `wich couchpy` em uma seção do terminal para encontrar seu path absoluto. Para usuários do Windows, utilizar o `easy_install` que acompanha o ActivePython deve colocar um arquivo `couchpy.exe` em *C:\PythonXY\Scripts*. Observar a saída quando você executa `easy_install couchdb` também revela essa informação.

A menos que queira escrever toneladas de código para realizar o parsing de strings com datas variáveis em um formato padronizado, você também deverá executar `easy_install dateutil`, e instalar um pacote de enorme utilidade que padroniza a maioria dos formatos de datas. Isso é importante, pois você nem sempre saberá com o que está lidando ao trabalhar com as marcações de data de um acervo desorganizado de e-mails. O exemplo 3.6 demonstra um script que mapeia documentos de acordo com suas marcações de data/hora, a ser executado nos dados de exemplo que importamos depois de configurarmos o CouchDB para a Python. Um exemplo de saída é simplesmente uma lista de documentos que corresponde à consulta, e foi omitido por brevidade.

Exemplo 3.6 – Simples mapeador que utiliza Python para mapear documentos de acordo com suas marcações de data/hora (mailboxes__map_json_mbox_by_date_time.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
try:
    import jsonlib2 as json
except ImportError:
    import json
DB = sys.argv[1]
START_DATE = sys.argv[2] #YYYY-MM-DD
END_DATE = sys.argv[3] #YYYY-MM-DD
server = couchdb.Server('http://localhost:5984')
db = server[DB]
def dateTimeToDocMapper(doc):
```

```
    # Note que você tem de incluir as importações utilizadas por
seu mapeador
    # dentro da definição da função
    from dateutil.parser import parse
    from datetime import datetime as dt
    if doc.get('Date'):
        # [ano, mês, dia, hora, minutos, segundos]
        _date = list(dt.timetuple(parse(doc['Date']))[:-3])
        yield (_date, doc)
# Especifica um índice para atender à consulta. Note que o índice
não será
# criado até a primeira execução da consulta
view = ViewDefinition('index', 'by_date_time',
dateTimeToDocMapper, language='python')
view.sync(db)
# Agora faz a consulta, verificando dados ordenados por data
start = [int(i) for i in START_DATE.split("-")]
end = [int(i) for i in END_DATE.split("-")]
print 'Finding docs dated from %s-%s-%s to %s-%s-%s' %
tuple(start + end)
docs = []
for row in db.view('index/by_date_time', startkey=start,
endkey=end):
    docs.append(db.get(row.id))
print json.dumps(docs, indent=4)
```

O ponto mais importante a compreender sobre esse código é o papel desempenhado pela função `dateTimeToDocMapper`, um *gerador*[7] personalizado que aceita um documento como parâmetro e emite o mesmo documento, agora marcado com um valor de data adequado, que poderá ser manipulado ou ordenado. Note que as funções de mapeamento no CouchDB não apresentam efeitos colaterais; independentemente do que ocorre na função de mapeamento, ou do que ela emite, o documento passado como parâmetro permanece inalterado. No linguajar do CouchDB, o `dateTimeToDocMapper` é uma *visualização* (*view*) nomeada de acordo com a data ("by_date_time"), que é parte de um *documento de design*, "index". Analisando a Futon, você verá que pode alterar o valor da combo box no canto superior direito, de "All documents" para "Design documents", e verificar você mesmo o resultado (Figura 3.3). Também pode ser interessante consultar o pydoc para

`ViewDefinition`, se você estiver interessado em mais informações.

Na primeira vez em que você for executar esse código, pode demorar em torno de cinco minutos para que a função de mapeamento seja realizada, o que deve ocupar um de seus núcleos durante esse intervalo. Cerca de 80% desse tempo é dedicado à construção do índice, enquanto o restante é utilizado para a realização da consulta. Entretanto, note que o índice tem de ser criado apenas uma vez, assim, consultas subsequentes serão relativamente mais eficientes, demorando cerca de 20 segundos para separar e retornar aproximadamente 2.200 documentos referentes ao intervalo de data especificado, o que significa cerca de 110 documentos por segundo.

A principal consequência dessa função de mapeamento é a de que todos os documentos serão indexados por suas datas, como pode ser verificado na Futon selecionando o valor de índice "by_date_time" na combo box "View", no canto superior direito da tela (Figura 3.4). Ainda que ordenar documentos por um determinado critério não seja a atividade mais interessante do mundo, este é um passo necessário em muitas funções analíticas agregadas, como na contagem do número de documentos que atendem a determinado critério dentro de um acervo inteiro. A próxima seção demonstra alguns elementos essenciais de tal análise.

Análise de frequência inspirada em mapeamento/redução

A tabulação de frequências é geralmente uma das primeiras tarefas exploratórias que você deve efetuar ao encontrar um novo conjunto de dados, pois oferece muitas informações e demanda pouco esforço. Esta seção investiga algumas das formas pelas quais você pode utilizar o CouchDB para construir índices baseados em frequência.

Frequência por intervalo de data/hora

Ainda que nosso índice `by_date_time` tenha feito um ótimo trabalho ao criar um índice que ordena documentos por data/hora, ele não é útil para contagem das frequências de documentos transmitidos dentro de limites usuais, como ano, mês, semana etc. Mesmo que certamente fosse possível escrever um código no cliente que calculasse frequências a partir de uma

chamada a `db.view('index/by_date_time', startkey=start, endkey=end)`, seria mais eficiente (e relaxante) deixar o CouchDB fazê-lo para nós. Para tanto, basta uma pequena alteração em nossa função de mapeamento e uma função de redução também muito simples. A função de mapeamento emitirá um valor de 1 para cada marcação de data/hora, e o redutor contará o número de chaves equivalentes. O exemplo 3.7 ilustra essa abordagem. Note que você deve executar `easy_install prettytable`, para instalar um pacote que produz sua saída em formato tabular, antes de executar este exemplo.

Figura 3.3 – Inspeção dos documentos de design na Futon.

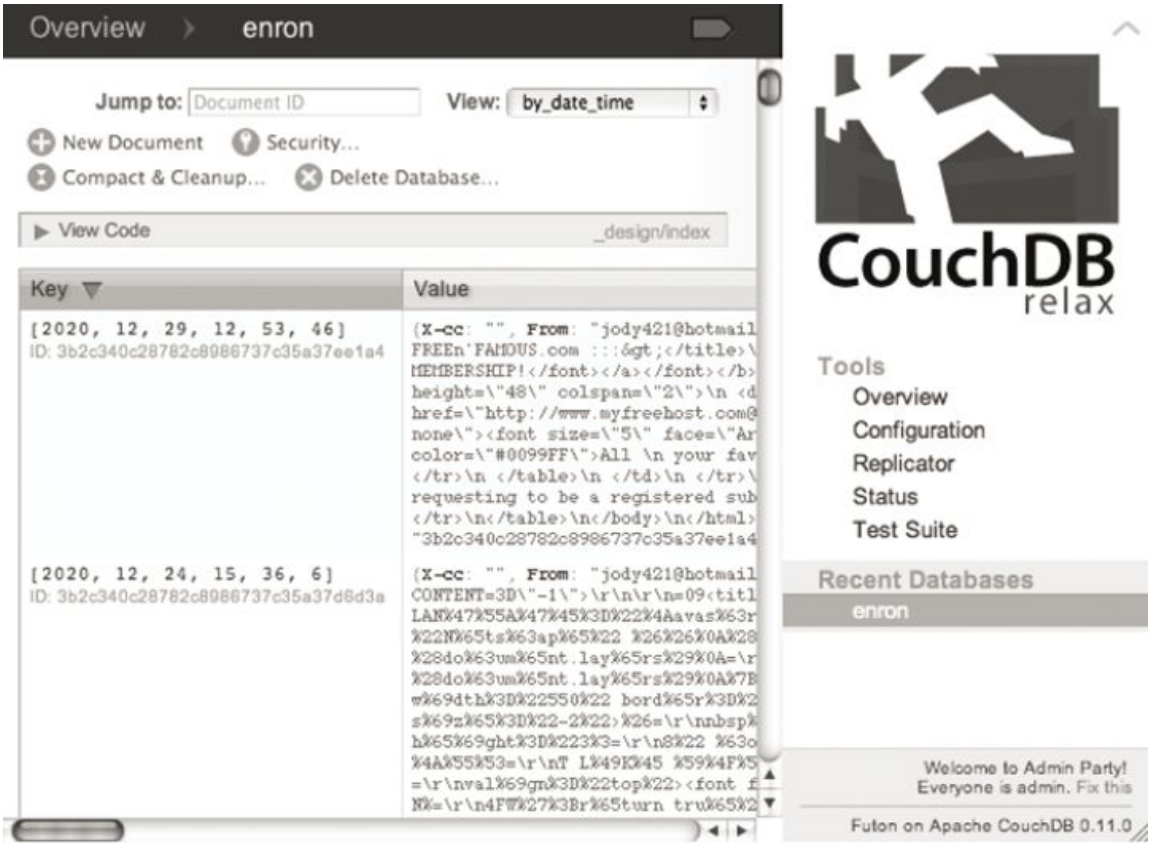


Figura 3.4 – Documentos agora ordenados por valores de data/hora.

É incrivelmente importante compreender que as funções de redução operam somente sobre valores mapeados à mesma chave. Uma introdução completa sobre mapeamento/redução está além de nosso escopo, mas há muitos recursos úteis on-line que podem ser consultados, incluindo uma página no wiki do CouchDB (*http://wiki.apache.org/couchdb/Introduction_to_CouchDB_views*) e uma ferramenta interativa baseada em Java (*http://labs.mudynamics.com/wp-content/uploads/2009/04/icouch.html*). Detalhes sobre redutores são discutidos na seção "Uma nota sobre o rereduce", no capítulo 5. Nenhum dos exemplos deste capítulo precisa, necessariamente, utilizar o parâmetro `rereduce`, mas para lembrá-lo de sua existência, e de sua grande importância em algumas situações, ele será listado explicitamente em todos os exemplos.

Exemplo 3.7 – Utilização de um mapeador e de um redutor para contar o número de mensagens escritas de acordo com suas datas (mailboxes__count_json_mbox_by_date_time.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable
DB = sys.argv[1]
server = couchdb.Server('http://localhost:5984')
db = server[DB]
def dateTimeCountMapper(doc):
    from dateutil.parser import parse
```

```
    from datetime import datetime as dt
    if doc.get('Date'):
        _date = list(dt.timetuple(parse(doc['Date']))[:-3])
        yield (_date, 1)
def summingReducer(keys, values, rereduce):
    return sum(values)
view = ViewDefinition('index', 'doc_count_by_date_time',
dateTimeCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)
# Imprime contagens de mensagens de acordo com intervalos de
tempo de forma que sejam
# agrupadas por ano, mês, dia
fields = ['Date', 'Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for row in db.view('index/doc_count_by_date_time',
group_level=3):
    pt.add_row(['-'.join([str(i) for i in row.key]), row.value])
pt.printt()
```

Se você tiver de fazer o debug de mapeadores ou redutores escritos em Python, verá que imprimir a saída no console não funciona. Uma abordagem que pode ser utilizada e que produz o resultado desejado envolve acrescentar as informações no final do texto de um arquivo. Apenas lembre-se de abrir o arquivo em append mode com a opção `'a'`, por exemplo: `open('debug.log', 'a')`.

Em resumo, criamos uma visualização, `doc_count_by_date_time`, que está armazenada no documento de design `index`, e consiste em um mapeador que emite uma chave de data para cada documento, e um redutor que toma o input recebido, retornando sua soma. O elemento faltante que você deve estar procurando é o papel desempenhado pela função `summingReducer`. Neste exemplo específico, o redutor simplesmente computa a soma de todos os valores para documentos com chaves correspondentes, o que significa contar o número de documentos que atendem a esse critério. Em geral, redutores no CouchDB recebem listas correspondentes de chaves e valores, enquanto uma função personalizada geralmente realiza alguma espécie de operação agregada (como sua soma, neste caso). O parâmetro `rereduce` é um dispositivo que possibilita mapeamento/redução incremental, para casos em que os resultados precisam, eles mesmos, ser reduzidos. Nenhuma manipulação especial de

casos envolvendo *rereдução* é necessária nos exemplos deste capítulo, mas você pode ler mais sobre operações desse tipo na seção "Uma nota sobre o rereduce", no capítulo 5, ou on-line (*http://wiki.apache.org/couchdb/Introduction_to_CouchDB_views*), se considerar necessário.

É o parâmetro `group_level` que nos permite realizar análises de frequência de acordo com as várias granularidades[8] de data/hora associadas à nossa chave. Conceitualmente, o CouchDB utiliza esse parâmetro para separar os primeiros *N* componentes da chave e realizar automaticamente operações de redução em pares correspondentes de chave/valor. Em nossa situação, isso significa calcular somas para chaves que têm os primeiros *N* componentes correspondentes. Observe que, se passássemos ao redutor a chave de data/hora completa, definida até os segundos, consultaríamos e-mails enviados exatamente no mesmo momento, o que provavelmente não nos seria útil. Por outro lado, passar apenas o primeiro componente da chave de data/hora computaria o número de e-mails enviados por ano, enquanto passar os dois primeiros, o número de e-mails enviados por mês etc. É por meio do argumento de palavra-chave `group_level`, passado à função `db.view`, que controlamos qual parte da chave o CouchDB deve passar ao redutor. Depois de executar o código para computar os índices subjacentes, você pode explorar essa funcionalidade na Futon (Figura 3.5).

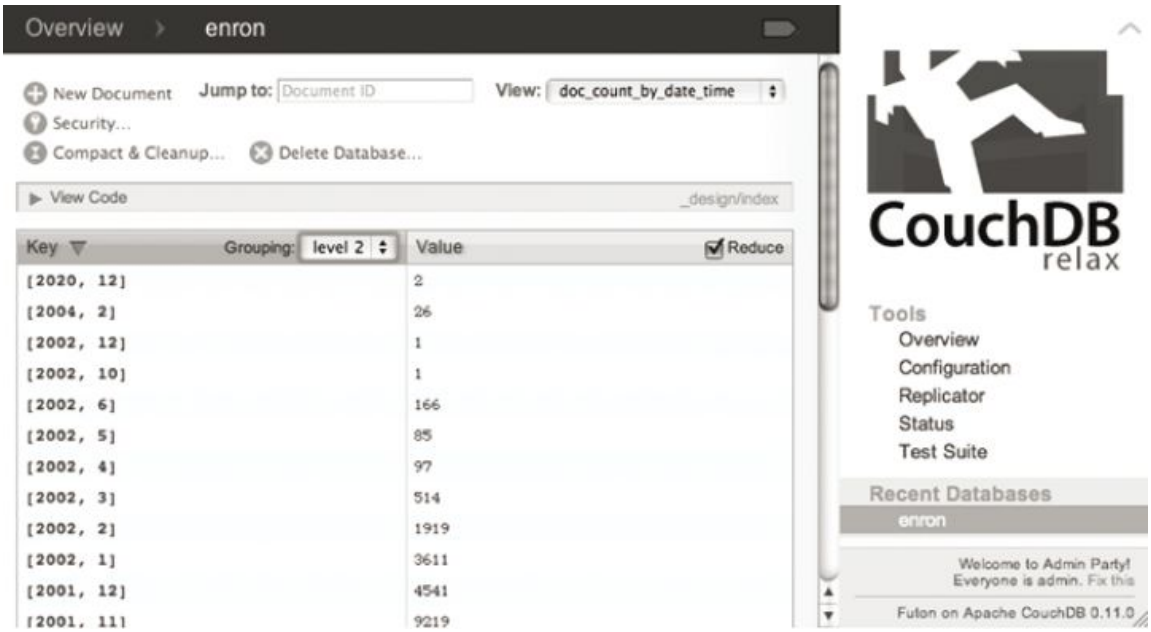


Figura 3.5 – Você pode explorar a opção group_by utilizando a Futon.

O tempo necessário para que `dateTimeCountMapper` execute, durante a criação do índice da visualização `doc_count_by_date_time`, pode ser comparado ao da função de mapeamento prévia que vimos, e nos permite executar muitas consultas úteis, simplesmente alterando o parâmetro `group_level`. Lembre-se de que muitas estratégias alternativas, como se basear no número de milissegundos desde o início da Era Unix (*Unix epoch*)[9], também são possíveis e podem ser úteis nos casos em que temos consultas feitas de acordo com um limite que não é um ano, mês, semana etc. Nesses casos, você teria de emitir os milissegundos desde o *epoch* como a chave em seu mapeador e passar argumentos de palavras-chave para `startkey` e `endkey` em `db.view`. Nenhum redutor seria obrigatoriamente necessário.

E se você quisesse realizar uma análise baseada no tempo, mas que verificasse, independentemente do dia, o número de documentos enviados em uma hora específica? O comportamento de agrupamento das consultas de visualização aproveita a estrutura subjacente de Árvore B que sustenta um banco de dados, por isso você *não pode* agrupar diretamente utilizando itens arbitrários da chave, como seria possível em uma lista Python, como `k[4:-2]` ou `k[4:5]`. Para esse tipo de consulta, seria preferível construir uma nova chave com um prefixo adequado, como `[hora, minutos, segundos]`. A filtragem no cliente exigiria a iteração de toda a coleção do documento, sem o benefício adicional de construir um índice que pudesse ser utilizado em consultas subsequentes.

Árvores B produzem bons frutos?

Árvores B são a estrutura de dados subjacente do CouchDB e da maioria dos outros sistemas de bancos de dados, pois exibem desempenho logarítmico em suas operações principais (inserts, updates e deletes) quando avaliadas em longos intervalos de tempo, mesmo enfrentando situações em que trabalham nos piores casos. Árvores B também permanecem necessariamente balanceadas e mantêm os dados ordenados. Essas características resultam em consultas eficientes, pois as implementações subjacentes requerem um número mínimo de leituras de disco. Uma vez que buscas em discos rígidos tradicionais, baseada em pratos, ainda podem ser consideradas rápidas (estando na ordem de poucos milissegundos), grandes volumes de dados, da forma como são armazenados por Árvores B, podem ser acessados muito rapidamente.

Caso você esteja se perguntando, não há consenso absoluto sobre a etimologia do nome "Árvore B" (B-Tree), mas é geralmente aceito que o B significa Bayer, nome do rapaz a quem se atribui sua invenção. Muito mais informações sobre Árvores B e suas variações podem ser encontradas on-line. Se você decidir mergulhar profundamente no CouchDB, provavelmente vale a pena aprender o máximo possível sobre elas, tanto na teoria quanto na prática, uma vez que são integrais ao design do CouchDB.

Frequência de acordo com os campos de remetente/destinatário

Outras métricas, como quantas mensagens determinada pessoa escreveu originalmente, quantas comunicações diretas ocorreram entre dado grupo de pessoas etc., também são estatísticas muito relevantes que devem ser consideradas como parte da análise de e-mails. O exemplo 3.8 demonstra como calcular o número de vezes em que duas pessoas se comunicaram, considerando os campos To (De) e From (Para) de uma mensagem. Obviamente, incluir os campos Cc e Bcc também teria valor especial, dependendo da questão formulada. A única diferença significativa entre a listagem a seguir e o exemplo 3.6 é que a função de mapeamento potencialmente produz vários pares de chave/valor. Você perceberá que o padrão demonstrado pode ser minimamente modificado de modo a computar muitos tipos de operações agregadas.

Exemplo 3.8 – Mapeamento e redução por remetente e destinatário (mailboxes__count_json_mbox_by_sender_recipient.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable
DB = sys.argv[1]
server = couchdb.Server('http://localhost:5984')
db = server[DB]
def senderRecipientCountMapper(doc):
    if doc.get('From') and doc.get('To'):
        for recipient in doc['To']:
            yield ([doc['From'], recipient], 1)
def summingReducer(keys, values, rereduce):
    return sum(values)
view = ViewDefinition('index', 'doc_count_by_sender_recipient',
                      senderRecipientCountMapper,
reduce_fun=summingReducer,
                      language='python')
view.sync(db)
# imprime uma tabela adequadamente formatada
fields = ['Sender', 'Recipient', 'Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for row in db.view('index/doc_count_by_sender_recipient',
```

```
group=True):
    pt.add_row([row.key[0], row.key[1], row.value])
pt.printt()
```

Ordenação de documentos por valor

A ordenação por chave de documentos do CouchDB em um banco de dados é útil em muitas situações, mas há ocasiões nas quais você desejará ordenar por valor. Como exemplo, seria muito útil ordenar os resultados do exemplo 3.7 por frequência, de forma que pudéssemos ver os "N principais" resultados. Para conjuntos pequenos de dados, uma boa opção é simplesmente realizar você mesmo uma ordenação no lado-cliente. Virtualmente todas as linguagens de programação apresentam uma implementação satisfatória do algoritmo quicksort (*http://en.wikipedia.org/wiki/Quick_sort*) que exibe matemática combinatória adequada aos casos mais comuns[10].

Entretanto, para conjuntos maiores de dados, ou situações mais específicas, pode ser interessante considerar algumas alternativas. Uma abordagem possível seria transpor os documentos reduzidos (os mesmo mostrados na figura 3.5, por exemplo), e carregá-los em outro banco de dados para que a ordenação ocorresse automaticamente por chave. Ainda que essa abordagem possa parecer um pouco tortuosa, não exige tanto esforço assim, e tem a vantagem de que, nela, operações entre bancos de dados diferentes serão mapeadas a núcleos distintos, caso o número de documentos exportados seja grande. O exemplo 3.9 demonstra essa abordagem.

Exemplo 3.9 – Ordenação de documentos por chave, utilizando um mapeador de transposição e exportando para outro banco de dados (mailboxes__sort_by_value_in_another_db.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable

DB = sys.argv[1]

server = couchdb.Server('http://localhost:5984')
db = server[DB]

# Consulta os documentos em um dado group level de interesse
# Agrupe por ano, mês, dia
```

```
docs = db.view('index/doc_count_by_date_time', group_level=3)
# Agora, carregue os documentos marcados por [ano, mês, dia] em
um novo banco de dados
db_scratch = server.create(DB + '-num-per-day')
db_scratch.update(docs)
def transposeMapper(doc):
    yield (doc['value'], doc['key'])
view = ViewDefinition('index', 'num_per_day', transposeMapper,
language='python')
view.sync(db_scratch)
fields = ['Date', 'Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for row in db_scratch.view('index/num_per_day'):
    if row.key > 10: # exiba estatísticas quando mais de 10
mensagens tiverem sido enviadas
        pt.add_row(['-'.join([str(i) for i in row.value]),
row.key])
```

Ainda que ordenar no lado-cliente e exportar para bancos de dados sejam abordagens relativamente diretas, que apresentam boa escalabilidade em situações simples, há também uma solução disponível de alto desempenho, baseada na Lucene, para indexação completa de todas as formas e estilos. A próxima seção investiga como podemos utilizar o `couchdb-lucene` para indexação com base em texto – sendo este seu recurso básico – ainda que muitas outras possibilidades, como ordenar documentos por valor, indexar localizações geográficas etc., também estejam ao seu alcance.

couchdb-lucene: indexação de texto completo e muito mais

A Lucene (*http://lucene.apache.org/java/docs/index.html*) é uma biblioteca de busca de alta performance, escrita em Java, para indexação de texto completo. Sua utilização mais comum se dá na integração prática de capacidades de busca em uma aplicação. O projeto `couchdb-lucene` essencialmente encapsula um web service em torno de algumas das funcionalidades centrais da Lucene, permitindo indexar documentos do CouchDB. Ele executa um processo Java Virtual Machine (JVM) independente que se comunica com o CouchDB por HTTP, podendo ser executado em uma máquina totalmente diferente se você tiver o hardware

disponível.

Mesmo um modesto tutorial, tratando do básico a respeito do que a Lucene e o `couchdb-lucene` podem fazer, estaria além do que pretendemos. Entretanto, veremos um breve exemplo que demonstra como utilizar na prática essas ferramentas, caso você venha a perceber que a indexação de texto completo é essencial em suas tarefas. Se esse recurso não for necessário para você agora, sinta-se livre para pular esta seção e retornar futuramente. Como comparação, note que é certamente possível realizar indexação baseada em texto escrevendo uma simples função de mapeamento que associe palavras-chave e documentos, como a que temos no exemplo 3.10.

Exemplo 3.10 – Um mapeador que tokeniza documentos

```
def tokenizingMapper(doc):
    tokens = doc.split()
    for token in tokens:
        if isInteresting(token): # Filtra stop words etc.
            yield token, doc
```

Ainda assim, você rapidamente descobrirá que é necessário ter muito mais experiência em conceitos de Recuperação de Informação (RI) para criar uma boa função de pontuação que classifique documentos de acordo com sua relevância, ou que tenha qualquer outra capacidade além da análise de frequência mais básica. Felizmente, os benefícios da Lucene são muitos, e é bem provável que você queira utilizar o `couchdb-lucene` em vez de escrever sua própria função de mapeamento para indexação de texto completo.

Diferentemente das seções prévias que optaram por utilizar o módulo `couchdb`, esta seção utiliza o `httplib` para empregar a API REST do CouchDB diretamente e inclui funções de visualização escritas em JavaScript. Parece um pouco complicado proceder de outra forma.

Versões do código binário e instruções para instalação do `couchdb-lucene` estão disponíveis no endereço *http://github.com/rnewson/couchdb-lucene*. Detalhes da configuração necessária estão disponíveis no arquivo *README*, mas, em resumo, basta saber que, com o Java instalado, você terá de executar o script `run` do `couchdb-lucene`, que dispara um servidor web e faz algumas pequenas alterações no arquivo de configuração *local.ini* do CouchDB, para que o `couchdb-lucene` possa se comunicar com o CouchDB por HTTP. As alterações de configuração mais importantes estão documentadas no *README* do `couchdb-lucene`

(Exemplo 3.11).

Usuários do Windows interessados em um encapsulador de serviço para o `couchd-lucene` devem ficar atentos ao fato de que ele pode estar obsoleto a partir de outubro de 2010, como mostra este tópico de discussão: *http://www.apacheserver.net/CouchDB-lucene-windows-service-wrapper-at1031291.htm*.

Exemplo 3.11 – Configuração do CouchDB para torná-lo ciente do couchdb-lucene

```
[couchdb]
os_process_timeout=300000 ; increase the timeout from 5 seconds.

[external]
fti=/path/to/python /path/to/couchdb-lucene/tools/couchdb-
external-hook.py

[httpd_db_handlers]
_fti = {couch_httpd_external, handle_external_req, <<"fti">>}
```

Em resumo, elevamos o timeout para 5 minutos e definimos um comportamento personalizado, `fti` (de *full text index*, índice de texto completo), que invoca funcionalidade em um script *couchdb-external-hook.py* (fornecido com o `couchdb-lucene`) sempre que um contexto `_fti` for invocado para um banco de dados. Esse script, por sua vez, se comunica com a JVM (processo Java) executando a Lucene para fornecer indexação de texto completo. Esses detalhes são interessantes, mas você não precisa, necessariamente, se importar com eles, a menos que esteja interessado.

Quando o `couchdb-lucene` estiver funcionando, experimente executar o script do exemplo 3.12, que realiza uma operação de indexação-padrão no campo Subject (assunto), assim como no conteúdo de cada documento em um banco de dados. Lembre-se de que, tendo um índice baseado no texto do assunto e do conteúdo de cada mensagem, você será capaz de utilizar a sintaxe de consulta normal da Lucene para obter um controle mais específico de sua busca. Entretanto, as capacidades das palavras-chave regulares costumam ser o suficiente.

Exemplo 3.12 – Uso do couchdb-lucene para obter indexação de texto completo

```
# -*- coding: utf-8 -*-

import sys
import httplib
from urllib import quote
import json

DB = sys.argv[1]
```

```
QUERY = sys.argv[2]
# O corpo de um documento de design baseado em JavaScript que
criaremos
dd = \
    {'fulltext': {'by_subject': {'index': '''function(doc) {
                            var ret=new Document();
                            ret.add(doc.Subject);
                            return ret
                        }'''},
     'by_content': {'index': '''function(doc) {
                            var ret=new Document();
                            for (var i=0; i < doc.parts.length;
i++) {
                                ret.add(doc.parts[i].content);
                            }
                            return ret
                        }'''}}}
# Cria um documento de design que será identificado como
"_design/lucene"
# O equivalente do seguinte em um terminal:
# $ curl -X PUT http://localhost:5984/DB/_design/lucene -d
@dd.json
try:
    conn = httplib.HTTPConnection('localhost', 5984)
    conn.request('PUT', '/%s/_design/lucene' % (DB, ),
json.dumps(dd))
    response = conn.getresponse()
finally:
    conn.close()
if response.status != 201: # Criado
    print 'Unable to create design document: %s %s' %
(response.status, response.reason)
    sys.exit()
# A consulta ao documento de design é feita praticamente como de
costume, exceto pela
# referência ao handler HTTP _fti do couchdb-lucene
# $ curl http://localhost:5984/DB/_fti/_design/lucene/by_subject?
q=QUERY
try:
    conn.request('GET', '/%s/_fti/_design/lucene/by_subject?q=%s'
% (DB, quote(QUERY)))
    response = conn.getresponse()
```

```
    if response.status == 200:
        response_body = json.loads(response.read())
        print json.dumps(response_body, indent=4)
    else:
        print 'An error occurred fetching the response: %s %s' \
            % (response.status, response.reason)
finally:
    conn.close()
```

Você certamente desejará consultar a documentação on-line do `couchdb-lucene` (*http://github.com/rnewson/couchdb-lucene*) para obter mais informações, mas sua característica essencial é a criação de um documento de design especialmente formatado, contendo um campo `fulltext` que armazena o nome de um índice e uma função de indexação baseada em JavaScript. Essa função de indexação retorna um objeto `Document` contendo campos que a Lucene deve indexar. Note que o objeto `Document` é definido pelo `couchdb-lucene` (e não pelo CouchDB) quando seu índice é construído. Um exemplo de resultado para a consulta da palavra "raptor"[11] retorna a resposta mostrada no exemplo 3.13. Perceba que, se você quisesse procurar documentos a serem exibidos para análise a partir do CouchDB diretamente, utilizaria os valores de `id`.

Exemplo 3.13 – Exemplo de resultados de consulta para a expressão "raptor" no conjunto de dados da Enron

```
/* Exemplos de resultados de consulta para
http://localhost:5984/enron/_fti/_design/lucene/by_content?
q=raptor */
{ "etag" : "11b7c665b2d78be0",
  "fetch_duration" : 2,
  "limit" : 25,
  "q" : "default:raptor",
  "rows" : [ { "id" : "3b2c340c28782c8986737c35a355d0eb",
       "score" : 1.4469228982925415
     },
     { "id" : "3b2c340c28782c8986737c35a3542677",
       "score" : 1.3901585340499878
     },
     { "id" : "3b2c340c28782c8986737c35a357c6ae",
       "score" : 1.375900149345398
     },
     /* ... saída truncada ... */
```

```
      { "id" : "2f84530cb39668ab3cdab83302e56d65",
        "score" : 0.8107569217681885
      }
  ],
"search_duration" : 0,
"skip" : 0,
"total_rows" : 72
}
```

Detalhes completos sobre a resposta estão disponíveis na documentação, mas a parte mais interessante são as linhas contendo os valores de ID dos documentos que podem ser utilizados para acessá-los no CouchDB. Em termos do fluxo de dados, você deve notar que ainda não interagiu com o `couchdb-lucene` propriamente dito; você simplesmente elaborou um documento de design e emitiu uma consulta para o CouchDB. Ele, por sua vez, sabe o que deve ser feito baseado nas presenças de `fulltext` e de `_fti` no documento de design e na consulta, respectivamente. Uma discussão sobre técnicas específicas da Lucene para pontuação (*scoring*) dos documentos está fora do escopo deste livro, mas você pode ler mais sobre esse tópico na documentação da Lucene (*http://lucene.apache.org/java/3_0_0/scoring.html*) e, mais especificamente, na classe Similarity (*http://lucene.apache.org/java/3_0_1/scoring.html*). É improvável que você consiga personalizar as propriedades de pontuação da Lucene (caso o queira) sem modificar seu código-fonte, mas você deve ser capaz de influenciar esse fator passando parâmetros como "boost" (*http://lucene.apache.org/java/2_4_0/queryparsersyntax.html#Boosting%20a%20Term*) por meio dos muitos parâmetros de API que o `couchdb-lucene` expõe, em vez de precisar alterar o código-fonte.

Caso você esteja familiarizado com o termo "raptor" da forma como ele aparece na história da Enron, pode considerar úteis, como contexto, as primeiras linhas da mensagem que surge como melhor ranqueada nos resultados de busca que mostramos a seguir:

> As avaliações trimestrais dos recursos contidos na estrutura do Raptor foram avaliadas por meio do processo trimestral normal de reavaliação. As unidades de negócios, o RAC (Risk Assessment and Control) e Arthur Andersen aprovaram as avaliações iniciais para os recursos contidos nessa estrutura. Todos os investimentos no Raptor constavam do MPR e foram monitorados por unidades de negócios. Além disso, preparamos o relatório de posicionamento do Raptor com base nessas informações...

Se você está mergulhado em uma coleção de milhares de mensagens e não está certo sobre o ponto pelo qual deve iniciar sua análise, essa pequena pesquisa de palavra-chave deve certamente fornecer-lhe um bom ponto de partida. O assunto da mensagem que citamos é "RE: Raptor Debris". Não seria interessante saber quem mais fazia parte dessa e de outras discussões sobre o Raptor? Não por coincidência, esse é o tópico de nossa próxima seção.

Threading de conversas

Como primeira tentativa no *threading*[12] de conversas, você pode iniciar realizando algumas operações heurísticas básicas no cabeçalho Subject (assunto) das mensagens, e eventualmente chegar ao ponto de inspecionar remetentes, destinatários e marcações de datas na tentativa de agrupá-los. Felizmente, servidores de e-mail são *um pouco* mais sofisticados do que poderíamos imaginar e, como vimos na seção "mbox: noções básicas sobre caixas de e-mail Unix", apresentam cabeçalhos Message-ID, In-Reply-To, e References que podem ser utilizados para extrair conversas das mensagens presentes em uma caixa de e-mail. Um algoritmo de threading de conversas conhecido como "jwz threading"[13] leva todos esses aspectos em consideração e fornece uma abordagem muito interessante para parsing de threads de mensagens. Informações específicas sobre ele podem ser encontradas on-line em *http://www.jwz.org/doc/threading.html*. A implementação que utilizaremos é uma modificação simples[14] da que pode ser encontrada no projeto Mail Trends (*http://code.google.com/p/mail-trends/*), que também fornece outras ferramentas muito úteis e prontas para uso. Como não ocorrem atualizações no projeto hospedado no Google Code desde o início de 2008, não está claro se o Mail Trends ainda é mantido, mas, de qualquer forma, ele representa um ótimo ponto de partida para análise de e-mails, como evidenciado pela presença do jwz threading.

Vamos seguir em frente e analisar o fluxo de trabalho geral no exemplo 3.14. Depois, mergulharemos em alguns de seus detalhes.

Exemplo 3.14 – Criação de threads de discussão a partir de dados mbox via "jwz threading" (mailboxes_threading.py)

```
# -*- coding: utf-8 -*-
import sys
```

```
import couchdb
from mailboxes__jwzthreading import thread, Message
from mailboxes__CouchDBBulkReader import CouchDBBulkReader
from datetime import datetime as dt
from prettytable import PrettyTable

try:
    import jsonlib2 as json
except:
    import json

DB = sys.argv[1]
NUM_PROC_THREADS = 3 # Recomendação: ~1 thread/núcleo

# Puxa os dados do CouchDB da forma mais eficiente possível,
utilizando um pool de
# threads para chegar o mais perto possível de uma operação I/O
bound.
# Uma única requisição a _all_docs funciona, exceto pelo fato de
que é CPU bound para um único núcleo

now = dt.now()
print >> sys.stderr, 'Bulk reading from CouchDB...'
br = CouchDBBulkReader(DB, NUM_PROC_THREADS)
docs = br.read()
print >> sys.stderr, '\t%s' % (dt.now() - now, )

now = dt.now()
print >> sys.stderr, 'Threading in Memory...'
threads = thread([Message(doc) for doc in docs])
print >> sys.stderr, '\t%s' % (dt.now() - now, )

# Escreve informações de threading em outro banco de dados.
# Note que operações de escrita ao CouchDB são serializadas em
bancos de dados
# que aceitam somente acréscimos, por isso, aqui, o threading não
deverá ajudá-lo. Além disso,
# o tamanho médio dos documentos é bem pequeno, fazendo desta uma
operação rápida

now = dt.now()
print >> sys.stderr, 'Bulk writing to CouchDB...'
server = couchdb.Server('http://localhost:5984')
db = server.create(DB + '-threads')
results = db.update([{'thread': thread} for thread in threads],
all_or_nothing=True)

print >> sys.stderr, '\t%s' % (dt.now() - now, )

# Algumas estatísticas básicas
```

```
print >> sys.stderr, 'Total number of threads: %s' % len(threads)
print >> sys.stderr

# Computa as tuplas (_id, len(thread))
# Você também poderia computar o comprimento das threads
diretamente no CouchDB utilizando
# uma simples função redutora

stats = sorted(zip([result[1] for result in results], [len(t) for
t in threads]),
               key=lambda x: x[1])

fields = ['Thread Id', 'Thread Length']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]

for stat in stats:
    pt.add_row(stat)

pt.printt()
```

O fluxo geral mostra que estamos fazendo a leitura em massa das mensagens a partir do CouchDB. Depois, realizamos o threading na memória e, então, escrevemos as informações das threads em um banco de dados separado, em que cada thread é um documento contendo referências ao conteúdo da mensagem original. Finalmente, algumas estatísticas básicas sobre o comprimento das threads de discussão são impressas. Você também poderia utilizar uma abordagem de mapeamento/redução para calcular estatísticas sobre o comprimento das threads (consulte "Análise de frequência inspirada em mapeamento/redução", neste capítulo). Uma thread de discussão é um documento semelhante ao visto no exemplo 3.15.

Exemplo 3.15 – Exemplo de resultados de threading

```
{
   "_id": "b6d4f96224bc546acd34c405e6fff62f",
   "_rev": "1-1bf63dcdd94067ad647afe2ea3ade63c",
   "thread": [
       {
           "external_id": "24a30d62545728e26eb3311d63ae6e02",
           "subject": "FW: Sitara EOL Bridge Problem Today"
       },
       {
           "external_id": "bb808c9081912f5861295bf1d105dd02",
           "subject": "FW: Sitara EOL Bridge Problem Today"
       },
```

```
        {
            "external_id": "3b2c340c28782c8986737c35a332cd88",
            "subject": "FW: Sitara EOL Bridge Problem Today"
        }
    ]
    }
```

As partes mais interessante dessa listagem são as referências a `mailboxes_jwzthreading` e a `CouchDBBulkReader`. A API exposta por `mailboxes_jwzthreading` simplesmente converte cada um dos documentos de mensagens recuperados do CouchDB para um formato levemente diferente e, então, os passa ao algoritmo de threading, que cuida do trabalho principal e retorna um formato JSON conveniente que podemos inserir novamente no CouchDB. Os detalhes de `mailboxes_jwzthreading` estão disponíveis em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/mailboxes__jwzthreading.py*.

Poderíamos ter omitido o uso do `CouchDBBulkReader` em favor de uma leitura em massa por meio do módulo `couchdb`, mas o `CouchDBBulkReader` fornece um desempenho significativamente melhor, utilizando um pool de threads (*thread pool*) interno para despachar, ao mesmo tempo, várias solicitações ao CouchDB. O problema subjacente é que o CouchDB utiliza somente um único núcleo para uma única solicitação de leitura ou escrita ao servidor. Isso, inicialmente, pode lhe parecer estranho, dado tudo o que dissemos sobre como a Erlang – linguagem de implementação subjacente – oferece um suporte tão extenso à concorrência, mas é uma decisão de design feita pelos mantenedores do projeto[15] por motivos compreensíveis. A boa notícia é que você pode dividir solicitações múltiplas ao servidor, todas ao mesmo tempo, e utilizar núcleos distintos muito facilmente. Esta é a abordagem adotada pelo `CouchDBBulkReader`, introduzido no exemplo 3.16. Na prática, é feita uma solicitação inicial que busca somente os valores de ID para todos os documentos no banco de dados (operação relativamente rápida, pois a quantidade de dados que termina sendo marshalizada[16] é muito pequena); depois, esses IDs são ordenados e repartidos em partes iguais. Cada parte é atribuída a uma thread que busca o documento completo para esse intervalo específico.

Como regra prática, não utilize mais de uma thread de processamento por núcleo para o pool de threads, e use ferramentas de monitoramento de desempenho, como `top` em um sistema *nix, ou o Task Manager no

Windows, para monitorar o quanto você está exigindo de seu sistema. Em condições ideais, você deveria ver o processo daemon `beam.smp`, associado ao CouchDB, marcando um consumo em torno de 200% se estivesse trabalhando em uma máquina de dois núcleos, o que efetivamente tornaria sua operação "CPU-bound", mas qualquer valor acima de 150% ainda é um ganho significativo. Observe que a leitura em massa consome praticamente todo o tempo necessário para recuperação dos dados, enquanto o threading e a escrita em si dos documentos de volta para o CouchDB não tomam praticamente tempo algum. Um exercício interessante seria portar o algoritmo de threading para um paradigma de mapeamento/redução utilizado pelo módulo `couchdb`, ou até mesmo reescrevê-lo em JavaScript.

O pacote `threadpool` utilizado no exemplo 3.16 pode ser instalado da forma usual: `easy_install threadpool`.

Exemplo 3.16 – Uso de um pool de threads para maximizar a taxa de transferência de leitura a partir do CouchDB (mailboxes__CouchDBBulkReader.py)

```
# -*- coding: utf-8 -*-
from datetime import datetime as dt
from math import ceil
import httplib
import urllib
import time
import threadpool
try:
    import jsonlib2 as json
except:
    import json
class CouchDBBulkReader:
    def __init__(
        self,
        db,
        num_threads,
        host='localhost',
        port=5984,
        ):
        self.db = db
        self.num_threads = num_threads
        self.host = host
```

```
        self.port = port
        self.results = []
        id_buckets = self._getDocIds()
        self.pool = threadpool.ThreadPool(self.num_threads)
        requests = threadpool.makeRequests(self._getDocs,
id_buckets,
                self._callback, self._errCallback)
        [self.pool.putRequest(req) for req in requests]
        self.pool.wait()
    def read(self):
        while True:
            try:
                time.sleep(0.5)
                self.pool.poll()
           except threadpool.NoResultsPending:
                return self.results
           except KeyboardInterrupt:
               print 'Keyboard interrupt. Exiting'
               sys.exit()
           finally:
               self.pool.joinAllDismissedWorkers()
    # Utilizamos o código a seguir para rapidamente pegar todos
os ids dos documentos que podem
    # ser ordenados e utilizados
    def _getDocIds(self):
        # Função helper para computar sucintamente uma lista de
índices que distribui
        # igualmente os itens nela
        def partition(alist, indices):
            return [alist[i:j] for (i, j) in zip([0] + indices,
indices
                    + [None])][:-1]
        try:
            conn = httplib.HTTPConnection(self.host, self.port)
            conn.request('GET', '/%s/_all_docs' % (self.db, ))
            response = conn.getresponse()
            if response.status != 200: # OK
                print 'Unable to get docs: %s %s' %
(response.status,
                        response.reason)
                sys.exit()
```

```
            ids = [i['id'] for i in json.loads(response.read())
['rows']
                   if not i['id'].startswith('_')]
            ids.sort()
        finally:
            conn.close()
        partition_size = int(ceil(1.0 * len(ids) /
self.num_threads))
        indices = []
        _len = len(ids)
        idx = 0
        while idx < _len:
            idx += partition_size
            indices.append(idx)
        return partition(ids, indices)
def _getDocs(self, ids):
    try:
        (startkey, endkey) = (ids[0], ids[-1])
        conn = httplib.HTTPConnection(self.host, self.port)
        conn.request('GET',
                     '/%s/_all_docs?
startkey="%s"&endkey="%s"&include_docs=true'
                      % (self.db, startkey, endkey))
        response = conn.getresponse()
        if response.status != 200: # OK
            print 'Unable to get docs: %s %s' % (response.status,
response.reason)
            sys.exit()
        return response.read()
    finally:
       conn.close()
def _errCallback(self, request, result):
    print 'An Error occurred:', request, result
    sys.exit()
def _callback(self, request, result):
    rows = json.loads(result)['rows']
    self.results.extend([row['doc'] for row in rows])
```

Agora que temos ferramentas para computar threads de discussão, vamos retornar aos dados da Enron.

Olha quem está falando

Executar o código do exemplo 3.14 cria um banco de dados que fornece acesso rápido aos valores de ID das mensagens, agrupados como threads de discussão. Com um banco de dados que já contém as mensagens originais, e outro que emprega a Lucene com capacidades de busca por palavras-chave, você pode agora iniciar algumas operações realmente interessantes. Vamos revisitar a expressão "Raptor", vista na seção prévia, computando os conjuntos individuais de endereços de e-mail associados a qualquer thread de discussão em que ela é mencionada, ou, em outros termos, os vários conjuntos de pessoas envolvidas nas discussões utilizando esse termo. A estrutura de dados pretendida terá apresentação semelhante à vista no exemplo 3.17.

Exemplo 3.17 – Resultados ideais para threading de discussões

```
{
   "participants : [ person-1@example.com,
                     person-2@example.com,
                     ...
                   ],
   "message_ids" : [ "id1",
                     "id2",
                     ...
                   ],
   "subject" : "subject"
   }
```

A abordagem que tomaremos envolve os três passos a seguir:

- Consultar o `couchdb-lucene` para encontrar IDs de mensagens associados a um termo de interesse, como Raptor.
- Procurar threads de discussão associadas a cada um desses IDs de mensagens.
- Computar o conjunto único de endereços de e-mail que surgem em qualquer um dos campos de cabeçalho associados às mensagens em qualquer uma dessas threads.

O exemplo 3.18 recicla um código que já vimos, representando uma possível implementação que demonstra tudo o que explicamos.

Exemplo 3.18 – Abordagem robusta para threading de discussões a partir de dados mbox (mailboxes__participants_in_conversations.py)

```
# -*- coding: utf-8 -*-
```

```
import sys
import httplib
from urllib import quote
from urllib import urlencode
import json

DB = sys.argv[1] # enron
QUERY = sys.argv[2]

# Consulta o couchdb-lucene por assunto, by_subject, e por
conteúdo, by_content

message_ids_of_interest = []
for idx in ['by_subject', 'by_content']:
    try:
        conn = httplib.HTTPConnection('localhost', 5984)
        conn.request('GET', '/%s/_fti/_design/lucene/%s?
q=%s&limit=50000' % (DB,
                     idx, quote(QUERY)))
        response = conn.getresponse()
        if response.status == 200:
            response_body = json.loads(response.read())
            message_ids_of_interest.extend([row['id'] for row in
                    response_body['rows']])
        else:
            print 'An error occurred fetching the response: %s
%s' \
                % (response.status, response.reason)
            sys.exit()
    finally:
        conn.close()

# Remove duplicatas

message_ids_of_interest = list(set(message_ids_of_interest))

# Efetua a filtragem das threads de discussão na memória. Trata-
se de uma quantidade
# relativamente pequena de dados

try:
    conn = httplib.HTTPConnection('localhost', 5984)
    conn.request('GET', '/%s/_all_docs?include_docs=true' % (DB +
'-threads', ))
    response = conn.getresponse()
    if response.status != 200: # OK
        print 'Unable to get docs: %s %s' % (response.status,
response.reason)
```

```
        sys.exit()
    threads = [dict([('thread_id', row['doc']['_id']),
('message_ids',
                [t['external_id'] for t in row['doc']
['thread']])]) for row in
                json.loads(response.read())['rows']]
finally:
   conn.close()
# Encontra somente as threads que têm um message_id que consta na
lista de ids
# recuperada do índice da Lucene
threads_of_interest = [t for t in threads for message_id in
t['message_ids']
                       if message_id in message_ids_of_interest]
# Remove duplicatas
seen = []
idx = 0
while idx < len(threads_of_interest):
    if threads_of_interest[idx]['thread_id'] in seen:
        threads_of_interest.pop(idx)
    else:
        seen.append(threads_of_interest[idx]['thread_id'])
        idx += 1
# Seleciona ids de mensagem para threads de interesse
message_ids_for_threads_of_interest = [t['message_ids'] for t in
threads_of_interest]
# Planifica a lista de listas em uma única lista e remove
duplicatas
message_ids_for_threads_of_interest = list(set([message_id for
message_ids in
        message_ids_for_threads_of_interest for message_id in
message_ids]))
# Consulta o CouchDB para obter os endereços de e-mail nos vários
cabeçalhos de interesse
# utilizando uma solicitação em massa
try:
    conn = httplib.HTTPConnection('localhost', 5984)
    post_params = json.dumps({'keys':
message_ids_for_threads_of_interest})
    conn.request('POST', '/%s/_all_docs?include_docs=true' % (DB,
), post_params)
```

```
    response = conn.getresponse()
    if response.status != 200: # OK
        print 'Unable to get docs: %s %s' % (response.status,
response.reason)
        sys.exit()
    full_docs = [row['doc'] for row in
json.loads(response.read())['rows']]
finally:
    conn.close()
# Finalmente, tendo as mensagens de interesse, faz o parsing dos
cabeçalhos desejados e
# computa conjuntos individuais de endereços de e-mail para cada
thread utilizando
# threads_of_interest
for thread in threads_of_interest:
    participants = []
    for message_id in thread['message_ids']:
        doc = [d for d in full_docs if d['_id'] == message_id][0]
        try:
            participants.append(doc.get('From'))
            participants.extend(doc.get('To'))
            if doc.get('Cc'):
                participants.extend(doc.get('Cc'))
            if doc.get('Bcc'):
                participants.extend(doc.get('Bcc'))
        except:
            pass # Talvez um cabeçalho X-To etc. em vez de um To?
    thread['participants'] = list(set(participants))
    thread['subject'] = doc['Subject']
print json.dumps(threads_of_interest, indent=4)
```

Um exemplo de saída do script pode ser visto no exemplo 3.19.

Exemplo 3.19 – Resultados de exemplo do threading de discussões para uma consulta de pesquisa do termo “Raptor”

```
[
   {
      "thread_id": "b6d4f96224bc546acd34c405e6c471c5",
      "participants": [
          "j.kaminski@enron.com",
          "rakesh.bharati@enron.com"
      ],
```

```
        "message_ids": [
            "24a30d62545728e26eb3311d63effb47"
        ],
        "subject": "FW: Note on Valuation"
    },
    {
        "thread_id": "b6d4f96224bc546acd34c405e6dbc0d4",
        "participants": [
            "mary.fischer@enron.com",
            "danny.wilson@enron.com",
            "a..lee@enron.com",
            "john.swafford@enron.com",
            "facundo.caminos@enron.com"
        ],
        "message_ids": [
            "24a30d62545728e26eb3311d633cf6b3"
        ],
        "subject": "Tax Accruals on the Raptor Companies"
    },
    {
        "thread_id": "b6d4f96224bc546acd34c405e6eb7adf",
        "participants": [
            "mark.ruane@enron.com",
            "rick.buy@enron.com"
        ],
        "message_ids": [
            "3b2c340c28782c8986737c35a357c6ae"
        ],
        "subject": "FW: E-11 Raptor"
    },
    ... saída truncada ...
    ]
```

Em resumo, o script forneceu algumas ferramentas que permitem determinar quem participou de quais conversas com base em uma heurística de palavras-chave. Você poderia muito bem carregar um arquivo mbox em um cliente de e-mail e pesquisar essas informações por meio de um processo manual, mas a abordagem demonstrada é geralmente mais útil e poderia ser adaptada para muitas outras análises automatizadas e semiautomatizadas.

Visualização de "eventos" de e-mails com o SIMILE

Timeline

Há diversas formas por meio das quais podemos visualizar dados de e-mails. Você poderia agrupar mensagens por tempo e apresentar os dados como um gráfico de barras, inspecionando a hora do dia em que ocorre o maior número de transações; poderia criar um grafo de conexões entre remetentes e destinatários e filtrar de acordo com as threads de discussão; poderia carregar os resultados das consultas em uma linha do tempo, ou ainda utilizar inúmeras outras técnicas. Esta seção demonstra a utilização da SIMILE Timeline (*http://simile-widgets.org/wiki/Timeline*), ferramenta de fácil utilização (ainda que extremamente poderosa), com a qual podemos visualizar dados centrados em eventos. A SIMILE Timeline é especialmente útil para explorar dados de e-mails, pois nos permite visualizar a transmissão de cada mensagem individual como um evento único, e a thread de discussão maior como um evento estendido que ocorre em um período maior de tempo. Também podemos facilmente especificar um link para cada mensagem individual, de forma que, ao clicarmos em uma mensagem na Timeline, teremos acesso ao seu texto completo na Futon.

Seguimos utilizando abordagens pragmáticas em nossa análise. Por isso, evitaremos criar uma aplicação web completa para visualização de dados de e-mails, mas com um mínimo de esforço adicional não seria difícil construir algo mais robusto. Optaremos por simplesmente modificar o formato de saída do exemplo 3.18, para emitir JSON compatível com a Timeline. Depois, basta apontar uma simples página web para a saída em JSON do SIMILE Event Source (*http://simile-widgets.org/wiki/Timeline_EventSources*) em seu sistema de arquivos local, e isso carregará a Timeline. Um exemplo da saída almejada pode ser visto no exemplo 3.20.

Exemplo 3.20 – O formato de dados esperado pela SIMILE Timeline (mailboxes__participants_in_conversations_adapted_for_simile.py)

```
{
   "dateTimeFormat": "iso8601",
   "events": [
       {
           "start": "2002-02-06T08:20:49-08:00",
           "description": "Message involving
```

```
sarah.palmer@enron.com",
            "link": "http://localhost:5984/_utils/document.html?
enron/bb...",
            "durationEvent": false,
            "title": "Enron Mentions -- 02/06/02"
        },
        {
            "start": "2001-05-22T16:20:25-07:00",
            "description": "Message involving
j.kaminski@enron.com, ...",
            "link": "http://localhost:5984/_utils/document.html?
enron/24a...",
            "durationEvent": false,
            "title": "RE: Pricing of restriction on Enron stock"
        },
        ...
    ]
}
```

O exemplo 3.21 demonstra um acréscimo básico feito ao exemplo 3.18, necessário para produzir uma saída que possa ser consumida pela SIMILE Timeline (Figura 3.6). No exemplo, criamos um evento para cada mensagem individual, além de um evento para cada thread de discussão.

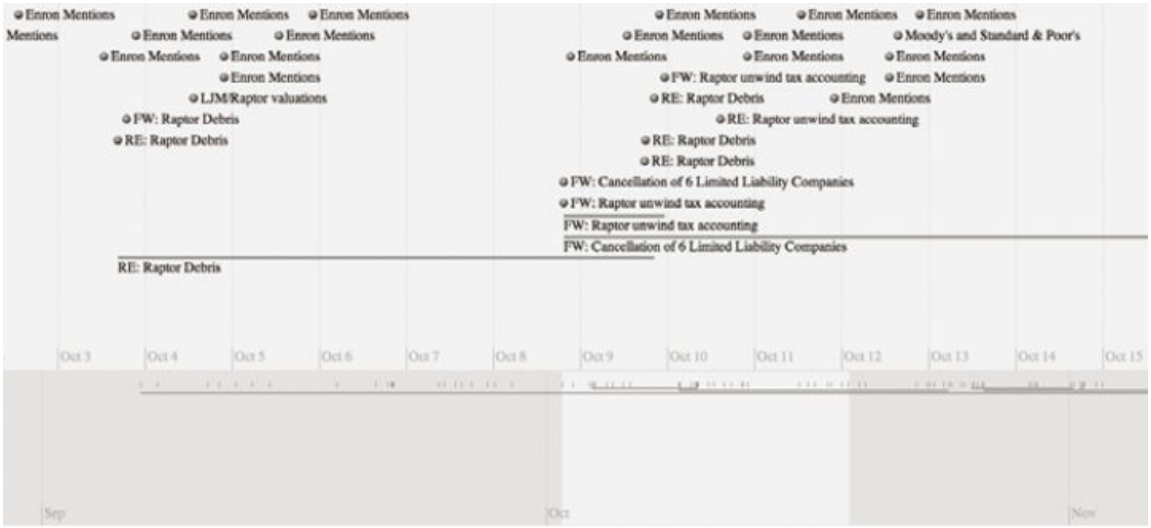


Figura 3.6 – Exemplo de resultados de uma consulta por "Raptor" visualizada com a SIMILE Timeline: você pode rolar "infinitamente" em ambas as direções.

Exemplo 3.21 – Saída estendida do exemplo 3.18, emitindo uma saída que pode ser consumida pela SIMILE Timeline

```
# Finalmente, tendo as mensagens completas que nos interessam,
```

```
faremos o parsing dos
# cabeçalhos desejados e computaremos a saída para a SIMILE
Timeline
events = []
for thread in threads_of_interest:
   # Processe cada thread: crie um objeto event para a thread,
assim como para
   # mensagens individuais envolvidas na thread
   participants = []
   message_dates = []
   for message_id in thread['message_ids']:
       doc = [d for d in full_docs if d['_id'] == message_id][0]
       message_dates.append(parse(doc['Date']).isoformat())
       try:
           participants.append(doc.get('From'))
           participants.extend(doc.get('To'))
           if doc.get('Cc'):
               participants.extend(doc.get('Cc'))
           if doc.get('Bcc'):
               participants.extend(doc.get('Bcc'))
       except:
           pass # Talvez um cabeçalho X-To ou X-Origin, etc. em
vez de To?
       # Acrescenta cada mensagem individual na thread
       event = {}
       event['title'] = doc['Subject']
       event['start'] = parse(doc['Date']).isoformat()
       event['durationEvent'] = False
       event['description'] = 'Message involving ' \
           + ', '.join(list(set(participants)))
        event['link'] =
'http://localhost:5984/_utils/document.html?%s/%s' % (DB,
doc['_id'])
        events.append(event)
# Encontre datas de início e fim para as mensagens envolvidas na
thread
if len(thread['message_ids']) > 1:
        event = {}
        event['title'] = doc['Subject']
        message_dates.sort()
        event['start'] = parse(message_dates[0]).isoformat()
```

```
        event['end'] = parse(message_dates[-1]).isoformat()
        event['durationEvent'] = True
        event['description'] = str(len(thread['message_ids'])) \
            + ' messages in thread'
        events.append(event) # acrescente o event da thread

if not os.path.isdir('out'):
    os.mkdir('out')

f = open(os.path.join('out', 'simile_data.json'), 'w')
f.write(json.dumps({'dateTimeFormat': 'iso8601', 'events':
events}, indent=4))
f.close()

print >> sys.stderr, 'Data file written to: %s' % f.name

# Aponta a SIMILE para o arquivo de dados
```

Há muitas demonstrações on-line da Timeline (*http://simile-widgets.org/timeline/*), além de ampla documentação. Este simples exemplo da representação de e-mails na Timeline mostra meramente o básico de suas funções e serve para que você dê seus primeiros passos. Trata-se apenas de uma amostra de tudo o que é possível. O tutorial "Getting Started with Timeline" (*http://www.simile-widgets.org/wiki/Getting_Started_with_Timeline*) é uma ótima introdução. Presumindo que você tenha os dados para realizar as consultas solicitadas, o script *mailboxes__participants_in_conversations_adapted_for_simile.py* traz todas as funcionalidades: primeiro faz o parsing dos dados, depois os coloca em um template HTML (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/web_code/simile/timeline.html*) e, por fim, abre as informações automaticamente em seu navegador. Aproveite!

Análise de seus próprios dados de emails

Os dados de e-mails da Enron servem como ótima ilustração em um capítulo sobre análise de e-mails, mas você certamente deve estar mais interessado em analisar seus próprios dados. Felizmente, muitos clientes populares de e-mail fornecem uma opção "export to mbox", fazendo com que seja simples obter seus dados em um formato adequado para análise, utilizando as técnicas descritas neste capítulo. Por exemplo, no Apple Mail, você pode selecionar determinado número de mensagens, escolher Save As... no menu File e, então, Raw Message Source na opção de formatação para

exportar as mensagens como um arquivo mbox (Figura 3.7). Pesquisando, é fácil descobrir como isso também pode ser feito na maioria dos outros principais clientes.

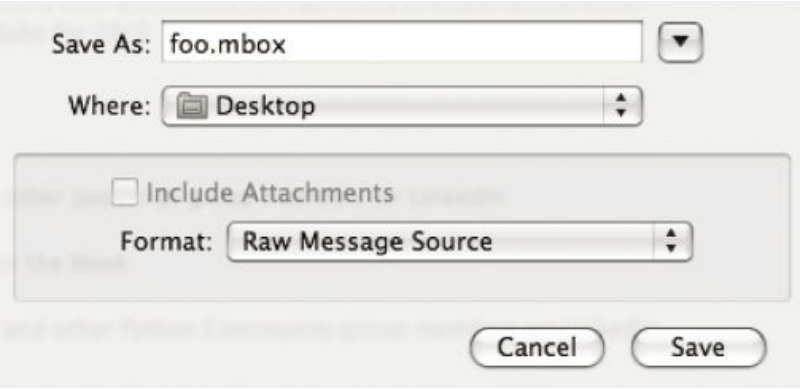


Figura 3.7 – Maioria dos clientes de e-mail fornece uma opção para exportar seus dados de e-mail para um arquivo mbox.

Se você utiliza exclusivamente um cliente on-line de email, pode optar por transferir seus dados para um cliente de e-mail e exportá-los, ou automatizar completamente a criação de um arquivo mbox, pegando os dados diretamente do servidor. Praticamente todos os serviços de correspondência on-line oferecem suporte ao POP3 (Post Office Protocol version 3). A maioria também aceita o IMAP (Internet Message Access Protocol), e scripts Python para acessar seus e-mails não são difíceis de formular. Uma ferramenta de linha de comando especialmente robusta que você pode utilizar para acessar dados de correspondência de qualquer fonte é o getmail (*http://pyropus.ca/software/getmail/*), escrito em Python. Dois módulos incluídos na biblioteca-padrão da Python, o `poplib` (*http://docs.python.org/library/poplib.html*) e o `imaplib` (*http://docs.python.org/library/imaplib.html*), fornecem bases muito sólidas, por isso é bem provável que você encontre muitos scripts úteis se fizer um pouco de pesquisa on-line. O getmail é muito fácil de utilizar. Para "engolir" os dados de sua caixa de entrada do Gmail, por exemplo, basta baixá-lo, instalá-lo, e então criar um arquivo *getmailrc* com algumas opções básicas. O exemplo 3.22 demonstra algumas configurações para um ambiente *nix. Usuários do Windows terão de alterar os valores de `[destination] path` e `[options] message_log` para paths válidos.

Exemplo 3.22 – Exemplo de configuração do getmail para um ambiente *nix

```
[retriever]
type = SimpleIMAPSSLRetriever
```

```
server = imap.gmail.com
username = ptwobrussell
password = blarty-blar-blar

[destination]
type = Mboxrd
path = /tmp/gmail.mbox

[options]
verbose = 2
message_log = ~/.getmail/gmail.log
```

Depois de configurado, simplesmente invocar `getmail` a partir de um terminal cuidará do restante. Assim que você tiver um mbox local disponível, poderá analisá-lo utilizando as técnicas que aprendeu neste capítulo:

```
$ getmail
getmail version 4.20.0
Copyright (C) 1998-2009 Charles Cazabon. Licensed under the GNU
GPL version 2.
SimpleIMAPSSLRetriever:ptwobrussell@imap.gmail.com:993:
  msg 1/10972 (4227 bytes) from ... delivered to Mboxrd
/tmp/gmail.mbox
  msg 2/10972 (3219 bytes) from ... delivered to Mboxrd
/tmp/gmail.mbox
  ...
```

A seção "Exploração de seus dados do Gmail", no capítulo 7, investiga o uso do `imaplib` para acessar seus dados do Gmail e analisá-los, como parte dos exercícios desse capítulo, concentrado em tecnologias do Google.

A extensão Gaph Your (Gmail) Inbox para o Chrome

Há muitos kits de ferramentas úteis que podem ser utilizados para analisar e-mails, e um dos mais promissores a surgir recentemente é a extensão do Chrome Graph Your Inbox (*http://www.graphyourinbox.com*). Para utilizar essa extensão, basta instalá-la, autorizar que acesse seus dados de e-mail, executar algumas consultas no Gmail e deixar que ela cuide do restante. Você pode pesquisar por palavras-chave como "pizza", valores cronológicos como "2010", ou executar consultas mais avançadas como "from:matthew@example.org" e "label:Strata". É altamente provável que essa extensão continue a evoluir, uma vez que se trata de uma criação

recente e muito bem recebida até agora. A figura 3.8 mostra uma tela de exemplo.

A seção "Exploração de seus dados do Gmail", no capítulo 7, fornece uma visão geral sobre o uso do módulo `smtplib` da Python para acessar sua conta do Gmail (ou qualquer outra conta de e-mail que se comunique em SMTP) e minerar as informações textuais das mensagens. Não se esqueça de verificá-la se você tem interesse em avançar para além das informações de cabeçalho e trabalhar com mineração de texto.

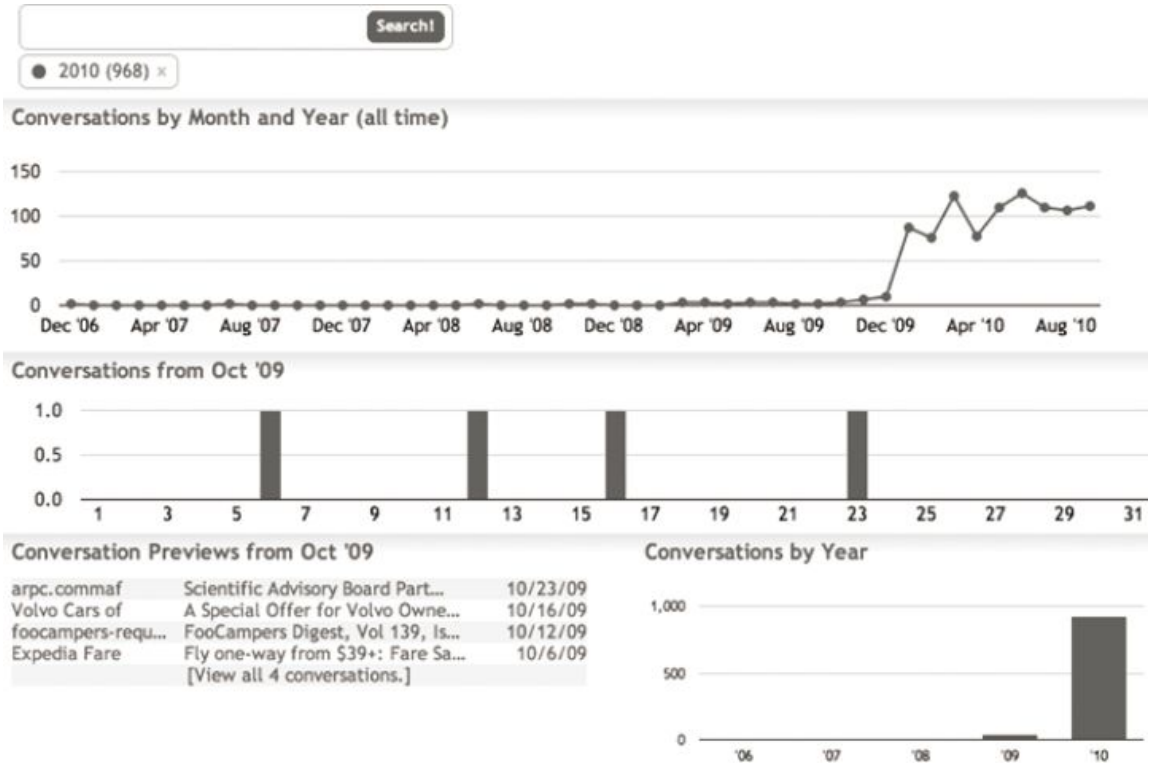


Figura 3.8 – A extensão Graph Your Inbox para o Chrome fornece um resumo conciso de sua atividade no Gmail.

Comentários finais

Abordamos *muitos* tópicos neste capítulo, mas mal arranhamos a superfície do que pode ser feito com dados de e-mails. Nosso foco esteve nos mboxes, um formato simples e conveniente, que permite extensa portabilidade e que pode ser analisado com facilidade por muitas ferramentas e pacotes Python. Há uma quantidade incrível de tecnologia de código aberto disponível para mineração de mboxes, e a Python é uma fantástica linguagem para analisá-los. Um pequeno investimento nessas ferramentas, como o CouchDB (perfeito para compartilhar com facilidade

seus dados), poderá render ótimos resultados.

---

1 De modo compreensível, ninguém teve (até agora) uma razão suficientemente boa para ser o primeiro a enfrentar processos judiciais multimilionários e descobrir se os termos de uso impostos por sites de mídia social podem ser aplicados. Com isso, o status quo (até agora) determina obediência às regras e que os usuários recuem quando comunicados acerca de possíveis violações.

2 N.T.: a Enron era uma companhia de energia norte-americana, com faturamento de US$101 bilhões em 2000, pouco antes do escândalo financeiro que ocasionou sua falência (fonte: Wikipédia).

3 Consulte *http://en.wikipedia.org/wiki/Quoted-printable* para obter uma visão geral, ou o RFC 2045 (*http://tools.ietf.org/html/rfc2045*), se estiver interessado nos detalhes práticos.

4 Usuários que trabalham com o CouchDB costumam empregar muito a palavra "relaxar". Nossa discussão acompanha essa característica, tentando ser o mais autêntica possível.

5 Como comparação, são necessários dois minutos para executar o exemplo 3.5 em um MacBook Pro 2.8GHz Core 2 Duo com o CouchDB 0.11.0 instalado utilizando as configurações-padrão.

6 Em termos gerais, ter uma característica de "alta concorrência" significa que várias operações podem ser executadas ao mesmo tempo (de modo concorrente), em múltiplos núcleos de um microprocessador, resultando em uma taxa de transferência máxima.

7 Em termos gerais, um gerador é uma função que retorna um iterador. Consulte *http://docs.python.org/tutorial/classes.html*.

8 N.T.: granularidade é a extensão à qual um sistema é dividido em partes pequenas, ou o sistema propriamente dito, ou sua descrição e observação (fonte: Wikipédia).

9 N.T.: a Era Unix teve início no dia 1° de janeiro de 1970. O nome se deve ao fato desta data ser o marco zero do sistema operacional Unix. O horário Unix, definido como o número de segundos passados desde o epoch, não considerando segundos bissextos, é largamente utilizado em sistemas operacionais do tipo Unix, bem como em outros sistemas (fonte: Wikipédia).

10 Na média, o quicksort realiza `n*lg(n)` comparações para ordenar uma coleção no pior caso, o que está perto do melhor desempenho possível sem conhecimento prévio que nos permita adequar o algoritmo aos dados.

11 No contexto da Enron, raptors eram dispositivos financeiros utilizados para ocultar prejuízos, empregados para manter milhões de dólares em débitos fora dos livros de contabilidade.

12 N.T.: uma thread, topic thread, ou ainda thread de discussão, é bastante comum na Internet, em especial em fóruns de discussão. É uma espécie de "linha de discussão" que, a partir de uma mensagem original, reúne respostas que vão sendo subordinadas umas às outras. O termo thread, em especial, é uma palavra em inglês que significa "linha"; nesse caso, thread significa "linha de discussão", e threading, a reunião de comunicações nessa forma (fonte: Wikipédia).

13 O “jwz” é uma referência ao seu autor, Jamie Zawinski.

14 Mais especificamente: fizemos algumas modificações para tornar o código um pouco mais orientado a objetos, alteramos os formatos de input/output para consumir nossos objetos de mensagens JSONificados e diminuímos drasticamente o perfil da memória, considerando somente os poucos campos necessários para o algoritmo de threading.

15 Fundamentalmente, é improvável que o CouchDB venha a oferecer suporte ao uso de vários núcleos para uma operação de escrita em massa, em razão da forma como escritas são serializadas para o disco. Entretanto, não é difícil imaginar um patch que aproveite a presença de múltiplos núcleos para certos tipos de operações de leitura, uma vez que a estrutura de dados subjacente é uma árvore, inerentemente adequada à travessia por vários nós.

16 N.T.: a marshalização é o processo que transforma a representação na memória de um objeto para um formato de dados, mais adequado ao armazenamento ou à transmissão. Ela é tipicamente utilizada quando dados têm de ser movidos entre partes diferentes de um programa de computador, ou de um programa para outro (fonte: Wikipédia).

CAPÍTULO 4

Twitter: amigos, seguidores e operações de conjuntos

A menos que você tenha passado os últimos anos criogenicamente congelado, certamente já ouviu falar do Twitter – serviço de microblogging que pode ser utilizado para transmitir pequenas atualizações de status (de no máximo 140 caracteres). Quer você ame, odeie, ou seja indiferente a ele, é inegável que o Twitter reformulou o modo como as pessoas se comunicam na web. Este capítulo tem a modesta intenção de apresentar algumas funções analíticas rudimentares que podem ser implementadas para aproveitar as APIs do Twitter e responder a algumas questões interessantes, como:

- Quantos amigos/seguidores eu tenho?
- Quem estou seguindo que não está me seguindo?
- Quem está me seguindo, mas eu não estou seguindo?
- Quem são as pessoas com o maior e o menor número de amigos em minha rede?
- Quem são meus "amigos mútuos" (pessoas que estou seguindo e que também me seguem)?
- Dados todos os meus seguidores e todos os seguidores dessas pessoas, qual minha influência potencial se fizerem um retweet de algo que escrevi?

A API do Twitter está em constante evolução. É altamente recomendado que você acompanhe sua conta, @TwitterAPI, e verifique diferenças entre o texto e o comportamento observado, comparando com a documentação oficial (*http://dev.twitter.com/doc*).

Este capítulo analisa relacionamentos entre usuários do Twitter, enquanto o próximo trata do conteúdo dos tweets em si. O código que desenvolveremos é relativamente robusto, no sentido em que considera problemas comuns como os famosos *rate limits* (*http://dev.twitter.com/pages/rate-limiting*)[1], erros de rede I/O, gerenciamento potencial de grandes volumes de dados etc. O resultado

final é um utilitário de linha de comando consideravelmente poderoso, que você deve ser capaz de adaptar facilmente à sua utilização personalizada (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/TwitterSocialGraphUtility.py*).

Ter ferramentas para coletar e minerar seus tweets é essencial. Entretanto, saiba que iniciativas que pretendem arquivar dados históricos do Twitter na Biblioteca do Congresso norte-americano (*http://www.loc.gov/tweet/how-tweet-it-is.html*) podem, em breve, fazer com que as inconveniências e dores de cabeça associadas à coleta e ao rate-limiting da API sejam algo do passado para muitos tipos de análise. Empresas como a Infochimps (*http://infochimps.org*) também estão surgindo e oferecendo meios para aquisição de diversos tipos de dados do Twitter (dentre outras opções). Uma consulta por dados do Twitter na Infochimps (*http://infochimps.org/search?query=twitter*) resulta em muitas informações, desde um arquivo dos tweets da hashtag #worldcup, até análises sobre o uso de smileys.

APIs RESTful e que empregam OAuth

O ano de 2010 será lembrado por alguns como o período de transição no qual o Twitter iniciou seu amadurecimento. A autenticação HTTP básica foi substituída pelo OAuth[2] (falaremos mais sobre isso em breve), a documentação melhorou, e os status das APIs se tornaram transparentes, dentre outras mudanças. APIs de busca do Twitter, introduzidas com a aquisição da Summize (*http://blog.twitter.com/2008/07/finding-perfect-match.html*), passaram a integrar a API REST (*http://en.wikipedia.org/wiki/Representational_State_Transfer*) "tradicional", enquanto as APIs de streaming se tornaram cada vez mais utilizadas em situações de produção. Se o Twitter estivesse em uma carta de vinhos, você poderia escolher a garrafa de 2010 e dizer: "Esse foi um ano bom – um ano muito bom." Muitas outras informações úteis podem ser encontradas on-line e este capítulo não pretende transmitir nada além do absolutamente necessário.

A maioria do desenvolvimento que mostraremos neste capítulo concentra-se nas APIs de grafo social, que nos permitem analisar amigos e seguidores de um usuário, na API para obtenção de informações estendidas de usuários (como nome, localização, último tweet etc.), acessadas a partir de uma lista, e na API para obtenção de dados dos tweets em si. Poderíamos escrever um livro inteiro (literalmente) que explorasse possibilidades adicionais, mas, assim que você entender como tudo funciona, sua imaginação não terá dificuldades em assumir o controle. Além disso, certos exercícios sempre devem ser deixados a cargo do "leitor interessado", não é mesmo?

O cliente Python que utilizaremos para o Twitter é chamado simplesmente de `twitter` (o mesmo que vimos em funcionamento no capítulo 1), e fornece um encapsulador básico aos web services RESTful do Twitter. Há pouca documentação sobre esse módulo, uma vez que solicitações são construídas da mesma forma que o URL é montado na documentação on-line do Twitter. Por exemplo, uma solicitação feita no terminal para recuperar as informações de usuário de Tim O'Reilly pode ser realizada simplesmente despachando uma solicitação ao recurso `/users/show` como um comando `curl`, da seguinte maneira:

```
$ curl 'http://api.twitter.com/1/users/show.json?
screen_name=timoreilly'
```

curl é uma ferramenta prática que pode ser utilizada para transferir dados de/para um servidor utilizando vários protocolos, além de ser especialmente útil na realização de solicitações HTTP a partir de um terminal. Sua inclusão é padrão e ela normalmente pode ser encontrada no PATH, na maioria dos sistemas *nix. Usuários do Windows talvez tenham de fazer seu download e configurá-la (*http://curl.haxx.se/latest.cgi?curl=win32-ssl-devel-msvc*).

Há algumas sutilezas nessa requisição. Primeiro, note que o Twitter tem uma API versionada, assim a presença de um `/1`, como contexto do URL, denota que é a Version 1 da API que está sendo utilizada. Além disso, um `user_id`, se disponível, poderia ter sido passado em vez de um `screen_name`. O mapeamento do comando `curl` ao script Python equivalente no exemplo 4.1 deve ser óbvio.

Exemplo 4.1 – Busca de informações estendidas sobre um usuário do Twitter

```
# -*- coding: utf-8 -*-
import twitter
import json
screen_name = 'timoreilly'
t = twitter.Twitter(domain='api.twitter.com', api_version='1')
response = t.users.show(screen_name=screen_name)
print json.dumps(response, sort_keys=True, indent=4)
```

Caso você ainda não tenha consultado a documentação on-line do Twitter (*http://dev.twitter.com/doc*), provavelmente vale a pena mencionar que a chamada à API `/users/show` não requer autenticação, e tem algumas peculiaridades específicas se um usuário tiver "protegido" seus tweets em suas configurações de privacidade. A chamada à API `/users/lookup` é muito semelhante a `/users/show`, exceto pelo fato de que requer autenticação e permite que você passe uma lista de valores `screen_name` e `user_id` separados por vírgulas, para realizar pesquisas em lotes. Para

obter autorização de uso da API do Twitter, você terá de saber mais sobre o OAuth, tópico da próxima seção.

Não, você não pode saber minha senha

OAuth significa "open authorization" (autorização aberta). Buscando se antecipar ao que o futuro nos reserva, esta seção oferece uma visão geral, ainda que superficial, do OAuth 2.0 (*http://tools.ietf.org/html/draft-ietf-oauth-v2-10*), um esquema de autorização em crescimento, já implementado sem ressalvas pelo Facebook (*http://developers.facebook.com/docs/authentication/*, veja no capítulo 9) e ao qual o Twitter planeja oferecer suporte "em breve" (*http://www.slideshare.net/episod/chirp-2010-too-many-secrets-but-never-enough-oauth-at-twitter*). Eventualmente, esse esquema se tornará o novo padrão do mercado. Quando da redação deste livro, o Twitter e muitos outros web services ofereciam suporte ao OAuth 1.0a, como definido pelo RFC 5849 (*http://tools.ietf.org/html/rfc5849*). Entretanto, esse panorama deve sofrer alterações, com o OAuth 2.0 sendo adotado para facilitar o trabalho de desenvolvedores e promovendo uma melhor experiência para o usuário (*http://hueniverse.com/2010/05/introducing-oauth-2-0/*). Ainda que a terminologia e os detalhes desta seção sejam específicos do OAuth 2.0, o fluxo de trabalho básico envolvido no OAuth 1.0a é muito semelhante. Ambos os esquemas têm "três pontas", no sentido em que envolvem uma troca de informações (muitas vezes chamada de "dança") entre uma aplicação cliente que necessita de acesso a um recurso protegido, o proprietário do recurso, como uma rede social, e um usuário final que tem de autorizar o acesso da aplicação cliente ao recurso protegido (sem fornecer uma combinação de nome de usuário/senha).

É simplesmente uma coincidência que a versão atual (e única) da API do Twitter seja a 1.0 e que seja oferecido suporte ao OAuth 1.0a. É provável que a versão 1 da API do Twitter eventualmente ofereça suporte ao OAuth 2.0 quando ele estiver pronto.

Assim, em resumo, o OAuth oferece uma forma por meio da qual você pode autorizar uma aplicação a acessar dados armazenados em outra aplicação, sem que tenha de compartilhar seu nome de usuário e senha. A especificação do IEFT para o OAuth 2.0 Protocol (*http://tools.ietf.org/html/draft-ietf-oauth-v2-10*) não é tão assustadora quanto parece, e analisá-la não deve consumir muito de seu tempo, uma vez que o OAuth está se tornando cada vez mais popular – especialmente

no mundo das redes sociais. A seguir, temos os passos essenciais envolvidos:

- Você (o *usuário final*) quer autorizar uma aplicação de algum tipo (o *cliente*) a acessar alguns de seus dados (um *escopo*), gerenciados por um web service (o *proprietário do recurso*).
  - Você é inteligente e sabe que não deve fornecer diretamente suas credenciais à aplicação.
- Em vez de pedir-lhe sua senha, o cliente redireciona você ao proprietário do recurso, e você autoriza um escopo para o cliente diretamente com o proprietário.
  - O cliente se identifica com um *identificador de cliente* único e uma forma de contatá-lo assim que a autorização tiver sido feita pelo usuário final.
- Presumindo que o usuário final autorize o cliente, o cliente é notificado e recebe um *código de autorização* confirmando que o usuário final o autorizou a acessar um escopo.
  - Entretanto, há um pequeno problema, se pararmos aqui: uma vez que o cliente se identificou com um identificador que não é necessariamente secreto, um cliente malicioso poderia se identificar de modo fraudulento e fingir ter sido criado por alguém de confiança, enganando o usuário final a autorizá-lo.
- O cliente apresenta ao proprietário do recurso o código de autorização que acabou de receber acompanhado do identificador e do correspondente *segredo do cliente*, e recebe de volta um *token de acesso*. A combinação do identificador do cliente, do segredo do cliente, e do código de autorização, garante que o proprietário do recurso pode identificar com segurança o cliente e sua autorização.
  - O token de acesso pode ser opcionalmente de curta duração, e ter de ser recarregado pelo cliente.
- O cliente utiliza o token de acesso para efetuar solicitações em nome do usuário final até que o token seja revogado ou expire.

A seção 1.4.1 da especificação (*http://tools.ietf.org/html/draft-ietf-oauth-v2-10#section-1.4 .1*) oferece mais detalhes e a base para o que vemos na figura 4.1.

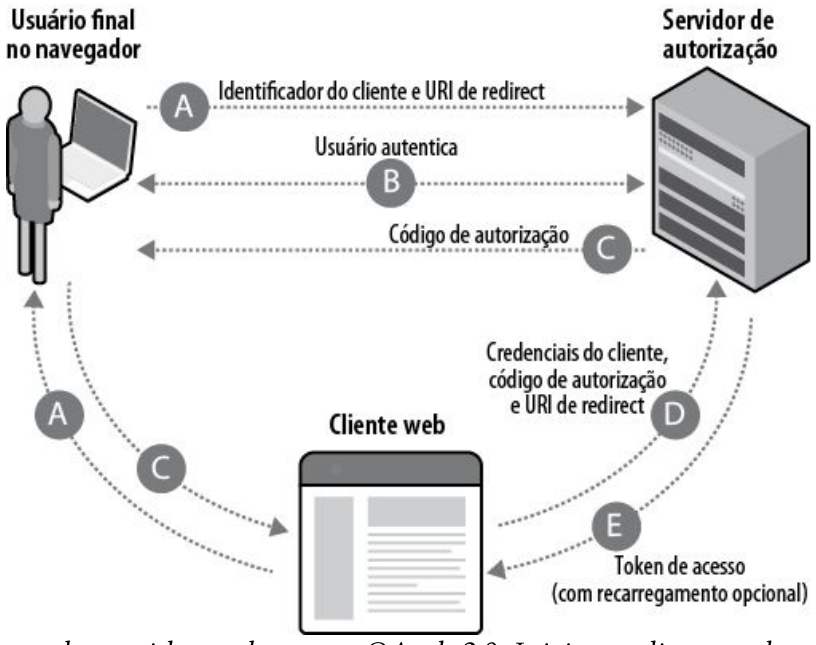


Figura 4.1 – Fluxo do servidor web para o OAuth 2.0. Inicie no cliente web e siga as setas em ordem lexicográfica, para voltar ao cliente depois de completar os passos A-E.

Repetimos: o OAuth 2.0 ainda é algo relativamente novo, e no final de 2010 muitos detalhes ainda estavam sendo implementados, sem que sua especificação fosse totalmente publicada como um RFC. A melhor opção para realmente compreendê-lo é ler a especificação, esboçar alguns fluxos, fazer algumas perguntas, desempenhar o papel de usuário malicioso etc. Para entender a política associada ao OAuth, consulte o artigo do Ars Technica, escrito por Ryan Paul, "Twitter: A Case Study on How to Do OAuth Wrong" (*http://arstechnica.com/security/guides/2010/09/twitter-a-case-study-on-how-to-do-oauth-wrong.ars*). Verifique também a resposta de Eran Hammer-Lahay, o primeiro autor do rascunho de trabalho do OAuth 2.0 Protocol, em *http://hueniverse.com/2010/09/all-this-twitter-oauth-security-nonsense/*, além de uma refutação bem detalhada, de autoria de Ben Adida, um especialista em criptografia (*http://benlog.com/articles/2010/09/02/an-unwarranted-bashing-of-twitters-oauth/*).

Como opção recente em situações que envolvem o que o Twitter chama de "caso de uso de usuário único", o Twitter também oferece uma forma mais rápida de obter as credenciais necessárias para solicitações, sem que você tenha de implementar o fluxo completo do OAuth. Leia mais sobre essa opção em *http://dev.twitter.com/pages/oauth_single_token*. Todavia, este capítulo e o seguinte presumem que você preferirá implementar o fluxo-padrão do OAuth.

Uma máquina poderosa de coleta de dados

A princípio, buscar dados do Twitter é muito simples: faça uma solicitação, armazene a resposta e repita, se necessário. Ainda assim, muitas situações práticas podem atrapalhar esse processo, como problemas com a rede I/O, a infame baleia do Twitter[3], e aqueles desagradáveis rate limits da API. Felizmente, não é tão difícil enfrentar esses problemas, desde que você faça um pouco de planejamento e se antecipe ao que pode dar (e certamente dará) errado.

Ao executar um programa de longa duração que está consumindo seu rate limit, é especialmente importante escrever um código que seja robusto; você deverá manipular todas as condições especiais que podem ocorrer, fazer seu melhor para remediar a situação e – caso isso simplesmente não seja suficiente – deixar indicações que permitam retornar futuramente ao problema. Em outras palavras, quando você escreve um código de coleta de dados para uma plataforma como o Twitter, deve presumir que terá algumas dificuldades. Existirão situações atípicas que terão de ser enfrentadas e, muitas vezes, elas serão mais regra do que exceção.

O código que desenvolveremos está preparado para lidar com os erros mais comuns e padronizado de modo que possa ser facilmente estendido para lidar com circunstâncias futuras. Dito isso, há dois erros HTTP específicos que você provavelmente encontrará ao coletar até mesmo quantidades modestas de dados do Twitter: o 401 Error (Not Authorized) e o 503 Error (Over Capacity). O primeiro ocorre quando você tenta acessar dados protegidos de um usuário, enquanto o segundo é basicamente imprevisível.

Sempre que o Twitter retornar um erro HTTP, o módulo `twitter` lançará uma exceção `TwitterHTTPError`, que pode ser manipulada como qualquer outra exceção Python, com um bloco `try/except`. O exemplo 4.2 ilustra um bloco mínimo de código que coleta IDs de amigos e manipula algumas das condições excepcionais mais comuns.

Você terá de criar uma app no Twitter (*http://twitter.com/apps/new*) para obter uma chave e um segredo que poderão ser utilizados com os exemplos deste livro. Isso é muito fácil, e também muito rápido.

Exemplo 4.2 – Uso do OAuth para autenticar e acessar alguns dados de amigos (friends_followers__get_friends.py)

```
# -*- coding: utf-8 -*-
```

```
import sys
import time
import cPickle
import twitter
from twitter.oauth_dance import oauth_dance

# Acesse http://twitter.com/apps/new para criar uma app e obter
estes itens

consumer_key = ''
consumer_secret = ''

SCREEN_NAME = sys.argv[1]
friends_limit = 10000

(oauth_token, oauth_token_secret) =
oauth_dance('MiningTheSocialWeb',
        consumer_key, consumer_secret)
t = twitter.Twitter(domain='api.twitter.com', api_version='1',
                    auth=twitter.oauth.OAuth(oauth_token,
oauth_token_secret,
                    consumer_key, consumer_secret))

ids = []
wait_period = 2 # segundos
cursor = -1

while cursor != 0:
    if wait_period > 3600: # 1 hora
        print 'Too many retries. Saving partial data to disk and
exiting'
        f = file('%s.friend_ids' % str(cursor), 'wb')
        cPickle.dump(ids, f)
        f.close()
        exit()

    try:
        response = t.friends.ids(screen_name=SCREEN_NAME,
cursor=cursor)
        ids.extend(response['ids'])
        wait_period = 2
    except twitter.api.TwitterHTTPError, e:
       if e.e.code == 401:
            print 'Encountered 401 Error (Not Authorized)'
            print 'User %s is protecting their tweets' %
(SCREEN_NAME, )
      elif e.e.code in (502, 503):
          print 'Encountered %i Error. Trying again in %i
```

```
seconds' % (e.e.code, wait_period)
            time.sleep(wait_period)
            wait_period *= 1.5
            continue
        elif t.account.rate_limit_status()['remaining_hits'] == 0:
            status = t.account.rate_limit_status()
            now = time.time() # UTC
            when_rate_limit_resets =
status['reset_time_in_seconds'] # UTC
            sleep_time = when_rate_limit_resets - now
            print 'Rate limit reached. Trying again in %i seconds'
% (sleep_time, )
            time.sleep(sleep_time)
            continue
    cursor = response['next_cursor']
    print 'Fetched %i ids for %s' % (len(ids), SCREEN_NAME)
    if len(ids) >= friends_limit:
        break
# faz algo interessante com os IDs
print ids
```

O módulo `twitter.oauth` fornece duas funções de conveniência, `read_token_file` e `write_token_file`, que podem ser utilizadas para armazenar e recuperar seu token OAuth e o segredo dele, de modo que você não tenha de digitar um PIN para autenticação todas as vezes.

No linguajar do OAuth 2.0, um "cliente" descreve a mesma função de um "consumidor" no OAuth 1.0. Por isso, utilizamos os nomes de variáveis `consumer_key` e `consumer_secret` no exemplo anterior.

Há vários itens que merecem destaque nessa listagem:

- Você pode obter seus próprios `consumer_key` e `consumer_secret` registrando uma aplicação com o Twitter em *http://dev.twitter.com/apps/new*. Esses dois itens, acompanhados das credenciais retornadas por meio da "dança" do OAuth, permitem que você forneça acesso aos dados de sua conta (sua lista de amigos, neste exemplo específico) para uma aplicação.
- A documentação on-line para as APIs de grafo social do Twitter (*http://dev.twitter.com/doc/get/friends/ids*) afirma que solicitações de dados de amigos/seguidores retornarão até 5.000 IDs por chamada. Caso haja mais de 5.000 IDs a serem retornados, um valor de cursor não igual a zero será retornado, por meio do qual você poderá navegar

até o lote seguinte. Este exemplo específico determina um máximo de 10.000 valores de ID, mas `friends_limit` poderia ser um número arbitrariamente maior.

- Dado que o recurso `/friends/ids` retorna até 5.000 IDs por vez, contas de usuário regulares podem recuperar até 1.750.000 IDs antes de atingir o rate limit, com base na métrica de 350 solicitações/hora. Mesmo que um usuário assim, com tantos amigos no Twitter, possa ser uma anomalia, não é de todo incomum que usuários populares tenham números como esse de seguidores.
- Não fica claro na documentação oficial, ou no código do exemplo em si, mas os valores de ID nos resultados parecem estar em ordem cronológica inversa, de forma que o primeiro valor representa a pessoa mais recente que você seguiu, e o último valor, a primeira. Solicitações de seguidores via `t.followers.ids` parecem retornar resultados na mesma ordem.

Até aqui, você foi apresentado somente a algumas das APIs do Twitter, o suficiente para responder a muitas questões interessantes sobre sua conta ou qualquer conta não protegida, mas há inúmeras outras APIs disponíveis. Em breve, veremos algumas delas, mas primeiro faremos um curto intervalo para refatorar o exemplo 4.2.

Um breve interlúdio de refatoração

Como praticamente todos os trechos de código interessantes envolvendo dados do Twitter realizarão repetidamente a "dança" do OAuth, fazendo solicitações robustas e enfrentando a enorme quantidade de erros que podem ocorrer, vale a pena estabelecer um padrão para realizá-las. A abordagem que empregaremos é isolar a lógica do OAuth para funções `login()` e `makeTwitterRequest`, de forma que o exemplo 4.2 fique como a versão refatorada do exemplo 4.3:

Exemplo 4.3 – Exemplo 4.2 refatorado de modo a utilizar utilitários usuais para OAuth e para a realização de solicitações à API (friends_followers___get_friends_refactored.py)

```
# -*- coding: utf-8 -*-
import sys
import time
import cPickle
import twitter
```

```
from twitter__login import login
from twitter__util import makeTwitterRequest

friends_limit = 10000

# Talvez você tenha de ajustar suas configurações do OAuth em
twitter__login.py
t = login()

def getFriendIds(screen_name=None, user_id=None,
friends_limit=10000):
    assert screen_name is not None or user_id is not None
    ids = []
    cursor = -1
    while cursor != 0:
        params = dict(cursor=cursor)
        if screen_name is not None:
            params['screen_name'] = screen_name
        else:
            params['user_id'] = user_id

        response = makeTwitterRequest(t, t.friends.ids, **params)

        ids.extend(response['ids'])
        cursor = response['next_cursor']
        print >> sys.stderr, \
            'Fetched %i ids for %s' % (len(ids), screen_name or
user_id)
        if len(ids) >= friends_limit:
            break

    return ids

if __name__ == '__main__':
    ids = getFriendIds(sys.argv[1], friends_limit=10000)
    # faz algo interessante com os ids
    print ids
```

Daqui em diante, continuaremos a utilizar `twitter__login` (*http://github.com/ptwobrussell/Mining-the-Social-eb/blob/master/python_code/twitter__login.py*) e `twitter__util` (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/twitter__util.py*) para manter os exemplos tão claros e simples quanto possível. Vale a pena fazer uma pausa e consultar o código-fonte desses módulos on-line antes de avançar. Eles serão utilizados repetidas vezes, e em breve adicionaremos muitas funções de conveniência a `twitter_util`.

A próxima seção apresenta o Redis, um poderoso servidor de estruturas de dados que rapidamente alcançou um grande número de seguidores em razão de seu desempenho e de sua simplicidade.

Redis: um servidor de estruturas de dados

Como já observamos, é muito importante planejar antecipadamente quando você pretende executar um programa de longa duração para acessar dados da web, uma vez que muitos procedimentos podem dar errado. Mas o que fazer com todos os dados depois de obtê-los? Inicialmente, você pode pensar em simplesmente salvá-los no disco. Na situação que acabamos de analisar, isso poderia resultar em uma estrutura de diretórios semelhante à seguinte:

```
./
screen_name1/
    friend_ids.json
    follower_ids.json
    user_info.json
screen_name2/
    ...
...
```

Esse resultado parece razoável até que você precise coletar todos os amigos/seguidores de um usuário muito popular – nesse caso, dependendo de sua plataforma, você poderia terminar com um diretório com milhões de subdiretórios, praticamente inútil, pois você não teria como analisá-lo com facilidade em um terminal. Salvar todas essas informações no disco também pode exigir que você mantenha um registro de algum tipo para monitorar todos os nomes de tela, dado que o tempo necessário para gerar uma listagem de diretório (caso necessária) para milhões de arquivos pode não representar um perfil de desempenho desejado. Caso a aplicação que utiliza os dados seja dividida em threads, você pode terminar com múltiplas operações de escrita que terão de acessar o mesmo arquivo ao mesmo tempo, por isso terá de lidar com operações como bloqueio de arquivos e coisas do tipo, o que provavelmente não é algo que queira fazer. Tudo de que realmente necessitamos, neste caso, é de um sistema que facilite o armazenamento de pares básicos de chave/valor e de um simples esquema de codificação de chaves – algo como um dicionário no disco seria um bom ponto de

partida. O próximo trecho de código demonstra a construção de uma chave, concatenando um ID de usuário, um delimitador e um nome de estrutura de dados:

```
s = {}
s["screen_name1$friend_ids"] = [1,2,3, ...]
s["screen_name1$friend_ids"] # retorna [1,2,3, ...]
```

Não seria interessante se o mapa pudesse computar automaticamente operações de conjuntos, de modo que pudéssemos simplesmente dizer-lhe para fazer algo como:

```
s.intersection("screen_name1$friend_ids",
"screen_name1$follower_ids")
```

computando automaticamente "amigos mútuos" de um usuário do Twitter (por exemplo, para descobrir qual de seus amigos também o estão seguindo)? Bem, há um projeto de código aberto, Redis (*http://code.google.com/p/redis/*), que fornece *exatamente* esse tipo de capacidade. Ele é fácil de instalar, incrivelmente rápido (escrito em C), tem boa escalabilidade, suporte ativo, e um ótimo cliente Python que acompanha documentação disponível. Para experimentá-lo, basta instalá-lo (*http://code.google.com/p/redis/*) e iniciar o servidor. Usuários do Windows podem facilitar esse processo fazendo o download de um binário (*http://code.google.com/p/servicestack/wiki/RedisWindowsDownload*), mantido pelo servicestack.net. Depois, simplesmente execute `easy_install redis` para obter um cliente Python que fornece acesso prático a tudo que o Redis tem a oferecer. Por exemplo, o trecho anterior de código pode ser traduzido no código Redis a seguir:

```
import redis
r = redis.Redis(host='localhost', port=6379, db=0) # Parâmetros
default
[ r.sadd("screen_name1$friend_ids", i) for i in [1, 2, 3, ...] ]
r.smembers("screen_name1$friend_ids") # Retorna [1, 2, 3, ...]
```

Note que ainda que `sadd` e `smembers` sejam operações baseadas em conjuntos, o Redis inclui operações específicas para muitos outros tipos de estruturas de dados, como conjuntos, listas e hashes. As operações de conjuntos são de interesse especial para nós, pois fornecem respostas para muitas das questões apresentadas no início deste capítulo. Vale a pena fazer uma pausa para revisar a documentação do cliente Python do Redis

e compreender melhor tudo que pode ser feito. Lembre-se de que você pode simplesmente executar um comando como `pydoc redis.Redis` para consultar rapidamente a documentação, a partir de um terminal.

Consulte "Redis: under the hood" (*http://www.pauladamsmith.com/articles/redis_under_the_hood.html*) para uma fantástica explanação técnica sobre como o Redis funciona internamente.

Operações elementares de conjuntos

As operações mais usuais de conjuntos que você provavelmente encontrará são as de união, intersecção e diferença. Lembre-se de que a diferença entre um conjunto e uma lista é que um conjunto é não-ordenado e contém somente membros únicos, enquanto uma lista é ordenada e pode conter membros duplicados. Em sua versão 2.6, a Python oferece suporte integrado para conjuntos por meio da estrutura de dados `set`. A tabela 4.1 ilustra alguns exemplos de operações usuais de conjuntos para um universo pequeno de discurso, envolvendo amigos e seguidores:

```
Friends = {Abe, Bob}, Followers = {Bob, Carol}
```

Tabela 4.1 – Exemplo de operações de conjuntos para Friends (amigos) and Followers (seguidores)

Operação	Resultado	Comentário
Friends U Followers	Abe, Bob, Carol	A rede total de alguém
Friends ∩ Followers	Bob	Os amigos mútuos de alguém
Friends – Followers	Abe	Pessoas que o usuário está seguindo, mas que não o estão seguindo
Followers – Friends	Carol	Pessoas que estão seguindo o usuário, mas que ele não está seguindo

Como mencionamos antes, o Redis fornece operações nativas para computação de operações usuais de conjuntos. Algumas das mais relevantes para o trabalho que faremos incluem:

`smembers`
: Retorna todos os membros de um conjunto.

`scard`
: Retorna a cardinalidade de um conjunto (o número de membros no conjunto).

`sinter`
: Computa a intersecção para uma lista de conjuntos.

`sdiff`
: Computa a diferença para uma lista de conjuntos.

`mget`
: Retorna uma lista de valores em strings para uma lista de chaves.

`mset`
: Armazena uma lista de valores em strings em uma lista de chaves.

`sadd`
: Adiciona um item a um conjunto (criando o conjunto, caso ele ainda não exista).

`keys`
: Retorna uma lista de chaves correspondendo a um padrão de tipo regex.

Ao analisar rapidamente o pydoc para os tipos de dados de conjuntos da Python, você deve perceber o mapeamento que há entre ele e as APIs do Redis.

Análise de conjuntos como nos velhos tempos

Ainda que os conceitos envolvidos na teoria dos conjuntos sejam muito mais antigos, é Georg Cantor quem recebe o crédito por elaborá-la. Seu artigo, "On a Characteristic Property of All Real Algebraic Numbers," escrito em 1874, formalizou a teoria dos conjuntos como parte de seu esforço para responder questões relacionadas ao conceito do infinito. Por exemplo:

- Existem mais números naturais (zero e inteiros positivos) do que inteiros (números positivos e negativos)?
- Existem mais números racionais (números que podem ser expressos como frações) do que inteiros?
- Existem mais números irracionais (números que não pode ser expressos como frações, como pi, √2 etc.) do que números racionais?

O ponto essencial do trabalho de Cantor sobre o infinito, no que se refere às duas primeiras questões, é que as cardinalidades dos conjuntos de números naturais, inteiros e racionais são iguais, uma vez que você pode mapear esses números de modo a formar uma sequência com um ponto definido de início, mas que se estende infinitamente em uma direção. Mesmo que nunca haja um ponto final, as cardinalidades desses conjuntos são *infinitos contáveis*, pois há uma sequência definida que poderia ser seguida de modo determinístico, se tivéssemos tempo para contá-la. A cardinalidade de um conjunto infinito contável tornou-se conhecida dos matemáticos como $\aleph_0$, uma definição oficial do infinito. Considere os seguintes padrões numéricos que transmitem a noção básica por trás dos conjuntos infinitos contáveis. Cada padrão mostra um ponto de início que pode se estender infinitamente:

- Números naturais: 0, 1, 2, 3, 4...
- Inteiros positivos: 1, 2, 3, 4, 5...

- Inteiros negativos: -1, -2, -3, -4, -5...
- Inteiros: 0, 1, -1, 2, -2, 3, -3, 4, -4...
- Números racionais: 0/0, 0/1, -1/1, -1/0, -1/-1, 0/-1, 1/-1, 1/0, 1/1...

Note que o padrão para os números racionais mostra que você pode iniciar na origem do plano Cartesiano e construir uma espiral na qual cada par de coordenadas x/y seja expresso como uma fração, que é um número racional (os dois casos em que a razão é indefinida, por causa da divisão por zero, não têm consequência quanto à cardinalidade do conjunto como um todo).

Conforme avança, entretanto, a cardinalidade do conjunto dos números irracionais não é igual a $\aleph_0$,, pois é impossível ordená-los de forma que sejam contáveis e que tenham uma correspondência de um-para-um com um conjunto de cardinalidade $\aleph_0$,. Cantor utilizou como prova o que ficou conhecido como o famoso *argumento de diagonalização (http://en.wikipedia.org/wiki/Cantor's_diagonal_argument)*.

De acordo com esse argumento, quando você acredita ter mapeado uma sequência que torna contáveis os números irracionais, pode ser mostrado que uma grande quantidade de números está faltando. De fato, você nunca pode formar a correspondência um-para-um necessária. Assim, a cardinalidade do conjunto de números irracionais não é idêntica às cardinalidades dos conjuntos de números naturais, inteiros positivos, e números irracionais, uma vez que uma correspondência de um-para-um a partir de um desses conjuntos não pode ser derivada.

Então, qual seria a cardinalidade do conjunto de números irracionais? Pode ser mostrado que o conjunto potência do conjunto de cardinalidade $\aleph_0$, é a cardinalidade do conjunto de todos os números irracionais. Esse valor é conhecido como $\aleph_1$. Além disso, o conjunto potência do conjunto de cardinalidade $\aleph_1$ é conhecido como $\aleph_2$ etc. Computar o conjunto potência de um conjunto de números infinitos é admitidamente um conceito complexo, mas vale a pena ponderá-lo quando você perder o sono à noite.

Adição de métricas básicas para amigos/seguidores

O Redis deve certamente lhe ajudar em sua busca por eficiência no processamento e análise de grandes quantidades de dados do Twitter para certos tipos de consultas. Adaptar o exemplo 4.2 com alguma lógica adicional capaz de armazenar dados no Redis requer apenas uma pequena alteração. O exemplo 4.4 é uma atualização que computa algumas estatísticas básicas de amigos/seguidores. Funções nativas no Redis são utilizadas para computar as operações de conjuntos.

Exemplo 4.4 – Coleta, armazenamento, e computação de estatísticas sobre amigos e seguidores (friends_followers__friend_follower_symmetry.py)

```
# -*- coding: utf-8 -*-
import sys
import locale
import time
import functools
import twitter
import redis
from twitter__login import login
```

```
# Uma função de tipo template para maximizar a reutilização do
código,
# essencialmente um encapsulador em torno de makeTwitterRequest
# com alguma lógica adicional para interface com o Redis
from twitter__util import _getFriendsOrFollowersUsingFunc

# Cria um valor de chave consistente para um usuário, dado um
screen name
from twitter__util import getRedisIdByScreenName

SCREEN_NAME = sys.argv[1]

MAXINT = sys.maxint

# Para uma formatação adequada dos números
locale.setlocale(locale.LC_ALL, '')

# Talvez você tenha de ajustar suas configurações OAuth em
twitter__login.py

t = login()

# Conecta-se utilizando as configurações default para localhost
r = redis.Redis()

# Alguns encapsuladores em torno de
_getFriendsOrFollowersUsingFunc
# que vinculam os dois primeiros argumentos
getFriends = functools.partial(_getFriendsOrFollowersUsingFunc,
                               t.friends.ids, 'friend_ids', t, r)
getFollowers = functools.partial(_getFriendsOrFollowersUsingFunc,
                                 t.followers.ids, 'follower_ids',
t, r)

screen_name = SCREEN_NAME

# pega os dados

print >> sys.stderr, 'Getting friends for %s...' % (screen_name,
)
getFriends(screen_name, limit=MAXINT)

print >> sys.stderr, 'Getting followers for %s...' %
(screen_name, )
getFollowers(screen_name, limit=MAXINT)

# utilize o redis para computar os números

n_friends = r.scard(getRedisIdByScreenName(screen_name,
'friend_ids'))

n_followers = r.scard(getRedisIdByScreenName(screen_name,
'follower_ids'))

n_friends_diff_followers = r.sdiffstore('temp',
```

```
                                          [getRedisIdByScreenName(s
creen_name,
                                          'friend_ids'),
                                          getRedisIdByScreenName(sc
reen_name,
                                         'follower_ids')])
r.delete('temp')
n_followers_diff_friends = r.sdiffstore('temp',
                                          [getRedisIdByScreenName(s
creen_name,
                                          'follower_ids'),
                                          getRedisIdByScreenName(sc
reen_name,
                                          'friend_ids')])
r.delete('temp')
n_friends_inter_followers = r.sinterstore('temp',
        [getRedisIdByScreenName(screen_name, 'follower_ids'),
        getRedisIdByScreenName(screen_name, 'friend_ids')])
r.delete('temp')
print '%s is following %s' % (screen_name, locale.format('%d',
n_friends, True))
print '%s is being followed by %s' % (screen_name,
locale.format('%d', n_followers, True))
print '%s of %s are not following %s back' % (locale.format('%d',
        n_friends_diff_followers, True), locale.format('%d',
n_friends, True), screen_name)
print '%s of %s are not being followed back by %s' %
(locale.format('%d',
        n_followers_diff_friends, True), locale.format('%d',
n_followers, True), screen_name)
print '%s has %s mutual friends' \
    % (screen_name, locale.format('%d',
n_friends_inter_followers, True))
```

Fora o uso de `functools.partial` (*http://docs.python.org/library/functools.html*), para criar `getFriends` e `getFollowers` a partir de um trecho regular de código parametrizado, entender o exemplo 4.4 não deve ser difícil. Todavia, há outro ponto sutil que merece destaque: não há uma chamada a `r.save` no exemplo 4.4, o que significa que são as configurações em *redis.conf* que ditam quando os dados são persistidos no disco. Por padrão, o Redis armazena dados na memória e salva os dados em disco de modo assíncrono, de acordo com

uma programação determinada pelo número de alterações feitas dentro de um intervalo específico de tempo. O risco com escritas assíncronas é que você pode perder dados, caso certas condições não esperadas, como falha do sistema, ou falta de energia, ocorram. O Redis fornece uma opção "append only" (apenas acréscimo) que você pode habilitar no *redis.conf* para se proteger contra essas eventualidades.

É altamente recomendável que você habilite a opção `appendonly` no `redis.conf` para se proteger contra perdas de dados. Consulte o "Append Only File HOWTO" (*http://code.google.com/p/redis/wiki/AppendOnlyFileHowto*) para maiores detalhes.

Considere a seguinte saída, relacionada à rede de seguidores de Tim O'Reilly. Lembre-se de que há um rate limit de 350 solicitações OAuth por hora. Assim, você pode esperar que este código demore pouco menos de uma hora para ser executado, uma vez que aproximadamente 300 chamadas à API terão de ser feitas para coletar todos os valores de ID de seguidor:

```
timoreilly is following 663
timoreilly is being followed by 1,423,704
131 of 633 are not following timoreilly back
1,423,172 of 1,423,704 are not being followed back by timoreilly
timoreilly has 532 mutual friends
```

Note que ainda que você possa coletar um número menor de seguidores, evitando a espera imposta pelo rate limit, a documentação da API não afirma que coletar as primeiras *N* páginas de dados resultará em uma amostragem randômica, e parece que os dados são retornados em ordem cronológica inversa – assim, talvez você não possa extrapolar conclusões de forma previsível se sua lógica depender disso. Por exemplo, se os primeiros 10.000 seguidores retornados contiverem 532 amigos mútuos, uma extrapolação a partir desses dados resultaria em uma análise distorcida, pois os resultados não seriam representativos da população geral. Para um usuário muito popular, como a Britney Spears, que tem mais de 5.000.000 de seguidores, algo em torno de 1.000 chamadas à API seriam necessárias para buscar todos os seguidores em um período de aproximadamente quatro horas. Mesmo assim, a espera provavelmente vale a pena para esse tipo de dado, e você poderia utilizar as APIs de streaming do Twitter para manter seus dados atualizados de forma a nunca ter de realizar essa tarefa novamente.

Uma causa comum de erros em alguns tipos de análise é esquecer o tamanho geral de uma população, quando comparado ao da amostra. Por exemplo, uma amostragem randômica de 10.000 dos amigos e

> seguidores de Tim O'Reilly seria o suficiente para conter a população total de seus amigos, mas apenas uma pequena fração de seus seguidores. Dependendo da sofisticação de sua análise, o tamanho da amostra (*http://en.wikipedia.org/wiki/Sampling_(statistics)*) relativo ao tamanho geral da população pode influenciar na determinação da significância estatística do resultado de um experimento, e do nível de confiança que você pode ter quanto a esses dados.

Analisando até essas simples estatísticas de amigos/seguidores, algumas questões interessantes surgem naturalmente. Por exemplo, quem são as 131 pessoas que Tim O'Reilly segue, e que não o estão seguindo? Dadas as várias possibilidades que podem ser consideradas sobre amigos e seguidores, a questão "Quem eu sigo que não está me seguindo?" é uma das mais interessantes e pode fornecer muitos insights sobre os interesses de uma pessoa. Assim, como podemos responder essa questão?

Analisar uma lista de IDs de usuário não é muito empolgante, justamente por isso transformá-los em objetos de usuário deve ser nosso primeiro passo. O exemplo 4.5 estende o exemplo 4.4 encapsulando código de tratamento ou manipulação de erros de uma forma reutilizável, além de fornecer uma função que demonstra como podemos transformar esses valores de IDs em nomes de tela utilizando a API `/users/lookup`, que aceita uma lista de até 100 IDs de usuário ou nomes de tela e retorna a mesma informação básica de usuário vista antes com `/users/show`.

Exemplo 4.5 – Transformação de informações básicas de usuários, como nomes de tela, a partir de seus IDs (friends_followers__get_user_info.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import redis
from twitter__login import login

# Uma chamada makeTwitterRequest passada ao recurso
/users/lookup,
# que aceita uma lista separada por vírgulas de até
# 100 nomes de tela. Os detalhes não são muito interessantes.
# Consulte também http://dev.twitter.com/doc/get/users/lookup
from twitter__util import getUserInfo

if __name__ == "__main__":
    screen_names = sys.argv[1:]

    t = login()
    r = redis.Redis()

    print json.dumps(
        getUserInfo(t, r, screen_names=screen_names),
```

```
          indent=4
        )
```

Ainda que não reproduzida inteiramente, a função `getUserInfo`, importada de `twitter_util`, é essencialmente apenas uma `makeTwitterRequest` ao recurso `/users/lookup`, utilizando uma lista de nomes de tela. O trecho seguinte demonstra:

```
def getUserInfo(t, r, screen_names):
    info = []
    response = makeTwitterRequest(t,
                                  t.users.lookup,
                                  screen_name=','.join(screen_nam
es)
                                 )
    for user_info in response:
        r.set(getRedisIdByScreenName(user_info['screen_name'],
'info.json'),
              json.dumps(user_info))
        r.set(getRedisIdByUserId(user_info['id'], 'info.json'),
              json.dumps(user_info))
        info.extend(response)
return info
```

Vale notar que `getUserInfo` armazena a mesma informação de usuário sob duas chaves diferentes: o ID do usuário e o nome de tela. Armazenar ambas essas chaves nos permite consultar com facilidade um nome de tela a partir de um valor de ID de usuário e um valor de ID de usuário a partir de um nome de tela. Traduzir um valor de ID de usuário para um nome de tela é uma operação particularmente útil, pois as APIs de grafo social utilizadas para obter dados de amigos e seguidores retornam somente valores de ID, destituídos de valor intuitivo até que convertidos em nomes de tela, ou em outras informações básicas de usuário. Ainda que haja armazenamento redundante neste esquema, comparado a outras abordagens, sua conveniência vale a pena. Sinta-se livre para adotar uma abordagem mais leve, caso sua capacidade de armazenamento seja um fator determinante.

Um exemplo de objeto de informação de usuário para Tim O'Reilly pode ser visto no exemplo 4.5, ilustrando o tipo de informação disponível sobre usuários do Twitter. Não há limite quanto ao que pode ser feito com dados tão ricos. Não mineraremos as descrições de usuário, nem os

tweets, das pessoas que Tim segue, mas que não o seguem; ainda assim, você deve ter dados suficientes caso queira conduzir esse tipo de análise.

Exemplo 4.6 – Exemplo de objeto de usuário representado como dados JSON para Tim O'Reilly


```
{
    "id": 2384071,
    "verified": true,
    "profile_sidebar_fill_color": "e0ff92",
    "profile_text_color": "000000",
    "followers_count": 1423326,
    "protected": false,
    "location": "Sebastopol, CA",
    "profile_background_color": "9ae4e8",
    "status": {
        "favorited": false,
        "contributors": null,
        "truncated": false,
        "text": "AWESOME!! RT @adafruit: a little girl asks after
seeing adafruit ...",
        "created_at": "Sun May 30 00:56:33 +0000 2010",
        "coordinates": null,
        "source": "<a href=\"http://www.seesmic.com/\"
rel=\"nofollow\">Seesmic</a>",
        "in_reply_to_status_id": null,
        "in_reply_to_screen_name": null,
        "in_reply_to_user_id": null,
        "place": null,
        "geo": null,
        "id": 15008936780
    },
    "utc_offset": -28800,
    "statuses_count": 11220,
    "description": "Founder and CEO, O'Reilly Media. Watching the
alpha geeks...",
    "friends_count": 662,
    "profile_link_color": "0000ff",
    "profile_image_url":
"http://a1.twimg.com/profile_images/941827802/IMG_...jpg",
    "notifications": false,
    "geo_enabled": true,
    "profile_background_image_url":
"http://a1.twimg.com/profile_background_...gif",
    "name": "Tim O'Reilly",
```

```
    "lang": "en",
    "profile_background_tile": false,
    "favourites_count": 10,
    "screen_name": "timoreilly",
    "url": "http://radar.oreilly.com",
    "created_at": "Tue Mar 27 01:14:05 +0000 2007",
    "contributors_enabled": false,
    "time_zone": "Pacific Time (US & Canada)",
    "profile_sidebar_border_color": "87bc44",
    "following": false
}
```

A lógica refatorada para manipular erros HTTP e obter informações de usuário em lotes é fornecida nas seções seguintes. Note que a função `handleTwitterHTTPError` deixa intencionalmente de incluir tratamento de erros para todos os casos concebíveis, uma vez que a ação pretendida pode variar de situação para situação. Por exemplo, caso haja um erro `urllib2.URLError` (time out da operação), disparado se alguém desconectar seu cabo de rede, você pode solicitar ao usuário um curso específico de ação.

O exemplo 4.5 chama nossa atenção por algumas boas notícias e outras não tão boas assim. A boa notícia é que a tradução dos IDs de usuário para objetos de usuário que contenham *byline* (linha em que consta o nome do autor), dados de localização, último tweet etc. oferece um vasto recurso de informações. A notícia não tão boa assim é que é bem custoso fazê-lo, em termos de rate limiting, uma vez que você pode obter os dados somente em lotes de 100. Para os amigos de Tim O'Reilly, isso significaria apenas sete chamadas à API. Entretanto, para seus seguidores, seriam necessárias mais de 14.000, informação que demoraria praticamente dois dias para ser coletada, considerando o rate limit de 350 chamadas por hora (e nenhum problema durante a coleta).

Ainda assim, dado um conjunto completo dos valores de ID dos amigos e seguidores de alguém, você pode randomicamente retirar amostras e calcular medidas de significância estatística como preferir. O Redis fornece a função `srandmember`, perfeita para esse caso. Você passa a ela o nome de um conjunto, como `timoreilly$follower_ids`, e ela retorna um membro randômico dele.

Cálculo de similaridade computando amigos e seguidores em comum

Outra ação a nosso alcance é a computação dos amigos e seguidores que dois ou mais usuários do Twitter têm em comum. Dado um universo, saber sobre essas pessoas pode ser interessante por muitos motivos. Um deles seria por representarem uma "ligação comum" que conecta redes distintas; você poderia interpretar esses dados como um tipo de métrica de similaridade. Por exemplo, caso dois usuários estejam seguindo um grande número de mesmas pessoas, você poderia concluir que eles têm interesses muito semelhantes. Partindo dessa constatação, seria possível analisar a informação dos tweets dos amigos em comum para obter um maior insight quanto ao que essas pessoas têm em comum, ou chegar a outras conclusões. Basta uma simples operação de conjuntos para computar amigos e seguidores em comum. O exemplo 4.7 ilustra o uso da função `sinterstore` do Redis, que armazena o resultado de uma intersecção de conjuntos, e apresenta o `locale.format`, utilizado para imprimir o resultado de uma forma que facilite sua leitura.

Exemplo 4.7 – Encontrando amigos/seguidores em comum para usuários diferentes, com uma saída em visual agradável (friends_followers__friends_followers_in_common.py)

```
# -*- coding: utf-8 -*-
import sys
import redis
from twitter__util import getRedisIdByScreenName
# Uma função que melhora o visual para os números
from twitter__util import pp
r = redis.Redis()
def friendsFollowersInCommon(screen_names):
    r.sinterstore('temp$friends_in_common',
                  [getRedisIdByScreenName(screen_name,
'friend_ids')
                      for screen_name in screen_names]
                 )
    r.sinterstore('temp$followers_in_common',
                  [getRedisIdByScreenName(screen_name,
'follower_ids')
                      for screen_name in screen_names]
                 )
    print 'Friends in common for %s: %s' % (',
'.join(screen_names),
            pp(r.scard('temp$friends_in_common')))
```

```
    print 'Followers in common for %s: %s' % (',
'.join(screen_names),
            pp(r.scard('temp$followers_in_common')))
    # Limpa o espaço de trabalho
    r.delete('temp$friends_in_common')
    r.delete('temp$followers_in_common')
if __name__ == "__main__":
    if len(sys.argv) < 3:
        print >> sys.stderr, "Please supply at least two screen
names."
        sys.exit(1)
    # Nota:
    # Presumimos que os nomes de tela que você está
    # fornecendo já foram adicionados ao Redis.
    # Consulte friends_followers__get_friends__refactored.py
    friendsFollowersInCommon(sys.argv[1:])
```

Note que ainda que os valores nos conjuntos de trabalho sejam valores de IDs, você poderia facilmente utilizar a função `randomkey` do Redis para obter uma amostra de amigos e seguidores, e utilizar a função `getUserInfo` do exemplo 4.5 para coletar informações úteis a partir desses dados, como nomes de tela, tweets mais recentes, localizações etc.

Medição da influência

Quando alguém compartilha informações em um serviço como o Twitter, é natural que se pergunte quão profundamente esses dados penetrarão na rede como um todo por meio de eventuais retweets. Não é absurdo presumir que, quanto mais seguidores uma pessoa tiver, maior o potencial de que seus tweets sejam "retwittados". Usuários com um grande percentual de seus tweets originais retwittados podem ser considerados mais influentes do que aqueles para os quais isso ocorre em menor frequência. Por sua vez, usuários que têm um grande percentual de tweets que não são originalmente de sua autoria retwittados podem ser considerados *mavens* – pessoas excepcionalmente bem conectadas, que gostam de compartilhar informações[4]. Uma forma simples de medir a influência relativa de dois ou mais usuários é simplesmente comparar seu número de seguidores, pois cada seguidor visualizará diretamente seus tweets. Sabemos, graças ao exemplo 4.6, que podemos obter o número de seguidores (e amigos) de um usuário por meio das APIs `/users/lookup` e

/users/show. Extrair informações dessas APIs é muito simples:

```
for screen_name in screen_names:
    _json = json.loads(r.get(getRedisIdByScreenName(screen_name,
"info.json")))
    n_friends, n_followers = _json['friends_count'],
_json['followers_count']
```

Contar o número de seguidores é interessante, mas há muito mais a fazer. Por exemplo, um usuário pode não ter a popularidade de um maven, como Tim O´Reilly, mas se você o tiver como seguidor e ele retwittá-lo, você terá acesso a uma vasta rede de pessoas que podem passar a segui-lo caso o considerem interessante. Assim, uma abordagem que pode ser utilizada para calcular a influência potencial de usuários é não apenas calcular seu número de seguidores, mas estender a rede, alcançando mais alguns níveis. De fato, para tanto, podemos utilizar a mesma abordagem de largura-primeiro que apresentamos no exemplo 2.4.

O exemplo 4.8 ilustra uma função de rastreamento generalizada que aceita uma lista de nomes de tela, uma profundidade de rastreamento, e parâmetros que controlam quantos amigos e seguidores devem ser recuperados. Os parâmetros `friends_limit` e `followers_limit` controlam quantos itens devem ser buscados nas APIs de grafo social (em lotes de 5.000), enquanto `friends_sample` e `followers_sample` controlam quantos objetos de usuário devem ser recuperados (em lotes de 100). Uma função atualizada para `getUserInfo` também foi incluída, para refletir a passagem dos parâmetros de amostragem.

Exemplo 4.8 – Rastreamento de conexões de amigos/seguidores (friends_followers__crawl.py)

```
# -*- coding: utf-8 -*-

import sys
import redis
import functools
from twitter__login import login
from twitter__util import getUserInfo
from twitter__util import _getFriendsOrFollowersUsingFunc

SCREEN_NAME = sys.argv[1]

t = login()
r = redis.Redis()

# Alguns encapsuladores em torno de
_getFriendsOrFollowersUsingFunc que
```

```
# criam funções de conveniência
getFriends = functools.partial(_getFriendsOrFollowersUsingFunc,
                               t.friends.ids, 'friend_ids', t, r)
getFollowers = functools.partial(_getFriendsOrFollowersUsingFunc,
                                 t.followers.ids, 'follower_ids',
t, r)
def crawl(
    screen_names,
    friends_limit=10000,
    followers_limit=10000,
    depth=1,
    friends_sample=0.2, #XXX
    followers_sample=0.0,
    ):
    getUserInfo(t, r, screen_names=screen_names)
    for screen_name in screen_names:
    friend_ids = getFriends(screen_name, limit=friends_limit)
    follower_ids = getFollowers(screen_name,
limit=followers_limit)
    friends_info = getUserInfo(t, r, user_ids=friend_ids,
sample=friends_sample)
    followers_info = getUserInfo(t, r, user_ids=follower_ids,
sample=followers_sample)
    next_queue = [u['screen_name'] for u in friends_info +
followers_info]
    d = 1
    while d < depth:
        d += 1
        (queue, next_queue) = (next_queue, [])
        for _screen_name in queue:
            friend_ids = getFriends(_screen_name,
limit=friends_limit)
            follower_ids = getFollowers(_screen_name,
limit=followers_limit)
            next_queue.extend(friend_ids + follower_ids)
            # Note que esta função toma um kw entre 0.0 e 1.0
chamado sample, que permite que você
            # rastreie somente uma amostra randômica de nós em
qualquer nível específico do grafo
            getUserInfo(user_ids=next_queue)
if __name__ == '__main__':
```

```
    if len(sys.argv) < 2:
        print "Please supply at least one screen name."
    else:
        crawl([SCREEN_NAME])
      # Os dados agora estão no sistema. Faz algo interessante.
Por exemplo,
      # localiza os seguidores mais populares de alguém como
indicador de sua influência potencial.
      # Consulte
friends_followers__calculate_avg_influence_of_followers.py
```

Presumindo que você tenha executado `crawl` com números suficientemente elevados para `friends_limit` e `followers_limit`, de modo a obter todos os IDs de amigos e seguidores de um usuário, falta apenas analisar uma amostra randômica de tamanho suficiente para que possamos calcular métricas interessantes, como o número médio de seguidores a um nível de distância. Também pode ser produtivo verificar os *N* principais seguidores desse usuário para ter uma noção de quem ele pode estar influenciando. O exemplo 4.9 demonstra uma abordagem possível que acessa os dados do Redis e calcula os seguidores mais populares de Tim O'Reilly.

Exemplo 4.9 – Cálculo dos seguidores mais populares de um usuário do Twitter (friends_followers__calculate_avg_influence_of_followers.py)

```
# -*- coding: utf-8 -*-

import sys
import json
import locale
import redis
from prettytable import PrettyTable

# Imprimindo os números com visual melhorado
from twitter__util import pp

# Estas funções criam chaves consistentes a partir de
# nomes de tela e valores de id de usuário
from twitter__util import getRedisIdByScreenName
from twitter__util import getRedisIdByUserId

SCREEN_NAME = sys.argv[1]

locale.setlocale(locale.LC_ALL, '')

def calculate():
    r = redis.Redis() # Configurações de conexão default em
```

```
localhost
    follower_ids =
list(r.smembers(getRedisIdByScreenName(SCREEN_NAME,
'follower_ids')))
    followers = r.mget([getRedisIdByUserId(follower_id,
'info.json')
                        for follower_id in follower_ids])
    followers = [json.loads(f) for f in followers if f is not
None]
    freqs = {}
    for f in followers:
        cnt = f['followers_count']
        if not freqs.has_key(cnt):
            freqs[cnt] = []
        freqs[cnt].append({'screen_name': f['screen_name'],
'user_id': f['id']})
    # Pode demorar alguns minutos para calcular freqs, por isso
salve um snapshot para uso futuro
    r.set(getRedisIdByScreenName(SCREEN_NAME, 'follower_freqs'),
json.dumps(freqs))
    keys = freqs.keys()
    keys.sort()
    print 'The top 10 followers from the sample:'
    fields = ['Date', 'Count']
    pt = PrettyTable(fields=fields)
    [pt.set_field_align(f, 'l') for f in fields]
    for (user, freq) in reversed([(user['screen_name'], k) for k
in keys[-10:]
                                  for user in freqs[k]]):
        pt.add_row([user, pp(freq)])
    pt.printt()
    all_freqs = [k for k in keys for user in freqs[k]]
    avg = reduce(lambda x, y: x + y, all_freqs) / len(all_freqs)
    print "\nThe average number of followers for %s's followers:
%s" \
        % (SCREEN_NAME, pp(avg))
# psyco pode compilar somente funções, por isso encapsule o
código em uma função
try:
    import psyco
```

```
    psyco.bind(calculate)
except ImportError, e:
    pass # psyco não instalado
calculate()
```

Em muitas situações comuns nas quais temos o processamento de grandes quantidades de dados, o módulo `psyco` (*http://psyco.sourceforge.net*) pode compilar dinamicamente o código e produzir melhorias de desempenho significativas. Mesmo sendo totalmente opcional, vale a pena analisá-lo calmamente, caso você esteja realizando cálculos que demorem mais do que poucos segundos.

A seguir, veja a saída para uma amostra de 150.000 (aproximadamente 10%) dos seguidores de Tim O'Reilly. Em uma análise estatística, utilizar um tamanho de amostra tão grande para determinada população garante uma margem de erro pequena e um alto nível de confiança (*http://en.wikipedia.org/wiki/Confidence_interval*)[5]. Ou seja, os resultados podem ser considerados muito representativos, ainda que não idênticos à absoluta verdade com referência a essa população:

```
The top 10 followers from the sample:
aplusk 4,993,072
BarackObama 4,114,901
mashable 2,014,615
MarthaStewart 1,932,321
Schwarzenegger 1,705,177
zappos 1,689,289
Veronica 1,612,827
jack 1,592,004
stephenfry 1,531,813
davos 1,522,621

The average number of followers for timoreilly's followers: 445
```

Interessante notar que alguns nomes familiares surgem na lista, incluindo alguns dos usuários mais populares do Twitter: Ashton Kutcher (*@aplusk*), Barack Obama, Martha Stewart, e Arnold Schwarzenegger, dentre outros. Se removermos esses 10 principais seguidores e recalcularmos os dados, veremos que o número médio de seguidores dos seguidores de Tim cai para aproximadamente 284. Entretanto, removendo todos os seguidores com menos de 10 seguidores, elevamos significativamente o número para mais de 1.000. Note que há dezenas de milhares de seguidores nesse intervalo e, ao verificar brevemente seus perfis, podemos compreender melhor a situação: muitos desses usuários são contas de spam, usuários que protegem seus tweets etc. Remover os principais 10 seguidores e todos os seguidores com menos de 10

seguidores poderia resultar em uma métrica mais adequada ao que pretendemos; ao aplicarmos essas limitações, obtemos um número em torno de 800, ainda consideravelmente alto. Fica evidente o alcance que um retweet pode ter quando feito por um usuário popular, que tem muitas conexões com outros usuários populares.

Construção de grafos de amizade

Este capítulo utilizou conjuntos como estrutura primária de dados para armazenar e manipular dados, pois trabalhamos principalmente com estruturas de dados como valores de ID para as quais operações de conjuntos fornecem ótima funcionalidade, exigindo pouco esforço. Ainda que existam algumas ferramentas de uso geral adequadas para a maioria dos trabalhos, não há ferramentas ideais para todos os propósitos. Há um preço que deve ser pago de alguma forma; em termos gerais, esse preço é pago antecipadamente, quando você armazena dados em índices especiais (como vimos no caso do CouchDB, no capítulo anterior), ou no momento da consulta, caso em que você não conta com o benefício de um índice criado com antecedência. Se você deseja o melhor de ambas as opções, terá de lidar com operações de armazenamento redundante e com as complexidades associadas à desnormalização dos dados – sem mencionar a manutenção do código em si.

Quando você começa a fazer perguntas referentes a uma topologia de rede, pode ser interessante exportar seus dados do Redis para um banco de dados de grafos, como o NetworkX (apresentado no capítulo 1). Há muitos grafos que podem ser construídos, mas vamos presumir que você esteja interessado em analisar as amizades que existem na rede social de alguém, e por isso deseja um grafo que indique quem é amigo de quem. Dado que grafos sociais oferecem insights poderosíssimos, e que o exemplo 4.8 já fornece os requisitos para rastreamento de relacionamentos no Twitter, não tenha pressa e colete dados referentes às amizades de algum dos usuários mais interessantes do Twitter. O restante desta seção apresenta alguns exemplos para análise desses dados.

Presumindo que você tenha coletado dados de amizades e armazenado essas informações no Redis, o exemplo 4.10 demonstra como construir um grafo de todas as amizades comuns que existem para um usuário. Basicamente, você percorre sistematicamente todos os usuários em um

loop aninhado e cria arestas sempre que encontra uma amizade entre duas pessoas no universo considerado. Assim que os dados forem disponibilizados no NetworkX, você terá à sua disposição um arsenal completo de algoritmos de grafos e utilitários. A próxima seção investiga algumas das informações mais valiosas que podem ser mineradas a partir de um grafo de amizade.

Exemplo 4.10 – Exportação de dados de amigos/seguidores a partir do Redis para o NetworkX, para realizar analíticas de grafo (friends_followers__redis_to_networkx.py)

```
# -*- coding: utf-8 -*-
# Resumo: Constrói um dígrafo em que existe uma aresta entre dois
usuários
# caso o nó de origem esteja seguindo o nó de destino
import os
import sys
import json
import networkx as nx
import redis
from twitter__util import getRedisIdByScreenName
from twitter__util import getRedisIdByUserId
SCREEN_NAME = sys.argv[1]
g = nx.Graph()
r = redis.Redis()
# Compute todos os ids para os nós que surgem no grafo
friend_ids = list(r.smembers(getRedisIdByScreenName(SCREEN_NAME,
'friend_ids')))
id_for_screen_name =
json.loads(r.get(getRedisIdByScreenName(SCREEN_NAME,
'info.json')))['id']
ids = [id_for_screen_name] + friend_ids
for current_id in ids:
    print >> sys.stderr, 'Processing user with id', current_id
    try:
        current_info =
json.loads(r.get(getRedisIdByUserId(current_id, 'info.json')))
        current_screen_name = current_info['screen_name']
        friend_ids =
list(r.smembers(getRedisIdByScreenName(current_screen_name,'frien
d_ids')))
        # filtre ids para esta pessoa se eles também não forem
```

```
amigos de SCREEN_NAME,
        # o que é a base da consulta
        friend_ids = [fid for fid in friend_ids if fid in ids]
    except Exception, e:
        print >> sys.stderr, 'Skipping', current_id
    for friend_id in friend_ids:
        try:
            friend_info =
json.loads(r.get(getRedisIdByUserId(friend_id, 'info.json')))
        except TypeError, e:
            print >> sys.stderr, '\tSkipping', friend_id, 'for',
current_screen_name
            continue
        g.add_edge(current_screen_name,
friend_info['screen_name'])
# Salve o grafo no disco...
if not os.path.isdir('out'):
    os.mkdir('out')
filename = os.path.join('out', SCREEN_NAME + '.gpickle')
nx.write_gpickle(g, filename)
print 'Pickle file stored in: %s' % filename
# É possível acessar os dados da seguinte maneira...
# g = nx.read_gpickle(os.path.join('out', SCREEN_NAME +
'.gpickle'))
```

Tendo construído uma representação conveniente das amizades que você rastreou e armazenou no Redis, vamos agora à parte mais divertida: a análise desses dados.

Detecção e análise de cliques

O exemplo 4.10 fornece uma rotina básica que demonstra como você pode descobrir *cliques* de amizades (*http://en.wikipedia.org/wiki/Clique*) que existem para um dado usuário, importando dados do Redis e adicionando repetidamente arestas ao grafo por meio da operação idempotente `add_edge`. Por exemplo, caso Abe seja amigo de Bob, Carol e Dale, e Bob e Carol também sejam amigos, a clique máxima no grafo existe entre Abe, Bob e Carol. Todavia, se Abe, Bob, Carol e Dale fossem todos amigos mútuos, o grafo seria completamente conectado, e a clique máxima teria como tamanho 4. A adição de nós ao grafo pode criar

cliques adicionais, mas não afeta necessariamente o tamanho da clique máxima no grafo. No contexto da web social, cliques são fascinantes, pois representam amizades mútuas, e a clique máxima é interessante, pois indica o maior conjunto de amizades no grafo. Dadas duas redes sociais, comparar o tamanho das cliques máximas de amizades pode fornecer muitas informações sobre dinâmicas de grupo, e outros tópicos.

A figura 4.2 ilustra um exemplo de grafo no qual temos a clique máxima destacada. Poderíamos dizer que este grafo tem uma clique máxima de tamanho 4.

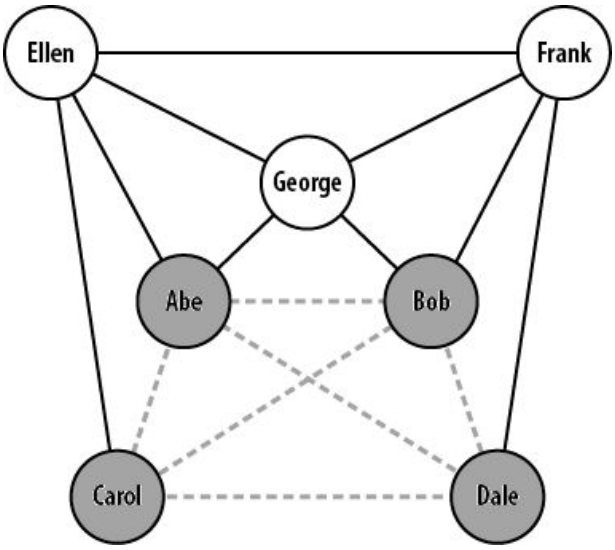


Figura 4.2 – Grafo de exemplo contendo uma clique máxima de tamanho 4.

Em termos técnicos, há uma diferença entre clique maximal e clique máxima. A clique máxima é a maior clique do grafo (ou cliques, se tiverem o mesmo tamanho). Uma clique maximal, por outro lado, é a que não é subgrafo de outra clique. A clique máxima é também uma clique maximal no sentido de que não é subgrafo de nenhuma outra clique. Entretanto, diversas outras cliques maximais muitas vezes existem em grafos, e não precisam necessariamente compartilhar nós com a clique máxima. A figura 4.2, por exemplo, ilustra uma clique máxima de tamanho 4, mas há muitas outras cliques maximais de tamanho 3 no grafo.

Encontrar cliques é um problema NP-completo[6] (implicando um tempo de execução exponencial), mas o NetworkX fornece o método `find_cliques` (*http://networkx.lanl.gov/reference/generated/networkx.find_cliques.html#networkx.find_cliques*), oferecendo uma implementação sólida que cuida de todo o trabalho pesado. Ainda assim, esteja avisado de que sua execução pode ser demorada, caso os grafos cresçam.

O exemplo 4.11, que veremos a seguir, poderia ser modificado de muitas formas para computar dados interessantes, desde que você primeiro

coletasse as informações necessárias de amizade. Presumindo que você tenha construído primeiro um grafo como demonstramos na seção prévia, ele mostra como utilizar algumas das APIs do NetworkX para encontrar e analisar cliques.

Exemplo 4.11 – Uso do NetworkX para encontrar cliques nos grafos (friends_followers__clique_analysis.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import networkx as nx

G = sys.argv[1]

g = nx.read_gpickle(G)

# Encontrar cliques é um problema complexo, por isso pode
# demorar um pouco para grafos maiores.
# Consulte http://en.wikipedia.org/wiki/NP-complete e
# http://en.wikipedia.org/wiki/Clique_problem

cliques = [c for c in nx.find_cliques(g)]

num_cliques = len(cliques)

clique_sizes = [len(c) for c in cliques]
max_clique_size = max(clique_sizes)
avg_clique_size = sum(clique_sizes) / num_cliques

max_cliques = [c for c in cliques if len(c) == max_clique_size]

num_max_cliques = len(max_cliques)

max_clique_sets = [set(c) for c in max_cliques]
people_in_every_max_clique = list(reduce(lambda x, y:
x.intersection(y), max_clique_sets))

print 'Num cliques:', num_cliques
print 'Avg clique size:', avg_clique_size
print 'Max clique size:', max_clique_size
print 'Num max cliques:', num_max_cliques
print
print 'People in all max cliques:'
print json.dumps(people_in_every_max_clique, indent=4)
print
print 'Max cliques:'
print json.dumps(max_cliques, indent=4)
```

Um exemplo de saída do script pode ser visto a seguir, para as mais de 600 amizades de Tim O'Reilly, e revela alguns insights interessantes. Há

mais de 750.000 cliques totais na rede, com um tamanho médio de clique de 14, e um tamanho de clique máxima de 26. Em outras palavras, o maior número de pessoas totalmente conectadas entre os amigos de Tim é de 26, e há 6 cliques distintas com esse tamanho, como ilustrado pela saída no exemplo 4.12. Os mesmos indivíduos representam 20/26 da população em todas as 6 dessas cliques. Você pode pensar nesses membros como de importância fundamental para a rede de amigos de Tim. Em termos gerais, ser capaz de descobrir cliques em grafos (especialmente cliques maximais e a clique máxima) pode resultar em insights tremendamente poderosos referentes a dados complexos. Uma análise da similaridade/diversidade do conteúdo dos tweets entre membros de uma clique extensa seria um exercício valioso de análise de tweets. Falaremos mais sobre isso no próximo capítulo.

Exemplo 4.12 – Exemplo de saída do exemplo 4.11, ilustrando membros comuns de cliques máximas

```
Num cliques: 762573
Avg clique size: 14
Max clique size: 26
Num max cliques: 6
People in all max cliques:
[
    "kwerb",
    "johnbattelle",
    "dsearls",
    "JPBarlow",
    "cshirky",
    "leolaporte",
    "MParekh",
    "mkapor",
    "steverubel",
    "stevenbjohnson",
    "LindaStone",
    "godsdog",
    "Joi",
    "jayrosen_nyu",
    "dweinberger",
    "timoreilly",
    "ev",
    "jason_pontin",
    "kevinmarks",
```

```
    "Mlsif"
]
Max cliques:
[
    [
        "timoreilly",
        "anildash",
        "cshirky",
        "ev",
        "pierre",
        "mkapor",
        "johnbattelle",
        "kevinmarks",
        "MParekh",
        "dsearls",
        "kwerb",
        "Joi",
        "LindaStone",
        "dweinberger",
        "SteveCase",
        "leolaporte",
        "steverubel",
        "Borthwick",
        "godsdog",
        "edyson",
        "dangillmor",
        "Mlsif",
        "JPBarlow",
        "stevenbjohnson",
        "jayrosen_nyu",
        "jason_pontin"
    ],
    ...
    ]
```

Fica claro que poderíamos fazer mais do que apenas detectar cliques. Dentre as possibilidades, poderíamos representar em um mapa a localização das pessoas envolvidas nas cliques, para verificar se há uma correlação entre as redes conectadas e localizações geográficas, analisando informações nos dados de seus perfis e/ou no conteúdo de seus tweets. O próximo capítulo trata da análise de tweets. Por isso, você poderá retornar a essa ideia depois de lê-lo. Entretanto, primeiro vamos investigar

brevemente uma interessante API Web oferecida pela InfoChimps que pode ser utilizada para confirmar algumas das análises que realizamos.

API "Strong Links" da InfoChimps

A InfoChimps (*http://infochimps.org*) é uma organização que oferece um imenso catálogo de dados. Dentre suas muitas ofertas, está um enorme arquivo de dados históricos do Twitter e muitas outras APIs de valor agregado que podem ser utilizadas para operar sobre esses dados. Uma das mais interessantes APIs de Metrics e Analytics (*http://api.infochimps.com/describe/soc/net/tw*) é a API Strong Links, que retorna uma lista dos usuários com os quais o usuário consultado se comunica com mais frequência. Para acessá-la, basta que você registre uma conta gratuita e obtenha uma chave para a API. Depois, é só realizar uma simples solicitação GET em um URL. A única ressalva é que os resultados fornecidos apresentam valores de ID de usuário, que devem ser transformados em nomes de tela para que tenham utilidade[7]. Felizmente, isso pode ser feito com facilidade utilizando código que vimos antes neste capítulo. O script no exemplo 4.13 demonstra como utilizar essa API e também emprega o Redis para converter nomes de tela a partir de valores de ID.

Exemplo 4.13 – Coleta de dados da API Strong Links da Infochimps (friends_followers__infochimps_strong_links.py)

```
# -*- coding: utf-8 -*-

import sys
import urllib2
import json
import redis

from twitter__util import getRedisIdByUserId

SCREEN_NAME = sys.argv[1]
API_KEY = sys.argv[2]
API_ENDPOINT = \
    'http://api.infochimps.com/soc/net/tw/strong_links.json?
screen_name=%s&apikey=%s'

r = redis.Redis() # configurações de conexão default em localhost

try:
    url = API_ENDPOINT % (SCREEN_NAME, API_KEY)
    response = urllib2.urlopen(url)
```

```
except urllib2.URLError, e:
    print 'Failed to fetch ' + url
    raise e
strong_links = json.loads(response.read())
# transforme em screen names e imprima na tela:
print "%s's Strong Links" % (SCREEN_NAME, )
print '-' * 30
for sl in strong_links['strong_links']:
    if sl is None:
        continue
try:
    user_info = json.loads(r.get(getRedisIdByUserId(sl[0],
'info.json')))
    print user_info['screen_name'], sl[1]
except Exception, e:
    print >> sys.stderr, "ERROR: couldn't resolve screen_name
for", sl
    print >> sys.stderr, "Maybe you haven't harvested data for
this person yet?"
```

O mais interessante é que dos 20 indivíduos envolvidos em todas as 6 cliques máximas de Tim, apenas *@kevinmarks* surge no conjunto de resultados. Isso parece implicar que simplesmente estar conectado a amigos no Twitter não significa que você se comunica diretamente com essas pessoas com frequência. Entretanto, avançando um capítulo, você verá na tabela 5.2, que muitos dos usuários que Tim retwitta aparecem frequentemente na lista de strong links – mais especificamente *@ahier*, *@gnat*, *@jamesoreilly*, *@pahlkadot*, *@OReillyMedia* e *@monkchips* (note que *@monkchips* é o segundo na lista de strong links, como mostra o exemplo 4.14). A InfoChimps não declara exatamente quais são os cálculos utilizados em sua API Strong Links, exceto por dizer que esses dados são construídos sobre o conteúdo dos tweets, cuja análise é o tópico do próximo capítulo. A lição que aprendemos aqui é que amizades próximas no Twitter não têm necessariamente de implicar em comunicações frequentes (e que a InfoChimps oferece APIs muito interessantes que merecem ser verificadas).

Exemplo 4.14 – Exemplo de resultados da API Strong Links da InfoChimps a partir do exemplo 4.13

```
timoreilly's Strong Links
------------------------------
jstan 20.115004
```

```
monkchips 11.317813
govwiki 11.199023
ahier 10.485066
ValdisKrebs 9.384349
SexySEO 9.224745
tdgobux 8.937282
Scobleizer 8.406802
cheeky_geeky 8.339885
seanjoreilly 8.182084
pkedrosky 8.154991
OReillyMedia 8.086607
FOSSwiki 8.055233
n2vip 8.052422
jamesoreilly 8.015188
pahlkadot 7.811676
ginablaber 7.763785
kevinmarks 7.7423387
jcantero 7.64023
gnat 7.6349654
KentBottles 7.40848
Bill_Romanos 7.3629074
make 7.326427
carlmalamud 7.3147154
rivenhomewood 7.276802
webtechman 7.1044493
```

Visualização interativa em grafo 3D

Está um pouco fora de nosso escopo abordar profundamente algoritmos de layout de grafos, mas é difícil desperdiçar a oportunidade de apresentar o Ubigraph (*http://ubietylab.net/ubigraph/*), ferramenta 3D de visualização de grafos. Mesmo que não haja valor analítico na visualização de uma clique, este ainda é um bom exercício para entender como tudo funciona. O Ubigraph é fácil de instalar e vem com associações para muitas linguagens de programação populares, incluindo a Python. O exemplo 4.15 ilustra um exemplo de uso do Ubigraph para criação de um grafo dos membros comuns entre cliques máximas de uma rede. A má notícia é que esta é uma daquelas oportunidades em que usuários do Windows provavelmente terão de executar uma máquina virtual Linux, uma vez que é improvável haver uma compilação do servidor Ubigraph tão cedo. A boa notícia é que não é tão difícil fazê-lo,

mesmo que você não tenha conhecimento avançado em Linux.

Exemplo 4.15 – Visualização de dados de grafos com o Ubigraph (friends_followers___ubigraph.py)

```
# -*- coding: utf-8 -*-

import sys
import json
import networkx as nx

# Acompanha o Ubigraph no diretório examples/Python
import ubigraph

SCREEN_NAME = sys.argv[1]
FRIEND = sys.argv[2]

g = nx.read_gpickle(SCREEN_NAME + '.gpickle')

cliques = [c for c in nx.find_cliques(g) if FRIEND in c]
max_clique_size = max([len(c) for c in cliques])
max_cliques = [c for c in cliques if len(c) == max_clique_size]

print 'Found %s max cliques' % len(max_cliques)
print json.dumps(max_cliques, indent=4)

U = ubigraph.Ubigraph()
U.clear()
small = U.newVertexStyle(shape='sphere', color='#ffff00',
size='0.2')
largeRed = U.newVertexStyle(shape='sphere', color='#ff0000',
size='1.0')

# encontre as pessoas que são comuns a todas as cliques para
visualização

vertices = list(set([v for c in max_cliques for v in c]))
vertices = dict([(v, U.newVertex(style=small, label=v)) for v in
vertices if v
                not in (SCREEN_NAME, FRIEND)])

vertices[SCREEN_NAME] = U.newVertex(style=largeRed,
label=SCREEN_NAME)
vertices[FRIEND] = U.newVertex(style=largeRed, label=FRIEND)

for v1 in vertices:
for v2 in vertices:
    if v1 == v2:
        continue
    U.newEdge(vertices[v1], vertices[v2])
```

Em suma, você simplesmente cria um grafo e adiciona vértices e arestas a ele. Note que, diferentemente do que temos no NetworkX, os nós não são

definidos por seus labels, por isso você deve ter cuidado para não adicionar nós duplicados ao mesmo grafo. O exemplo precedente demonstra a construção de um grafo pela iteração de um dicionário de vértices com pequenas customizações para que alguns nós possam ser visualizados mais facilmente que outros. Seguindo o padrão dos exemplos prévios, essa listagem específica visualiza os membros comuns das cliques máximas compartilhadas por um usuário (Tim O'Reilly, neste caso) e um amigo (definido por `FRIEND`) – em outras palavras, a maior clique contendo dois nós compartilhados. A figura 4.3 mostra uma imagem do Ubigraph funcionando. Como praticamente tudo neste livro, isso é apenas uma introdução mínima, que deve servir para inspirá-lo a fazer coisas incríveis com seus dados.

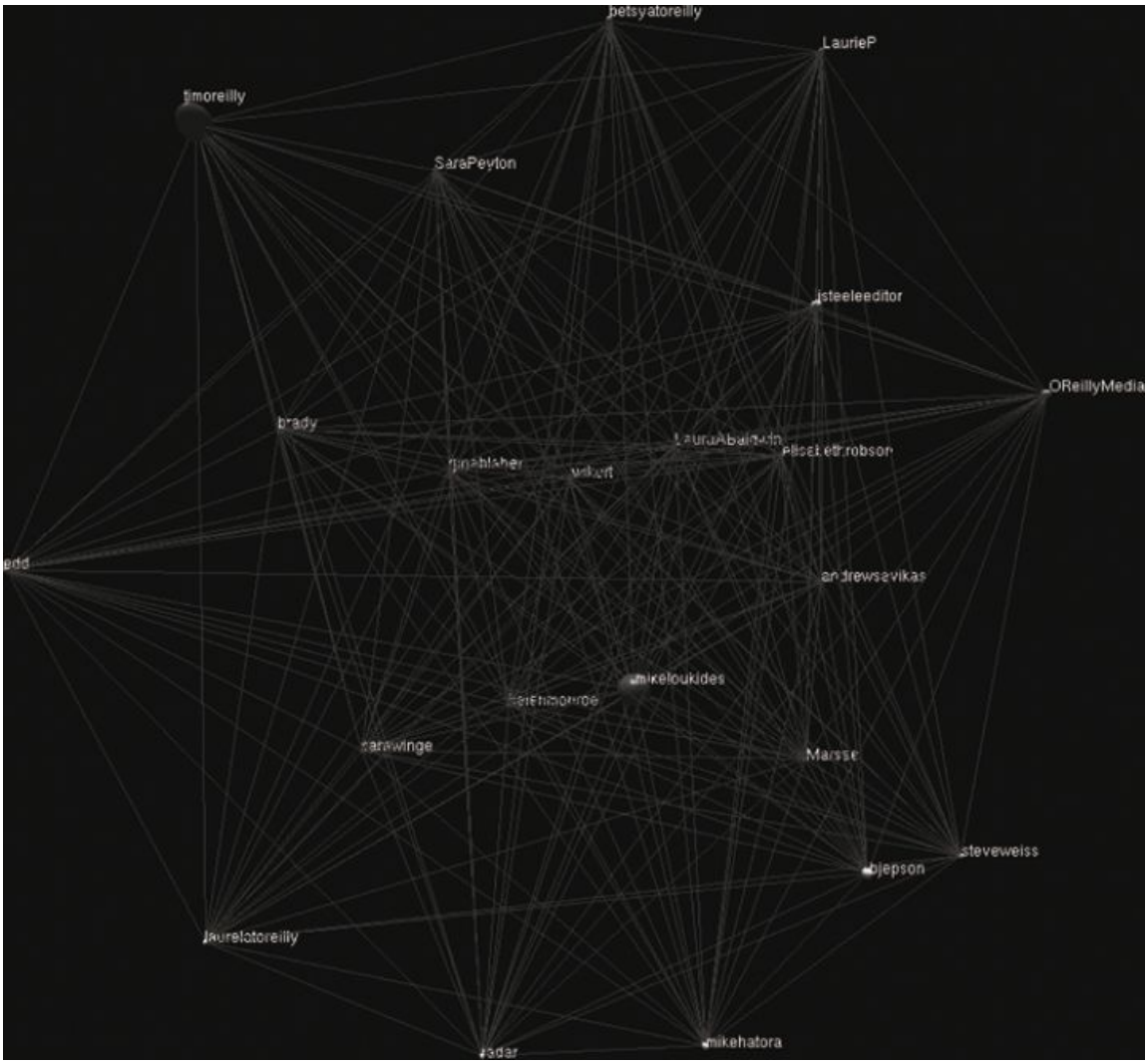


Figura 4.3 – Imagem de uma visualização 3D interativa de membros comuns presentes nas cliques maximais contendo tanto @timoreilly quanto @mikeloukides (a tela não faz justiça

ao resultado; você realmente tem de experimentar esse recurso de modo interativo!).

Resumo

Este capítulo apenas arranhou a superfície do que pode ser feito com dados do Twitter, concentrando-se principalmente em relacionamentos de amigos/seguidores que podem ser analisados. Um exemplo da implementação de uma ferramenta de linha de comando que concentra toda a funcionalidade mostrada neste capítulo está disponível em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/TwitterSocialGraphUtility.py*, e você deve ser capaz de adaptá-la facilmente aos seus propósitos (se o fizer, por favor não deixe de oferecê-la no GitHub para disponibilizar suas alterações a todos). Algumas sugestões interessantes para exercícios divertidos são:

- Como não tratamos de dados geográficos neste capítulo, tente representar um histograma dos locais de origem de seus seguidores e visualizá-lo em um mapa on-line. O capítulo 6 estuda de modo mais profundo dados geográficos, fornecendo uma introdução razoável a algumas tecnologias de mapeamento que podem ser utilizadas.
- Experimente realizar algumas análises básicas de frequência nas palavras que estão nos campos de descrição dos objetos de usuário de seus amigos/seguidores. Os capítulos 7 e 8 fornecem ferramentas e técnicas que podem lhe ajudar nessa tarefa.
- Experimente criar um histograma do número de tweets de acordo com cada um de seus seguidores, para ver quem são os mais conversadores. Esse dado está disponível como `statuses_count` nos objetos de usuário.
- Neste capítulo, não falamos sobre listas – uma forma de agrupar usuários, geralmente a partir de uma característica comum que compartilham. Verifique a documentação da API (*http://dev.twitter.com/doc*) sobre esse assunto, e escreva algum código para tentar determinar quem listou você (ou algum usuário do Twitter que você admira) e quais as semelhanças entre as pessoas da lista. Um bom ponto de partida é verificar as listas em que uma pessoa está presente diretamente por meio da interface web pública do Twitter.

---

1 Quando da redação deste livro, o Twitter limita solicitações OAuth a 350 por hora, e

solicitações anônimas a 150 por hora, pois acredita que tais limites devem ser o suficiente para o funcionamento de praticamente todas as aplicações clientes. Caso não sejam, você provavelmente não está construindo sua aplicação direito.

2 Quando da redação deste livro, em dezembro de 2010, o Twitter implementou o OAuth 1.0a, mas você deve esperar suporte ao OAuth 2.0 em algum momento de 2011.

3 Sempre que o Twitter excede sua capacidade, um erro HTTP 503 é emitido. Em um navegador, a página de erro exibe a imagem da infame "fail whale", a baleia do Twitter. Consulte *http://twitter.com/503*.

4 Consulte The Tipping Point, por Malcolm Gladwell (Back Bay Books), para uma ótima explicação sobre esse termo.

5 Considera-se uma margem de erro em torno de 0,14 para um nível de confiança de 99%.

6 N.T.: na teoria da complexidade computacional, a classe de complexidade NP-completo é o subconjunto dos problemas de decisão em NP, de tal modo que todo problema em NP se pode reduzir, com uma redução de tempo polinomial, a um dos problemas NP-Completo (fonte: Wikipédia).

7 Uma API "Who Is" da InfoChimps (*http://api.infochimps.com/describe/soc/net/tw/whois*) deverá ser criada em breve e fornecerá uma maneira de transformar nomes de tela a partir de IDs de usuário.

CAPÍTULO 5

Twitter: o tweet, todo o tweet, o nada além do tweet

Tweet e RT estavam sentados sobre uma cerca. O tweet caiu.

Quem sobrou?

Neste capítulo, utilizaremos amplamente as capacidades de mapeamento/redução do CouchDB para explorar as entidades dos tweets (@menções, #hashtags etc.) e tentar responder à seguinte pergunta: "Sobre o que todos estão falando?" Com um fluxo que excede 50 milhões de tweets por dia (*http://mashable.com/2010/02/22/twitter-50-million-tweets/*) e velocidades de pico ocasionais de mais de 3 mil tweets por segundo (*http://blog.twitter.com/2010/06/another-big-record-part-deux.html*), há vasto potencial na mineração do conteúdo dos tweets, e este é o capítulo em que iniciaremos esse processo. Enquanto o capítulo anterior tratou principalmente dos vínculos que existem entre amigos e seguidores, e de como podemos mostrá-los em grafos sociais, este procurará descobrir o máximo sobre os usuários do Twitter inspecionando as entidades (*http://dev.twitter.com/pages/tweet_entities*) presentes em seus tweets. Você também verá menções ao Redis, utilizado para acessar os dados de usuário que você coletou no capítulo 4, e ao NetworkX, para análises de grafos. Tantos tweets e tão pouco tempo – vamos começar!

É altamente recomendado que você leia os capítulos 3 e 4 antes de avançar. Muito da discussão que apresentaremos está fundada nas bases estabelecidas por esses capítulos, incluindo o Redis e o CouchDB, que utilizaremos novamente.

Caneta : Espada :: Tweet : Metralhadora (?!?)

Se a caneta é mais poderosa que a espada, o que dizer do tweet? Sabemos de muitos incidentes interessantes nos quais o Twitter ajudou a salvar vidas, sendo que um dos mais notórios foi o famoso tweet "Arrested" de James Karl Buck (*http://techcrunch.com/2008/04/16/twitter-saves-man-*

from-egyptian-justice/), que resultou em sua pronta libertação, após ter sido detido por autoridades egípcias. Não é necessário fazer muita pesquisa para encontrar evidências de incidentes semelhantes, assim como de incontáveis situações nas quais o Twitter foi utilizado para divulgar ações de arrecadação de recursos e outras causas benevolentes. Ter um canal de comunicação pode realmente fazer a diferença em alguns casos. Ainda assim, com frequência, a *timeline* de sua página inicial (linha do tempo que mostra o fluxo dos tweets ordenados cronologicamente, também chamada de histórico) e a timeline pública estão repletas de informações não tão dramáticas ou intrigantes. Nessas situações, filtrar e eliminar informações desnecessárias pode ajudá-lo a enxergar o panorama geral. Dado que praticamente 50% de todos os tweets contêm ao menos uma entidade (*http://dev.twitter.com/pages/tweet_entities*) intencionalmente incluída por seus autores, esses dados representam um ponto de partida lógico para análise. De fato, o Twitter reconheceu o valor das entidades e passou a expô-las diretamente nas chamadas à API da timeline, fazendo com que, a partir do início de 2010, esse tipo de consulta se tornasse algo regular em toda a API do Twitter. Considere o tweet do exemplo 5.1, recuperado de uma chamada à API da timeline com o parâmetro `include_entities=true` especificado na consulta.

Exemplo 5.1 – Exemplo de tweet de uma API de busca que ilustra entidades dos tweets

```
{
    "created_at" : "Thu Jun 24 14:21:11 +0000 2010",
    "entities" : {
                    "hashtags" : [
                        { "indices" : [ 97, 103 ],
                          "text" : "gov20"
                        },
                        { "indices" : [ 104, 112 ],
                          "text" : "opengov"
                        }
                    ],
                    "urls" : [
                        { "expanded_url" : null,
                          "indices" : [ 76, 96 ],
                          "url" : "http://bit.ly/9o4uoG"
                        }
                    ],
                    "user_mentions" : [
```

```
                    { "id" : 28165790,
                      "indices" : [ 16, 28 ],
                      "name" : "crowdFlower",
                      "screen_name" : "crowdFlower"
                    }
                ]
      },
    "id" : 16932571217,
    "text" : "Great idea from @crowdflower: Crowdsourcing the
Goldman ... #opengov",
    "user" : {
        "description" : "Founder and CEO, O'Reilly Media.
Watching the alpha ...",
        "id" : 2384071,
        "location" : "Sebastopol, CA",
        "name" : "Tim O'Reilly",
        "screen_name" : "timoreilly",
        "url" : "http://radar.oreilly.com",
    }
}
```

Por padrão, um tweet especifica muitas informações úteis sobre seu autor por meio do campo `user` no objeto `status`, mas as entidades do tweet fornecem insight quanto ao conteúdo do tweet em si. Inspecionando rapidamente um tweet de exemplo, podemos deduzir com segurança que *@timoreilly* está provavelmente interessado em tópicos transformacionais sobre governo aberto e Governo 2.0, como indicado pelas hashtags incluídas no tweet. Provavelmente também podemos inferir que *@crowdflower* tem alguma relação com esses assuntos e que o URL pode apontar para algum conteúdo relacionado. Assim, se você quisesse descobrir de forma automatizada informações adicionais sobre o autor deste tweet, poderia avançar de *@timoreilly* para *@crowdflower* e explorar os tweets desse usuário ou suas informações de perfil, iniciando com uma busca nas hashtags presentes no tweet para descobrir que tipo de informação pode ser encontrada, ou seguir o link e realizar um pouco de raspagem de página para aprender mais sobre seu contexto.

Dado que muito valor pode ser obtido analisando as entidades dos tweets, você sentirá falta desse recurso em algumas APIs e arquivos de dados antigos do Twitter, cuja mineração está se tornando cada vez mais popular. Em vez de realizar o parsing manual das entidades a partir do

texto (algo não tão fácil quando os tweets contêm caracteres Unicode arbitrários), simplesmente execute `easy_install twitter-text-py`[1] para que você possa concentrar seus esforços em problemas mais interessantes. O script no exemplo 5.2 ilustra a utilização básica de sua classe `Extractor`, que produz estrutura semelhante à exposta pelas APIs da timeline. Muito pode ser ganho pela incorporação de entidades dessa maneira, ao menos até que as entidades dos tweets se tornem o padrão.

Quando da redação deste livro, em dezembro de 2010, entidades de tweets estavam se tornando mais e mais usuais em todas as APIs, mas ainda não eram exatamente reconhecidas como regra. Este capítulo foi escrito trabalhando com a suposição de que você não deseja saber como fazer o parsing dessas entidades sozinho, mas você deve perceber que se manter atualizado quanto às novidades da API do Twitter pode economizar muito de seu trabalho. A extração manual de entidades dos tweets também poderia ser útil nas situações em que você estivesse minerando arquivos históricos de organizações, como a Infochimps ou a GNIP.

Exemplo 5.2 – Extração de entidades dos tweets com auxílio do pacote twitter_text (the_tweet__extract_tweet_entities.py)

```
# -*- coding: utf-8 -*-

import sys
import json
import twitter_text
import twitter
from twitter__login import login

# Pegue o id de um tweet clicando em um status diretamente no
twitter.com.
# Por exemplo,
http://twitter.com/#!/timoreilly/status/17386521699024896

TWEET_ID = sys.argv[1]

# Talvez você tenha de modificar suas configurações OAuth em
twitter__login.py
t = login()

def getEntities(tweet):

    # Agora, extraia várias entidades dele e construa uma
estrutura familiar

    extractor = twitter_text.Extractor(tweet['text'])

    # Note que a API de produção do Twitter contém alguns campos
adicionais no
    # hash das entidades que necessitariam de chamadas adicionais
à API para serem transformados

      entities = {}
      entities['user_mentions'] = []
```

```
        for um in
extractor.extract_mentioned_screen_names_with_indices():
            entities['user_mentions'].append(um)
        entities['hashtags'] = []
        for ht in extractor.extract_hashtags_with_indices():
            # Ajusta o nome do campo para corresponder à API
Twitter de produção
            ht['text'] = ht['hashtag']
            del ht['hashtag']
            entities['hashtags'].append(ht)
        entities['urls'] = []
        for url in extractor.extract_urls_with_indices():
            entities['urls'].append(url)
        return entities
# Busca um tweet utilizando um método da API de sua escolha e
inclui as entidades
tweet = t.statuses.show(id=TWEET_ID)
tweet['entities'] = getEntities(tweet)
print json.dumps(tweet, indent=4)
```

Agora, munidos de uma visão geral acerca das entidades dos tweets e de algumas possibilidades interessantes, vamos coletar e analisar alguns tweets.

Análise de tweets (uma entidade por vez)

O CouchDB é uma ótima ferramenta de armazenamento para coleta de tweets, já que, assim como as mensagens de e-mail que vimos no capítulo 3, tweets são convenientemente representados como documentos JSON, que podem ser submetidos a uma análise de mapeamento/redução apenas com um mínimo de esforço.

Nosso próximo script de exemplo coleta tweets de timelines. Além disso, ele é relativamente robusto, e deve ser de fácil compreensão, pois todos os módulos e grande parte do código foram previamente apresentados nos capítulos anteriores. Uma consideração particular em sua análise se refere ao fato de que ele utiliza um simples trabalho de mapeamento/redução para computar o valor máximo de ID para um tweet, e passa esse dado como uma restrição da consulta, de modo a evitar o acesso a dados duplicados a partir da API do Twitter. Para mais detalhes, consulte as

informações associadas ao parâmetro `since_id` das APIs da timeline.

Da mesma forma, vale notar que o número máximo de tweets mais recentes disponível a partir da timeline do usuário é de cerca de 3.200, enquanto a timeline da página inicial[2] retorna cerca de 800 status; assim, não é muito custoso (em termos do consumo de seu rate limit) acessar todos os dados disponíveis. Um fato que talvez não seja tão intuitivo quando interagimos pela primeira vez com as APIs da timeline é o de que solicitações para dados na timeline pública retornam apenas 20 tweets, atualizados somente a cada 60 segundos. Para coletar quantidades maiores de dados, você tem de utilizar a API de streaming.

Por exemplo, se você quisesse aprender um pouco mais sobre Tim O'Reilly, tido como a personalidade talentosa favorita do Vale do Silício[3], bastaria verificar se o CouchDB está sendo executado, e invocar o script do exemplo 5.3, da seguinte maneira:

```
$ python the_tweet__harvest_timeline.py user 16 timoreilly
```

Demoraria apenas alguns minutos para coletar praticamente 3.200 tweets contendo dados interessantes para sua análise.

Exemplo 5.3 – Coleta de tweets a partir da timeline pública de um usuário (the_tweet__harvest_timeline.py)

```
# -*- coding: utf-8 -*-

import sys
import time
import twitter
import couchdb
from couchdb.design import ViewDefinition
from twitter__login import login
from twitter__util import makeTwitterRequest

def usage():
    print 'Usage: $ %s timeline_name [max_pages] [user]' %
(sys.argv[0], )
    print
    print '\ttimeline_name in [public, home, user]'
    print '\t0 < max_pages <= 16 for timeline_name in [home,
user]'
    print '\tmax_pages == 1 for timeline_name == public'
    print 'Notes:'
    print '\t* ~800 statuses are available from the home
timeline.'
```

```
    print '\t* ~3200 statuses are available from the user
timeline.'
    print '\t* The public timeline updates once every 60 secs and
returns 20 statuses.'
    print '\t* See the streaming/search API for additional
options to harvest tweets.'
    exit()
if len(sys.argv) < 2 or sys.argv[1] not in ('public', 'home',
'user'):
    usage()
if len(sys.argv) > 2 and not sys.argv[2].isdigit():
    usage()
if len(sys.argv) > 3 and sys.argv[1] != 'user':
    usage()
TIMELINE_NAME = sys.argv[1]
MAX_PAGES = int(sys.argv[2])
USER = None
KW = { # Para a chamada à API do Twitter
    'count': 200,
    'skip_users': 'true',
    'include_entities': 'true',
    'since_id': 1,
    }
if TIMELINE_NAME == 'user':
    USER = sys.argv[3]
    KW['id'] = USER # id ou screen name
if TIMELINE_NAME == 'home' and MAX_PAGES > 4:
    MAX_PAGES = 4
if TIMELINE_NAME == 'user' and MAX_PAGES > 16:
    MAX_PAGES = 16
if TIMELINE_NAME == 'public':
    MAX_PAGES = 1
t = login()
# Estabelece uma conexão a um banco de dados do CouchDB
server = couchdb.Server('http://localhost:5984')
DB = 'tweets-%s-timeline' % (TIMELINE_NAME, )
if USER:
    DB = '%s-%s' % (DB, USER)
try:
    db = server.create(DB)
```

```
except couchdb.http.PreconditionFailed, e:
    # Já existe, por isso acrescenta a ele, lembrando-se de que
duplicatas podem ocorrer
    db = server[DB]
    # Procure evitar acrescentar dados duplicados recuperando
apenas tweets
    # mais recentes do que os já presentes no sistema. Uma
simples combinação de mapeador/redutor
    # nos permite acessar o tweet de id máximo, nos protegendo de
duplicatas para
    # as timelines da página inicial e do usuário. Não tem efeito
para a timeline pública
    def idMapper(doc):
        yield (None, doc['id'])
    def maxFindingReducer(keys, values, rereduce):
        return max(values)
    view = ViewDefinition('index', 'max_tweet_id', idMapper,
maxFindingReducer,
                          language='python')
    view.sync(db)
    KW['since_id'] = int([_id for _id in
db.view('index/max_tweet_id')][0].value)
# Coleta tweets para a dada timeline.
# Para as timelines de amigos e da página inicial, a limitação
não-oficial é de 800 status
# ainda que haja documentação que determine o contrário. A
timeline pública retorna somente 20
# status e é atualizada a cada 60 segundos.
# Consulte http://groups.google.com/group/twitter-development-
talk/browse_thread/
# thread/4678df70c301be43
# Note que os parâmetros count e since_id não têm efeito para a
timeline pública
page_num = 1
while page_num <= MAX_PAGES:
    KW['page'] = page_num
    api_call = getattr(t.statuses, TIMELINE_NAME + '_timeline')
    tweets = makeTwitterRequest(t, api_call, **KW)
    db.update(tweets, all_or_nothing=True)
    print 'Fetched %i tweets' % len(tweets)
    wait_period = 2
```

```
    page_num += 1
```

Dada uma infraestrutura básica para coleta de tweets, vamos analisar os dados obtidos e ver quais informações interessantes podemos descobrir.

Explorando os tweets (do Tim)

Esta seção investiga algumas das questões mais usuais que surgem quando realizamos uma simples verificação das entidades mineradas da timeline de usuário de Tim O'Reilly. Ainda que, por um propósito educacional, Tim tenha concordado em oferecer seus dados para análise, você poderá muito facilmente reconfigurar esses scripts para analisar seus próprios tweets, ou aplicá-los a qualquer outro usuário interessante do Twitter. Você poderá até iniciar utilizando uma quantidade considerável de dados da timeline pública, coletados como base inicial de exploração. Algumas questões interessantes que devem ser consideradas incluem:

- Quantas das entidades de usuário que surgem com frequência nos tweets de Tim são também seus amigos?
- Quais são as entidades mais frequentes que surgem nos tweets de Tim?
- Quem Tim retwitta com maior frequência?
- Quantos dos tweets de Tim são retwittados?
- Quantos dos tweets de Tim contêm ao menos uma entidade?

Assim como ocorre em muitas outras situações que envolvem uma fonte de dados relativamente desconhecida, uma das primeiras ações para aprender mais sobre ela é contar seus elementos. No caso de dados de tweets, contar menções de usuários, hashtags e URLs é uma ótima escolha. A próxima seção desenvolve esse tópico empregando algumas funcionalidades básicas de mapeamento/redução para contar entidades nos tweets. Ainda que entidades de tweets já existam em dados da timeline, alguns dos exemplos de código no restante deste capítulo supõem que talvez você tenha obtido seus dados a partir de outra fonte (como APIs de busca, APIs de streaming etc.). Dessa forma, fazemos o parsing das entidades para manter uma estrutura boa e consistente. Daqui a apenas alguns ajustes, você poderá confiar no Twitter para extrair para você suas entidades.

Quais entidades estão nos tweets do Tim?

Você pode utilizar a Futon do CouchDB para analisar os dados da timeline de usuário de Tim, um tweet por vez, navegando até *http://localhost:5984/_utils*. Todavia, você logo verá que isso não satisfaz suas necessidades e que seria preferível fazer um resumo geral, mostrando sobre quais assuntos Tim está twittando, em vez de uma longa lista de menções referentes a muitos elementos específicos. Basta um rápido trabalho de mapeamento/redução para termos acesso a algumas respostas preliminares, que podem ser computadas com o script do exemplo 5.4.

Exemplo 5.4 – Extração de entidades dos tweets e realização de análise de frequência (the_tweet__count_entities_in_tweets.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable
DB = sys.argv[1]
server = couchdb.Server('http://localhost:5984')
db = server[DB]
if len(sys.argv) > 2 and sys.argv[2].isdigit():
    FREQ_THRESHOLD = int(sys.argv[2])
else:
    FREQ_THRESHOLD = 3
# Mapeia entidades nos tweets aos documentos em que elas surgem
def entityCountMapper(doc):
    if not doc.get('entities'):
        import twitter_text
        def getEntities(tweet):
            # Agora, extraia várias entidades dele e construa uma
estrutura familiar
            extractor = twitter_text.Extractor(tweet['text'])
            # Note que a API de produção do Twitter contém alguns
campos adicionais no
            # hash de entidades que necessitariam de chamadas
adicionais à API para serem transformados
            entities = {}
            entities['user_mentions'] = []
            for um in
extractor.extract_mentioned_screen_names_with_indices():
```

```
                entities['user_mentions'].append(um)
            entities['hashtags'] = []
            for ht in extractor.extract_hashtags_with_indices():
                # Ajusta o nome do campo para corresponder à API
Twitter de produção
                ht['text'] = ht['hashtag']
                del ht['hashtag']
                entities['hashtags'].append(ht)
            entities['urls'] = []
            for url in extractor.extract_urls_with_indices():
                entities['urls'].append(url)
            return entities
        doc['entities'] = getEntities(doc)
    if doc['entities'].get('user_mentions'):
        for user_mention in doc['entities']['user_mentions']:
            yield ('@' + user_mention['screen_name'].lower(),
[doc['_id'], doc['id']])
    if doc['entities'].get('hashtags'):
        for hashtag in doc['entities']['hashtags']:
            yield ('#' + hashtag['text'], [doc['_id'],
doc['id']])
    if doc['entities'].get('urls'):
        for url in doc['entities']['urls']:
            yield (url['url'], [doc['_id'], doc['id']])
def summingReducer(keys, values, rereduce):
    if rereduce:
        return sum(values)
    else:
        return len(values)
view = ViewDefinition('index', 'entity_count_by_doc',
entityCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)
# Imprime uma tabela devidamente formatada. Ordenar por valor no
cliente é simples e fácil
# caso você esteja lidando com centenas ou poucos milhares de
tweets
entities_freqs = sorted([(row.key, row.value) for row in
                         db.view('index/entity_count_by_doc',
```

```
group=True)],
                          key=lambda x: x[1], reverse=True)
fields = ['Entity', 'Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for (entity, freq) in entities_freqs:
    if freq > FREQ_THRESHOLD:
        pt.add_row([entity, freq])
pt.printt()
```

Note que ainda que pudéssemos ter construído um índice de menor utilidade para computar contagens de frequência sem utilizar o parâmetro `rereduce`, a construção de um índice mais útil nos permite utilizar esse parâmetro, uma consideração importante em qualquer trabalho mais avançado de mapeamento/redução.

Uma nota sobre o rereduce

O exemplo 5.4 é o primeiro código que vemos que utiliza explicitamente o parâmetro `rereduce`. Assim, pode ser útil explicar exatamente o que está acontecendo aqui. Lembre-se de que, além de reduzir a saída para alguns mapeadores (necessariamente agrupados por chave), também é possível passarmos uma saída ao redutor para que alguns redutores façam a *redução*. Em funções usuais, como as de contagem de elementos e de computação de somas, pode ser indiferente o ponto de origem das entradas, desde que uma operação comutativa, como uma adição, seja continuamente aplicada. Todavia, em alguns casos, a diferença entre uma redução inicial e uma rereredução é significativa. É justamente nessas situações que o parâmetro `rereduce` se torna útil.

Para a função `summingReducer`, vista no exemplo 5.4, considere as seguintes saídas de exemplo do mapeador (o conteúdo em si da seção de valor de cada tupla é irrelevante):

```
[ ["@foo", [x1, x2]], ["@foo", [x3, x4]], ["@bar", [x5, x6]],
  ["@foo", [x7, x8]], ("@bar", [x9, x10]] ]
```

Por simplicidade, vamos supor que cada nó da Árvore B que armazena essas tuplas é capaz de abrigar somente dois itens de cada vez (o valor em si como implementado pelo CouchDB atinge os milhares). Durante a primeira passagem, quando `rereduce` é falso, o redutor operaria conceitualmente na entrada vista a seguir. Uma vez mais, lembre-se de que os valores são agrupados por chave:

```
# len( [ [x1, x2], [x3, x4] ] ) == 2
summingReducer(["@foo", "@foo"], [ [x1, x2], [x3, x4] ], False)

# len( [ [x7, x8] ] ) => 1
summingReducer(["@foo"], [ [x7, x8] ], False)

# len( [ [x5, x6], [x9, x10] ] ) == 2
summingReducer(["@bar", "@bar"], [ [x5, x6], [x9, x10] ], False)
```

Durante a próxima passagem pelo redutor, quando `rereduce` é `True` (uma vez que o *redutor está operando sobre saída já reduzida e necessariamente agrupada por chave*), nenhuma comparação relacionada aos valores das chaves é necessária. Uma vez que a operação `len` prévia não fez efetivamente nada além de contar o número de ocorrências de cada chave (uma entidade do tweet), passagens adicionais pelo redutor agora têm de somar esses valores para computar uma contagem final, como ilustrada pela fase de rereredução:

```
# valores da redução prévia de chaves @foo: sum([2, 1]) == 3
summingReducer(None, [ 2, 1 ], True)

# valores da redução prévia de chaves @bar: sum([2]) == 2
summingReducer(None, [ 2 ], True)
```

Uma visão geral mostra que, como passo intermediário, `len` foi utilizada primeiramente para contar o número de vezes em que uma entidade apareceu; então, em um passo final de reredução, a função `sum` computou esses resultados intermediários.

Em resumo, o script utiliza um mapeador para emitir uma tupla da forma `(entidade, [couchdb_id, tweet_id])` para cada documento, e então utiliza um redutor para contar o número de vezes em que cada entidade distinta se faz presente. Como você está provavelmente trabalhando com um conjunto relativamente pequeno de itens, sendo que já foi apresentado a alguns outros mecanismos de ordenação no capítulo 3, basta ordenar os dados no lado cliente, e aplicar um limite de frequência. Uma saída de exemplo com um limite de 15 pode ser vista na tabela 5.1, mas também como um gráfico na figura 5.1, para que você possa entender a distribuição.

Tabela 5.1 – Entidades ordenadas por frequência a partir dos tweets coletados de @timoreilly

Entidade	Frequência
#gov20	140
@OReillyMedia	124
#Ebook	89
@timoreilly	77
#ebooks	55
@slashdot	45
@jamesoreilly	41
#w2e	40
@gnat	38
@n2vip	37
@monkchips	33
#w2s	31
@pahlkadot	30
@dalepd	28
#g2e	27
#ebook	25
@ahier	24
#where20	22
@digiphile	21

Entidade	Frequência
@fredwilson	20
@brady	19
@mikeloukides	19
#pdf10	19
@nytimes	18
#fooeast	18
@andrewsavikas	17
@CodeforAmerica	16
@make	16
@pkedrosky	16
@carlmalamud	15
#make	15
#opengov	15

Então, quais assuntos estão na cabeça de Tim atualmente? Não é nenhuma surpresa que alguns de seus tópicos favoritos apareçam na lista – por exemplo, #gov20, com um número elevadíssimo de 140 menções, superando todos os outros – mas talvez até mais intrigante sejam algumas das menções menos óbvias, que possivelmente indicam relacionamentos próximos. De fato, pode ser correto presumir que Tim considera interessantes os usuários que menciona. Você até poderia inferir que ele é influenciado por esses usuários, ou que neles confia (muitos algoritmos interessantes poderiam ser criados para tentar determinar tais tipos de relacionamentos, com níveis variáveis de certeza). Ainda que computar contagens brutas, como fizemos nesta seção, seja interessante, aplicar filtragens com base em data e hora é outra possibilidade tentadora. Por exemplo, poderíamos obter insights úteis aplicando um filtro ao código de mapeamento/redução do exemplo 5.4, para que pudéssemos calcular sobre o que Tim tem falado nos últimos *N* dias, em vez de em seus últimos 3.200 tweets, que abrangem um período indeterminado de tempo. Alguns usuários do Twitter não chegam perto de 3.200 tweets, mesmo depois de muitos anos. Também não seria difícil representar tweets no SIMILE Timeline, apresentado na seção “Visualização de "eventos" de e-mails com o SIMILE Timeline” do capítulo 3, e analisá-los para compreender rapidamente sobre o que Tim tem twittado recentemente.

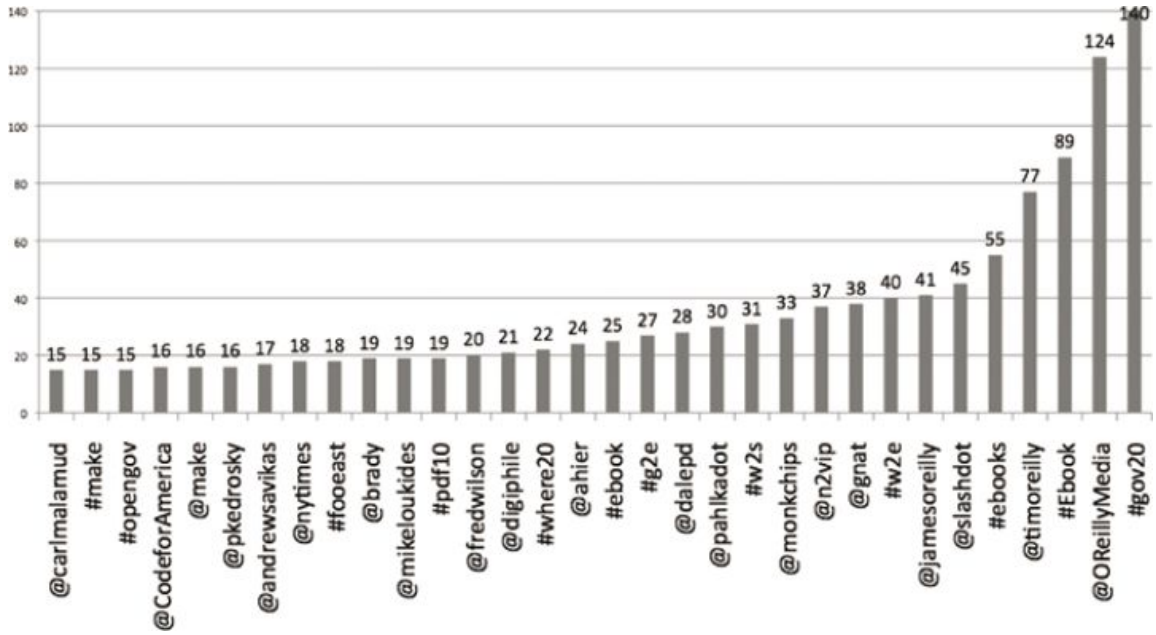


Figura 5.1 – Frequência das entidades retwittadas por @timoreilly, para uma amostra de tweets recentes.

Por acaso entidades de usuário mencionadas com frequência implicam na existência de amizades?

O exemplo 4.4 forneceu um código que demonstrou como coletar amigos e seguidores de um usuário do Twitter, e utilizou o Redis para armazenar os resultados. Presumindo que você tenha buscado os amigos e seguidores de Tim nesse código, os resultados devem estar disponíveis no Redis, e não será nada complicado computar quantas das *N* entidades de usuário mais twittadas também representam amigos de Tim. O exemplo 5.5 ilustra a utilização das informações já disponíveis no Redis desde o capítulo 4 para converter IDs de usuário em screen names; as operações de conjuntos feitas na memória pelo Redis são utilizadas para computar quais das entidades de usuário mais frequentes na timeline de usuário de Tim também representam amigos seus.

Exemplo 5.5 – Encontrando entidades @mencionadas nos tweets que também representam amigos (the_tweet___how_many_user_entities_are_friends.py)

```
# -*- coding: utf-8 -*-
import json
import redis
import couchdb
import sys
```

```
from twitter__util import getRedisIdByScreenName
from twitter__util import getRedisIdByUserId

SCREEN_NAME = sys.argv[1]
THRESHOLD = int(sys.argv[2])

# Conecta-se utilizando as configurações-padrão para localhost

r = redis.Redis()

# Processa screen_names para os amigos

friend_ids = r.smembers(getRedisIdByScreenName(SCREEN_NAME,
'friend_ids'))
friend_screen_names = []
for friend_id in friend_ids:
    try:
        friend_screen_names.append(json.loads(r.get(getRedisIdByU
serId(friend_id,
                                    'info.json')))['screen_name'])
    except TypeError, e:
        continue # não disponível localmente no Redis - faz uma
consulta, ou pula esse processo

# Acessa a lista de tuplas (entidade, frequência) do CouchDB

server = couchdb.Server('http://localhost:5984')
db = server['tweets-user-timeline-' + SCREEN_NAME]

entities_freqs = sorted([(row.key, row.value) for row in
                         db.view('index/entity_count_by_doc',
group=True)],
                         key=lambda x: x[1])

# Mantém apenas entidades de usuário com frequências
insuficientes

user_entities = [(ef[0])[1:] for ef in entities_freqs if ef[0][0]
== '@'
                 and ef[1] >= THRESHOLD]

# Faça uma comparação de conjuntos

entities_who_are_friends = \
    set(user_entities).intersection(set(friend_screen_names))

entities_who_are_not_friends = \
    set(user_entities).difference(entities_who_are_friends)

print 'Number of user entities in tweets: %s' %
(len(user_entities), )
print 'Number of user entities in tweets who are friends: %s' \
    % (len(entities_who_are_friends), )
for e in entities_who_are_friends:
```

```
    print '\t' + e
print 'Number of user entities in tweets who are not friends: %s'
\
    % (len(entities_who_are_not_friends), )
for e in entities_who_are_not_friends:
    print '\t' + e
```

A saída, quando utilizamos um limite de frequência de 15 (Exemplo 5.6), é previsível, mas traz à luz algumas considerações importantes.

Exemplo 5.6 – Exemplo de saída do exemplo 5.5, exibindo entidades @mencionadas nos tweets que também representam amigos de @timoreilly

```
Number of user entities in tweets: 20
Number of user entities in tweets who are friends: 18
    ahier
    pkedrosky
    CodeforAmerica
    nytimes
    brady
    carlmalamud
    pahlkadot
    make
    jamesoreilly
    andrewsavikas
    gnat
    slashdot
    OReillyMedia
    dalepd
    mikeloukides
    monkchips
    fredwilson
    digiphile
Number of user entities in tweets who are not friends: 2
    n2vip
    timoreilly
```

No total, há 20 entidades de usuário que excederam um limite de frequência de 15, e descobrimos que 18 delas são amigos. Uma vez que a maioria das pessoas que surge nesses tweets também é formada por amigos de Tim, é seguro dizer que há um forte relacionamento de confiança entre ele e esses indivíduos. Faça uma pausa e compare esta lista com os resultados de nossos exercícios na seção "Construção de grafos de amizade", do capítulo 4. Também pode ser interessante notar que o

próprio Tim surge como uma de suas entidades mais twittadas, assim como outro indivíduo, *@n2vip*. Talvez seja útil analisar mais atentamente o contexto dos tweets que envolvem *@n2vip*. Por acaso você tem alguma teoria que explique por que há tantas menções a esse usuário, sem que ele seja um amigo de Tim? Vamos descobrir.

Depois dos exercícios feitos no capítulo 3, você já sabe como é rápido e fácil aplicar as capacidades de indexação de texto completo da Lucene a um banco de dados do CouchDB. Adaptando e estendendo minimamente o exemplo 3.12, podemos nos concentrar nos tweets que mencionam *@n2vip*. O exemplo 5.7 demonstra como isso pode ser feito.

Exemplo 5.7 – Uso do couchdb-lucene para consultar dados de tweets (the_tweet__couchdb_lucene.py)

```
# -*- coding: utf-8 -*-
import sys
import httplib
from urllib import quote
import json
import couchdb
DB = sys.argv[1]
QUERY = sys.argv[2]
# O corpo de um documento de design em JavaScript que criaremos
dd = \
    {'fulltext': {'by_text': {'index': '''function(doc) {
                            var ret=new Document();
                            ret.add(doc.text);
                            return ret
                        }'''}}}
try:
    server = couchdb.Server('http://localhost:5984')
    db = server[DB]
except couchdb.http.ResourceNotFound, e:
    print """CouchDB database '%s' not found.
Please check that the database exists and try again.""" % DB
    sys.exit(1)
try:
    conn = httplib.HTTPConnection('localhost', 5984)
    conn.request('GET', '/%s/_design/lucene' % (DB, ))
    response = conn.getresponse()
```

```
finally:
    conn.close()
# Caso o documento de design não exista, cria um que será
identificado
# como "_design/lucene". O equivalente do seguinte em um
terminal:
# $ curl -X PUT http://localhost:5984/DB/_design/lucene -d
@dd.json
if response.status == 404:
    try:
        conn = httplib.HTTPConnection('localhost', 5984)
        conn.request('PUT', '/%s/_design/lucene' % (DB, ),
json.dumps(dd))
        response = conn.getresponse()
        if response.status != 201:
            print 'Unable to create design document: %s %s' %
(response.status,
                  response.reason)
            sys.exit(1)
    finally:
       conn.close()
# A consulta ao documento de design é feita como de costume,
exceto pela referência
# ao manipulador HTTP _fti do couchdb-lucene
# $ curl http://localhost:5984/DB/_fti/_design/lucene/by_subject?
q=QUERY
try:
    conn.request('GET', '/%s/_fti/_design/lucene/by_text?q=%s' %
(DB, quote(QUERY)))
    response = conn.getresponse()
    if response.status == 200:
        response_body = json.loads(response.read())
    else:
        print 'An error occurred fetching the response: %s %s' \
         % (response.status, response.reason)
        print 'Make sure your couchdb-lucene server is running.'
        sys.exit(1)
finally:
    conn.close()
doc_ids = [row['id'] for row in response_body['rows']]
# acessa os tweets do CouchDB e extrai o texto para exibição
```

```
tweets = [db.get(doc_id)['text'] for doc_id in doc_ids]
for tweet in tweets:
    print tweet
    print
```

Uma saída abreviada do script (Exemplo 5.8) explica por que *@n2vip* surge em tantos tweets de *@timoreilly*: ambos estavam engajados em várias conversas pelo Twitter.

Exemplo 5.8 – Saída de exemplo do exemplo 5.7

```
@n2vip Thanks. Great stuff. Passing on to the ebook team.
@n2vip I suggested it myself the other day before reading this
note.
RT @n2vip Check this out if you really want to get your
'#Churchill on', a ...
@n2vip Remember a revolution that began with a handful of farmers
and tradesmen ...
@n2vip Good suggestion re having free sample chapters as ebooks,
not just pdfs...
@n2vip I got those statistics by picking my name off the
influencer list in ...
RT @n2vip An informative, non-partisan FAQ regarding Health Care
Reform at ...
@n2vip Don't know anyone who is advocating that. FWIW, it was Rs
who turned ...
@n2vip No, I don't. But a lot of the people arguing against
renewables seem ...
@n2vip You've obviously never read an ebook on the iPhone. It's a
great reading ...
@n2vip I wasn't suggesting that insurance was the strange world,
just that you ...
@n2vip In high tech, there is competition from immigrant workers.
Yet these two ...
@n2vip How right you are. We really don't do a good job teaching
people ...
@n2vip The climategate stuff is indeed disturbing. But I still
hold by what ...
@n2vip FWIW, I usually do follow links, so do include them if
appropriate. Thanks.
@n2vip I don't mind substantive disagreement - e.g. with pointers
to real info ...
@n2vip Totally agree that ownership can help. But you need to
understand why ...
@n2vip Maybe not completely extinct, but certainly economically
```

```
extinct. E.g. ...
@n2vip I wasn't aware that it was part of a partisan agenda. Too
bad, because ...
RT @n2vip if only interesed in his 'Finest Hour' speech, try this
- a ...
@n2vip They matter a lot. I was also struck by that story this
morning. Oil ...
@n2vip I understand that. I guess "don't rob MYsocialized
medicine to fund ...
RT @n2vip Electronic medical record efforts pushing private
practice docs to ...
@n2vip I think cubesail can be deployed in 2 ways: to force
quicker re-entry ...
RT @ggreenwald Wolf Blitzer has major epiphany on public opinion
and HCR that ...
```

Inclusão da outra metade da conversa

Parece que as muitas menções feitas a *@n2vip* estão relacionadas ao fato de que ele participou de algumas conversas com Tim. Depois de analisarmos rapidamente esses comentários, notamos que essas discussões parecem ter sido de natureza política (o que não nos surpreende, dada a elevada frequência com que surge a hashtag #gov20, como vimos em um exercício anterior deste capítulo). Um exercício interessante que poderíamos fazer na sequência é aperfeiçoar o exemplo 5.7 para extrair os campos `in_reply_to_status_id` dos tweets e montar uma versão mais completa e legível das conversas.

Uma técnica para reconstrução da thread que minimiza o número de chamadas necessárias à API envolve simplesmente buscar todos os tweets de *@n2vip* com IDs maiores que o tweet de ID mínimo da thread de interesse, em vez de coletar IDs de status individuais um por um. O exemplo 5.9 mostra como isso poderia ser feito.

Exemplo 5.9 – Reconstrução de threads de discussão em tweets (the_tweet__reassemble_discussion_thread.py)

```
# -*- coding: utf-8 -*-

import sys
import httplib
from urllib import quote
import json
import couchdb
```

```
from twitter__login import login
from twitter__util import makeTwitterRequest

DB = sys.argv[1]
USER = sys.argv[2]

try:
    server = couchdb.Server('http://localhost:5984')
    db = server[DB]
except couchdb.http.ResourceNotFound, e:
    print >> sys.stderr, """CouchDB database '%s' not found.
Please check that the database exists and try again.""" % DB
    sys.exit(1)

# consulte por termo

try:
    conn = httplib.HTTPConnection('localhost', 5984)
    conn.request('GET', '/%s/_fti/_design/lucene/by_text?q=%s' %
(DB, quote(USER)))
    response = conn.getresponse()
    if response.status == 200:
        response_body = json.loads(response.read())
    else:
        print >> sys.stderr, 'An error occurred fetching the
response: %s %s' \
            % (response.status, response.reason)
        sys.exit(1)
finally:
   conn.close()

doc_ids = [row['id'] for row in response_body['rows']]

# acessa os tweets do CouchDB

tweets = [db.get(doc_id) for doc_id in doc_ids]

# minera os campos in_reply_to_status_id e busca os tweets como
uma solicitação em lote

conversation = sorted([(tweet['_id'],
int(tweet['in_reply_to_status_id']))
                       for tweet in tweets if
tweet['in_reply_to_status_id']
                       is not None], key=lambda x: x[1])
min_conversation_id = min([int(i[1]) for i in conversation if
i[1] is not None])
max_conversation_id = max([int(i[1]) for i in conversation if
i[1] is not None])

# Acessa os tweets de outro usuário utilizando a API da timeline
```

```
do usuário
# para minimizar o gasto da API...
t = login()
reply_tweets = []
results = []
page = 1
while True:
    results = makeTwitterRequest(t,
        t.statuses.user_timeline,
        count=200,
        # De acordo com
<http://dev.twitter.com/doc/get/statuses/user_timeline>, algumas
        # ressalvas se aplicam ao id mais antigo que você pode
buscar utilizando #since_id"
        since_id=min_conversation_id,
        max_id=max_conversation_id,
        skip_users='true',
        screen_name=USER,
        page=page)
    reply_tweets += results
    page += 1
    if len(results) == 0:
        break
# Durante testes, observamos que alguns tweets podem não ser
retornados, ou possivelmente
# até mesmo retornar com valores nulos de id -- talvez uma falha
temporária. A seguir, veja
# uma possível solução:
missing_tweets = []
for (doc_id, in_reply_to_id) in conversation:
    try:
        print [rt for rt in reply_tweets if rt['id'] ==
in_reply_to_id][0]['text']
    except Exception, e:
        print >> sys.stderr, 'Refetching <<tweet %s>>' %
(in_reply_to_id, )
        results = makeTwitterRequest(t, t.statuses.show,
id=in_reply_to_id)
        print results['text']
    # Estes tweets já estão disponíveis
    print db.get(doc_id)['text']
    print
```

Grande parte desse código deve lhe parecer familiar. Com a habilidade de autenticar dados na API do Twitter, armazenar e recuperar dados localmente, e buscar dados remotos a partir do Twitter, muito pode ser feito com uma quantidade mínima de "lógica de negócio". Uma saída abreviada de exemplo para este script, apresentada no exemplo 5.10, pode ser vista a seguir, mostrando o fluxo da discussão entre *@timoreilly* e *@n2vip*.

Exemplo 5.10 – Exemplo de saída do exemplo 5.9

```
Question: If all Ins. Co. suddenly became non-profit and approved
ALL Dr. ...
@n2vip Don't know anyone who is advocating that. FWIW, it was Rs
who turned ...

@timoreilly RT @ggreenwald Wolf Blitzer has major epiphany on
public opinion ...
RT @ggreenwald Wolf Blitzer has major epiphany on public opinion
and HCR that ...

@timoreilly RE: Cubesail - I don't get it, does the sail collect
loose trash ...
@n2vip I think cubesail can be deployed in 2 ways: to force
quicker re-entry ...

@timoreilly How are you finding % of your RT have links? What
service did you ...
@n2vip I got those statistics by picking my name off the
influencer list in ...

@timoreilly a more fleshed-out e-book 'teaser' chapter idea here:
http://bit.ly/aML6eH
@n2vip Thanks. Great stuff. Passing on to the ebook team.

@timoreilly Tim, #HCR law cuts Medicare payments to fund, in
part, broader ...
@n2vip I understand that. I guess "don't rob MYsocialized
medicine to fund ...

@timoreilly RE: Auto Immune - a "revolution" that is measured by
a hand-full ...
@n2vip Remember a revolution that began with a handful of farmers
and tradesmen ...

Do oil spills in Africa not matter? http://bit.ly/bKqv01
@jaketapper @yunjid ...
@n2vip They matter a lot. I was also struck by that story this
morning. Oil ...
```

Quem Tim retwitta mais frequentemente?

Assim como o (não tão antigo) adágio que diz que um retweet é a maior forma de elogio, poderíamos reformular a questão "Quem Tim retwitta mais frequentemente?" perguntando "Quem Tim elogia mais frequentemente?", ou ainda, como não faria sentido ele retwittar conteúdo que não considera interessante, poderíamos perguntar "Quem Tim considera que está falando sobre *os assuntos mais importantes*?" Uma hipótese razoável seria pensar que muitas das entidades de usuário que surgem na timeline pública de Tim estejam relacionadas aos autores dos tweets que Tim está retwittando. Vamos explorar um pouco mais essa noção e calcular quantos dos tweets de Tim são retweets e, desses retweets, qual autor é retwittado com maior frequência. Há muitos métodos de API que podemos utilizar para coletar retweets, mas, como já temos alguns milhares de tweets à disposição, obtidos em um exercício prévio, vamos analisá-los para aproveitar ao máximo nossas chamadas à API.

Há alguns padrões básicos para um retweet:

- RT *@usuário* Mary tinha um carneirinho
- Mary tinha um carneirinho (via *@usuário*)

Em ambos os casos, o significado é simples: alguém está dando crédito a *@usuário* por determinada informação. O exemplo 5.11 fornece um programa que extrai o número de vezes em que Tim retwittou outros usuários do Twitter, aplicando um verificador de expressão regular como parte de uma simples rotina de mapeamento/redução.

Note que, mesmo durante a redação deste livro, a API do Twitter segue em constante evolução. O exemplo 5.11 faz uma contagem de retweets extraindo indicações como "RT" do próprio texto do tweet. Quando você estiver lendo este texto, pode muito bem já ter sido criada uma forma mais eficiente de computar o número de vezes em que um usuário retwittou outro, para um intervalo específico de tempo.

Exemplo 5.11 – Contagem do número de vezes em que usuários foram retwittados por alguém (the_tweet__count_retweets_of_other_users.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable
```

```
DB = sys.argv[1]
try:
    server = couchdb.Server('http://localhost:5984')
    db = server[DB]
except couchdb.http.ResourceNotFound, e:
    print """CouchDB database '%s' not found.
Please check that the database exists and try again.""" % DB
    sys.exit(1)
if len(sys.argv) > 2 and sys.argv[2].isdigit():
    FREQ_THRESHOLD = int(sys.argv[2])
else:
    FREQ_THRESHOLD = 3
# Mapeia entidades nos tweets aos documentos em que elas surgem
def entityCountMapper(doc):
    if doc.get('text'):
        import re
        m = re.search(r"(RT|via)((?:\b\W*@\w+)+)", doc['text'])
        if m:
            entities = m.groups()[1].split()
            for entity in entities:
                yield (entity.lower(), [doc['_id'], doc['id']])
         else:
             yield ('@', [doc['_id'], doc['id']])
def summingReducer(keys, values, rereduce):
    if rereduce:
        return sum(values)
    else:
        return len(values)
view = ViewDefinition('index', 'retweet_entity_count_by_doc',
entityCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)
# Ordenar por valor no cliente é simples e fácil
# caso você esteja lidando com centenas ou poucos milhares de
tweets
entities_freqs = sorted([(row.key, row.value) for row in
                         db.view('index/retweet_entity_count_by_do
c',
                         group=True)], key=lambda x: x[1],
reverse=True)
```

```
fields = ['Entity', 'Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for (entity, freq) in entities_freqs:
    if freq > FREQ_THRESHOLD and entity != '@':
        pt.add_row([entity, freq])
pt.printt()
```

Se você pensa que os resultados terão visual praticamente idêntico ao das contagens brutas de entidades computadas na tabela 5.1, ficará um pouco surpreso. O código na tabela 5.2 também utiliza um limite de 15, para justapor a diferença.

Tabela 5.2 – Entidades mais frequentes que surgem nos retweets feitos por @timoreilly; as colunas adicionais ilustram a normalização dessas contagens

Entidade	Número de vezes em que @usuário foi retwittado por @timoreilly	Número total de tweets por @usuário	Pontuação de retweets normalizada
@monkchips	30	33215	0,000903206
@ahier	18	14849	0,001212203
@slashdot	41	22081	0,0018568
@gnat	28	11322	0,002473061
@mikeloukides	15	2926	0,005126452
@pahlkadot	16	3109	0,005146349
@oreillymedia	97	6623	0,014645931
@jamesoreilly	34	4439	0,007659383
@dalepd	16	1589	0,010069226

Então, quem é a pessoa que Tim mais retwitta/elogia? Bem, não deve surpreendê-lo que sua empresa e os profissionais relacionados a ela preencham a maior parte da lista, com *@oreillymedia* surgindo no topo da classificação. Uma inspeção visual da contagem de retweets e de entidades brutas mostra uma correlação óbvia. Lembre-se, entretanto, de que os resultados mostrados na coluna "Número de vezes em que *@usuário* foi retwittado por *@timoreilly*" na tabela 5.2 não estão normalizados. Por exemplo, *@dalepd* foi retwittado muito menos que *@oreillymedia*, mas uma breve inspeção do campo `statuses_count` incorporado aos objetos de usuário dos tweets (dentre outros lugares nos resultados da API) revela que *@dalepd* twitta muito mais raramente do que *@oreillymedia*. De fato, se você normalizar os dados, dividindo o número de retweets pelo número total de tweets de cada usuário, verá que

@dalepd fica em segundo lugar e muito próximo de *@oreillymedia* – o que significa que Tim retwitta mais de seus tweets do que de qualquer outro usuário na tabela, exceto por *@oreillymedia*, mesmo que essa conclusão não fique tão evidente se você simplesmente ordenar os dados pela frequência bruta de retweets. A figura 5.2 ilustra essa análise como um gráfico de bolha. Também seria interessante explorar um pouco mais esses dados e descobrir por que alguns usuários não são retwittados na mesma frequência com que são mencionados.

Uma vez que não é tão difícil determinar quem Tim retwitta com mais frequência, que tal invertermos o pergunta: quem mais retwitta Tim? Responder a essa questão de forma agregada, sem uma população-alvo concentrada, seria um pouco difícil, se não impossível, dadas as limitações da API do Twitter, e posto que Tim tem aproximadamente 1.500.000 seguidores. Entretanto, a tarefa ficará um pouco mais fácil se você restringir seu foco a uma população-alvo de tamanho razoável. A próxima seção investiga algumas opções.

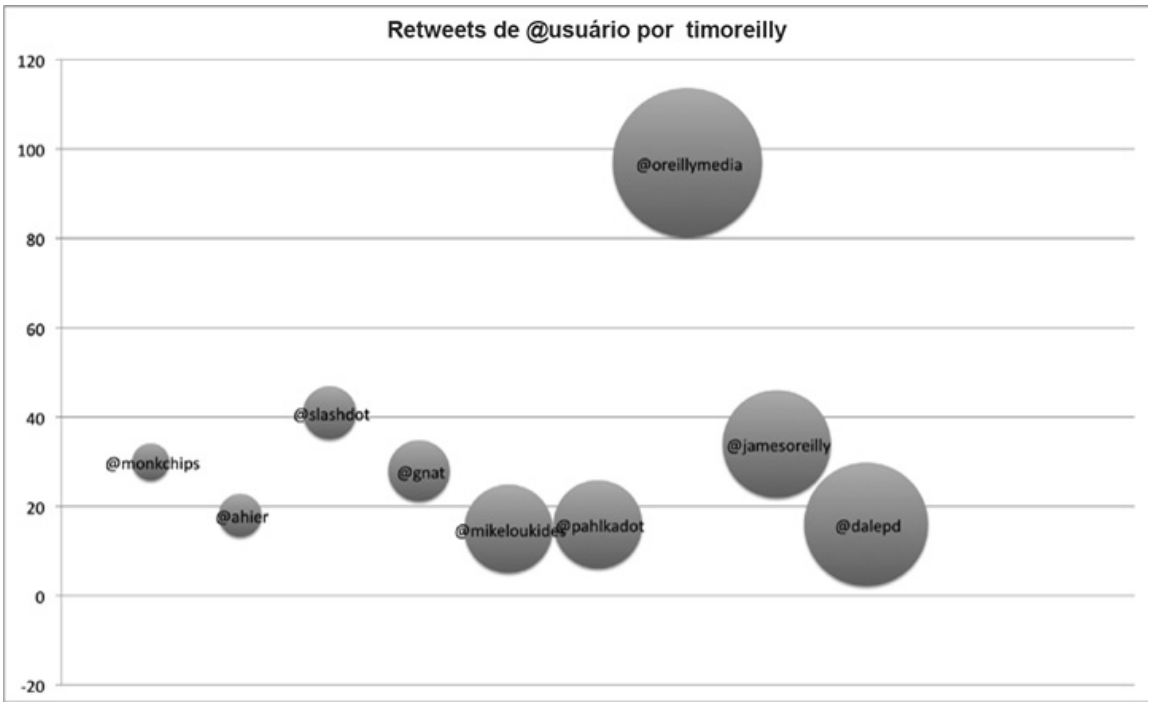


Figura 5.2 – Note que o eixo y desse gráfico de bolha representa o número bruto de vezes em que um usuário foi retwittado; a área da bolha representa uma pontuação normalizada para o número de vezes em que um usuário foi retwittado quando comparado ao número total de vezes que ele twittou.

Qual a influência de Tim?

Perguntar quantos de seus tweets são retwittados é apenas outra forma de medir sua influência. Se você twitta muito e ninguém retwitta você, podemos dizer que sua influência é fraca – a menos como usuário do Twitter. De fato, seria um tanto paradoxal verificar que você tem muitos seguidores, mas poucos retweets, uma vez que você geralmente obtém seguidores e retweets pela mesma razão: por ser interessante e influente.

Uma métrica básica, que pode ser calculada de forma simples e econômica, é a razão de tweets para retweets. Uma razão de 1 significa que todo tweet individual de sua autoria foi retwittado, indicando que sua influência é forte – potencialmente com efeitos de segundo e terceiro nível que atingiriam milhões de usuários únicos (literalmente) – enquanto valores próximos a 0 revelariam uma influência mais fraca. É claro que, dada a natureza do Twitter, é altamente improvável que qualquer usuário humano tenha uma razão de tweet-para-retweet de valor 1, uma vez que a própria mecânica das conversas (@respostas) faz com que esse valor diminua. O Twitter expõe o recurso `statuses/retweets_of_me`, fornecendo ao usuário autenticado informações referentes a quantos de seus tweets foram retwittados. Entretanto, não temos acesso a essa API para analisar os retweets de Tim, por isso uma abordagem alternativa se faz necessária.

O exemplo 5.12 aproveita o campo `retweet_count` de um tweet para computar o número de tweets que foram retwittados um determinado número de vezes, como parte do que agora parece uma combinação trivial de mapeamento/redução. Uma saída de exemplo formatada como um gráfico segue o exemplo.

Exemplo 5.12 – Encontrando os tweets retwittados com maior frequência (the_tweet__count_retweets_by_others.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
from prettytable import PrettyTable
from twitter__util import pp
DB = sys.argv[1]
try:
    server = couchdb.Server('http://localhost:5984')
    db = server[DB]
```

```
except couchdb.http.ResourceNotFound, e:
    print """CouchDB database '%s' not found.
Please check that the database exists and try again.""" % DB
    sys.exit(1)

# Mapeia entidades nos tweets aos documentos em que elas surgem
def retweetCountMapper(doc):
    if doc.get('id') and doc.get('text'):
        yield (doc['retweet_count'], 1)

def summingReducer(keys, values, rereduce):
    return sum(values)

view = ViewDefinition('index', 'retweets_by_id',
retweetCountMapper,
                      reduce_fun=summingReducer,
language='python')

view.sync(db)

fields = ['Num Tweets', 'Retweet Count']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]

retweet_total, num_tweets, num_zero_retweets = 0, 0, 0
for (k,v) in sorted([(row.key, row.value) for row in
                      db.view('index/retweets_by_id', group=True)
                      if row.key is not None],
                    key=lambda x: x[0], reverse=True):
    pt.add_row([k, v])
    if k == "100+":
        retweet_total += 100*v
    elif k == 0:
        num_zero_retweets += v
    else:
        retweet_total += k*v
    num_tweets += v

pt.printt()

print '\n%s of %s authored tweets were retweeted at least once' %
\
    (pp(num_tweets - num_zero_retweets), pp(num_tweets),)
print '\t(%s tweet/retweet ratio)\n' % (1.0*(num_tweets -
num_zero_retweets)/num_tweets,)

print 'Those %s authored tweets generated %s retweets' %
(pp(num_tweets), pp(retweet_total),)
```

A figura 5.3 mostra resultados do exemplo 5.12, formatados em um gráfico

mais compacto que utiliza uma escala logarítmica para "achatar" o eixo *y*. Valores ao longo do eixo *x* correspondem ao número de tweets para um dado valor de retweet indicado pelo eixo *y*. A "área total sob a curva" é um pouco maior que 3.000 – o número total de tweets analisados. Por exemplo, cerca de 533 dos tweets de Tim não chegaram a ser retwittados, como denota a coluna mais à esquerda. Além disso, 50 de seus tweets foram retwittados 50 vezes, e mais de 60 foram retwittados mais de 100 vezes[4], como denota a coluna à direita.

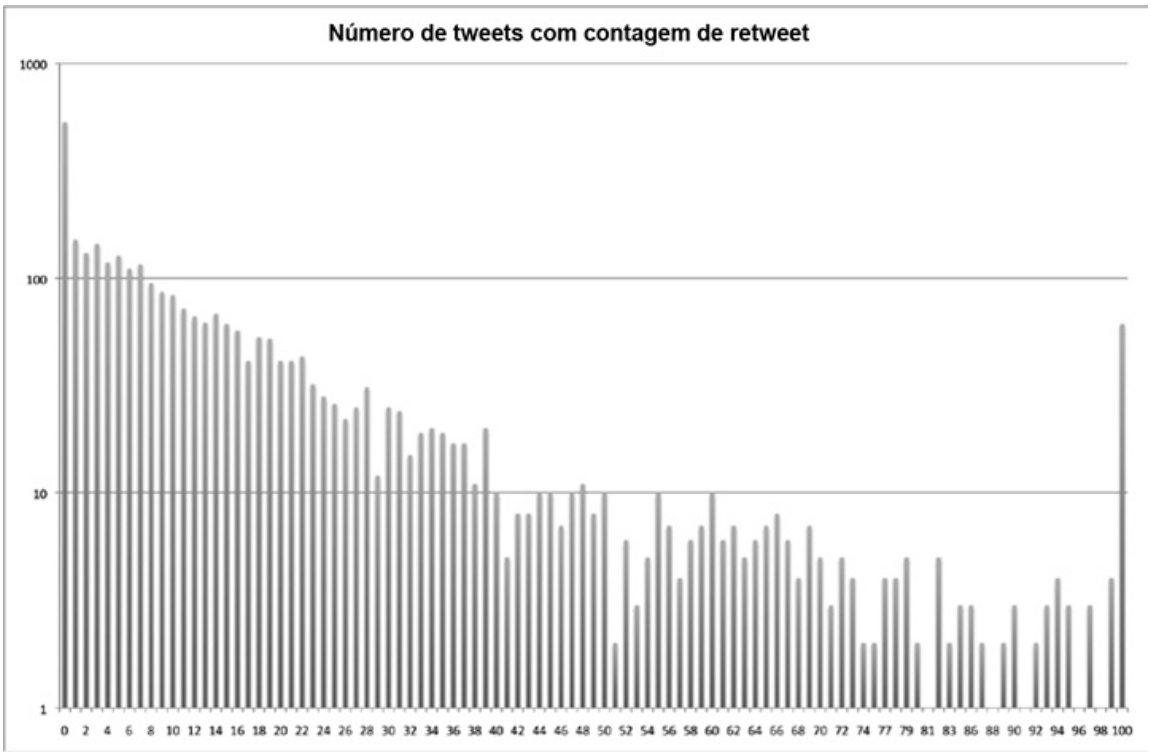


Figura 5.3 – Exemplo de resultados do exemplo 5.12.

Essa distribuição não é tão surpreendente, pois apresenta uma tendência que *geralmente* acompanha uma lei de potência e mostra que um número considerável dos tweets de Tim teve desempenho viral, com tweets que foram retwittados potencialmente muitas centenas de vezes. O panorama geral é que dos mais de 3.000 tweets gerais, 2.536 foram retwittados ao menos uma vez (uma razão de cerca de 0,80) e geraram mais de 50.000 retweets no total (um fator de cerca de 16).

Para dizer o mínimo, tais números confirmam Tim como um maven influente da área da informação.

Quantos dos tweets de Tim contêm hashtags?

Parece razoável dizer que tweets contendo entidades hashtag são inerentemente mais valiosos do que aqueles em que isso não ocorre, uma vez que alguém se dispôs a deliberadamente incorporar informações agregadas a esses tweets, essencialmente os transformando em informação semiestruturada. Por exemplo, parece razoável presumir que alguém com uma média de 2 ou mais hashtags por tweet está muito interessado em agregar conhecimento e ciente do poder da informação, enquanto alguém com uma média de 0,1 hashtags por tweet provavelmente não se enquadra nessas características.

O que é uma folksonomia?

Um aspecto fundamental da inteligência humana é o desejo de classificar elementos e disso derivar uma hierarquia na qual cada um deles "faça parte", ou "seja filho", de um elemento-pai, localizado um nível acima na hierarquia. Deixando de lado debates filosóficos sobre a diferença entre taxonomia e ontologia, podemos dizer que uma *taxonomia* é essencialmente uma estrutura hierárquica que classifica elementos em categorias pai/filho. O termo *folksonomia* (*folksonomy*, *http://en.wikipedia.org/wiki/Folksonomy*) foi estabelecido por volta de 2004 como uma forma de descrever o universo das iniciativas de classificação e indexação social que emergem em vários ecossistemas da web, além de ser um jogo de palavras que mistura as expressões "folk" e "taxonomia". Assim, em essência, folksonomia é apenas uma forma pomposa de descrever o universo descentralizado das tags; elas emergem como um mecanismo de *inteligência coletiva* por permitirem a classificação de conteúdo com rótulos.

Computar o número médio de hashtags por tweet deve ser agora algo muito fácil para você. Reciclaremos código já utilizado para computar o número total de hashtags em uma fase de mapeamento/redução e o número total de tweets em outra fase desse tipo. Depois, dividiremos esses dois números (Exemplo 5.13).

Exemplo 5.13 – Contagem de entidades hashtag nos tweets (the_tweet__avg_hashtags_per_tweet.py)

```
# -*- coding: utf-8 -*-
import sys
import couchdb
from couchdb.design import ViewDefinition
DB = sys.argv[1]
try:
    server = couchdb.Server('http://localhost:5984')
    db = server[DB]
except couchdb.http.ResourceNotFound, e:
    print """CouchDB database '%s' not found.
Please check that the database exists and try again.""" % DB
```

```
    sys.exit(1)
# Emite o número de hashtags em um documento
def entityCountMapper(doc):
    if not doc.get('entities'):
        import twitter_text
        def getEntities(tweet):
            # Agora, extraia diversas entidades dele e construa
uma estrutura familiar
            extractor = twitter_text.Extractor(tweet['text'])
            # Note que a API de produção do Twitter contém alguns
campos adicionais no
            # hash de entidades que necessitariam de chamadas
adicionais à API para serem transformados
            entities = {}
            entities['user_mentions'] = []
            for um in
extractor.extract_mentioned_screen_names_with_indices():
                entities['user_mentions'].append(um)
            entities['hashtags'] = []
            for ht in extractor.extract_hashtags_with_indices():
                # Ajusta o nome do campo para corresponder à API
Twitter de produção
                ht['text'] = ht['hashtag']
                del ht['hashtag']
                entities['hashtags'].append(ht)
            entities['urls'] = []
            for url in extractor.extract_urls_with_indices():
                entities['urls'].append(url)
            return entities
        doc['entities'] = getEntities(doc)
    if doc['entities'].get('hashtags'):
        yield (None, len(doc['entities']['hashtags']))
def summingReducer(keys, values, rereduce):
    return sum(values)
view = ViewDefinition('index', 'count_hashtags',
entityCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)
```

```
num_hashtags = [row for row in db.view('index/count_hashtags')]
[0].value

# Agora, conta o número total de tweets que não são respostas
diretas

def entityCountMapper(doc):
    if doc.get('text')[0] == '@':
        yield (None, 0)
    else:
        yield (None, 1)

view = ViewDefinition('index', 'num_docs', entityCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)

num_docs = [row for row in db.view('index/num_docs')][0].value

# Por fim, computa a média

print 'Avg number of hashtags per tweet for %s: %s' % \
        (DB.split('-')[-1], 1.0 * num_hashtags / num_docs,)
```

Para um lote recente de tweets que buscamos há pouco, executar este script revela que Tim tem uma média aproximada de 0,5 hashtag por tweet, no caso de tweets que não são uma resposta direta a alguém. Em outras palavras, ele inclui uma hashtag em aproximadamente metade de seus tweets. Para alguém que twitta regularmente, incluir uma hashtag tantas vezes representa uma contribuição significativa ao índice geral de busca do Twitter e à sempre-crescente folksonomia. Como exercício complementar, talvez seja interessante computar o número médio de entidades hiperlink presentes por tweet, ou até mesmo seguir os links e tentar descobrir novas informações sobre os interesses de Tim, inspecionando o título e o conteúdo das páginas web cujos links são fornecidos. Nos capítulos futuros, especialmente nos capítulos 7 e 8, aprenderemos mais sobre mineração de texto, uma habilidade essencial para analisar páginas web.

Justaposição de redes sociais latentes (ou #JustinBieber versus #TeaParty)

Um dos aspectos mais fascinantes em mineração de dados é o fato de que ela permite a descoberta de novos conhecimentos, partindo de informações existentes. Muito pode ser dito a respeito do velho adágio

que afirma que "conhecimento é poder", especialmente sobre sua validade em uma era na qual a quantidade de informação disponível cresce constantemente e sem sinais de declínio. Como um exercício interessante, vejamos o que podemos descobrir sobre algumas redes sociais latentes do oceano de dados do Twitter. Em nossa abordagem, coletaremos alguns dados concentrados em dois ou mais tópicos de forma específica, buscando uma hashtag em particular, e então aplicaremos algumas das métricas que codificamos na seção prévia (em que analisamos os tweets de Tim) para verificar a similaridade entre as redes.

Como não existe "pergunta burra", vamos seguir o espírito do famoso economista Steven D. Levitt[5] e perguntar: "O que #TeaParty[6] e #JustinBieber têm em comum?" [7]

O exemplo 5.14 fornece um mecanismo simples para coletar aproximadamente 1.500 dos tweets mais recentes (o máximo atualmente retornado pela API de busca) sobre um tópico específico, e armazená-los no CouchDB. Assim como outras listagens vistas antes neste capítulo, ele inclui uma lógica simples de mapeamento/redução para atualizar de forma incremental os tweets, caso você queira executar esta operação por um período mais prolongado e coletar um lote maior de dados do que os que a API de busca fornece. Talvez seja interessante investigar a API de streaming (*http://dev.twitter.com/pages/streaming_api*) para esse tipo de tarefa.

Exemplo 5.14 – Coleta de dados para uma dada consulta (the_tweet__search.py)

```
# -*- coding: utf-8 -*-

import sys
import twitter
import couchdb
from couchdb.design import ViewDefinition
from twitter__util import makeTwitterRequest

SEARCH_TERM = sys.argv[1]
MAX_PAGES = 15

KW = {
    'domain': 'search.twitter.com',
    'count': 200,
    'rpp': 100,
    'q': SEARCH_TERM,
    }
```

```
server = couchdb.Server('http://localhost:5984')
DB = 'search-%s' % (SEARCH_TERM.lower().replace('#',
'').replace('@', ''), )
try:
    db = server.create(DB)
except couchdb.http.PreconditionFailed, e:
    # já existe, por isso acrescente a ele, e esteja ciente
quanto a duplicatas
    db = server[DB]
t = twitter.Twitter(domain='search.twitter.com')
for page in range(1, 16):
    KW['page'] = page
    tweets = makeTwitterRequest(t, t.search, **KW)
    db.update(tweets['results'], all_or_nothing=True)
    if len(tweets['results']) == 0:
        break
    print 'Fetched %i tweets' % len(tweets['results'])
```

As seções seguintes têm como base aproximadamente 3.000 tweets por tópico e presumem que você executou o script e coletou dados sobre #TeaParty e #JustinBieber (ou outros tópicos que lhe interessem).

Dependendo das preferências de seu terminal, pode ser necessário escapar certos caracteres (como o símbolo de cerquilha) em razão da forma como eles são interpretados por seu shell. Por exemplo, em Bash, você teria de escapar uma consulta de hashtag para #TeaParty como \#TeaParty, para garantir que o shell interpretasse o símbolo de cerquilha como parte do termo de consulta, e não como o início de um comentário.

Quais entidades coocorrem com maior frequência nos tweets de #JustinBieber e #TeaParty?

Uma das formas mais simples, e talvez mais eficientes, de caracterizar dois públicos diferentes é analisar as entidades que surgem em um pool agregado de tweets. Além de fornecer-lhe uma boa noção dos tópicos sobre os quais cada público conversa, dados como estes também permitem que você compare as entidades que coocorrem, alcançando uma métrica rudimentar de similaridade. O exemplo 5.4 já forneceu a lógica de que necessitamos para realizar uma primeira passagem pela análise das entidades. Presumindo que você tenha executado consultas de busca para #JustinBieber e #TeaParty, você deve ter dois bancos de dados do CouchDB, "search-justinbieber" e "search-teaparty", que podem ser utilizados para produzir seus próprios resultados. Resultados de exemplo

para cada hashtag com uma frequência de entidade maior que 20 podem ser vistos nas tabelas 5.3 e 5.4; a figura 5.4 exibe um gráfico que transmite as distribuições de frequência subjacentes para essas tabelas. Como deve conter tais extremos, o eixo *y* foi ajustado para ser uma escala logarítmica, facilitando a leitura de seus valores.

Tabela 5.3 – Entidades que surgem com maior frequência em tweets contendo #TeaParty

Entidade	Frequência
#teaparty	2834
#tcot	2426
#p2	911
#tlot	781
#gop	739
#ocra	649
#sgp	567
#twisters	269
#dnc	175
#tpp	170
#GOP	150
#iamthemob	123
#ucot	120
#libertarian	112
#obama	112
#vote2010	109
#TeaParty	106
#hhrs	104
#politics	100
#immigration	97
#cspj	96
#acon	91
#dems	82
#palin	79
#topprog	78
#Obama	74
#tweetcongress	72
#jcot	71
#Teaparty	62
#rs	60
#oilspill	59

Entidade	Frequência
#news	58
#glennbeck	54
#FF	47
#liberty	47
@welshman007	45
#spwbt	44
#TCOT	43
http://tinyurl.com/24h36zq	43
#rnc	42
#military	40
#palin12	40
@Drudge_Report	39
@ALIPAC	35
#majority	35
#NoAmnesty	35
#patriottweets	35
@ResistTyranny	34
#tsot	34
http://tinyurl.com/386k5hh	31
#conservative	30
#AZ	29
#TopProg	29
@JIDF	28
@STOPOBAMA2012	28
@TheFlaCracker	28
#palin2012	28
@thenewdeal	27
#AFIRE	27
#Dems	27
#asamom	26
#GOPDeficit	25
#wethepeople	25
@andilinks	24
@RonPaulNews	24
#ampats	24
#cnn	24
#jews	24

Entidade	Frequência
@First_Patriots	23
#patriot	23
#pjtv	23
@Liliaep	22
#nvsen	22
@BrnEyeSuss	21
@crispix49	21
@koopersmith	21
@Kriskxx	21
#Kagan	21
@blogging_tories	20
#cdnpoli	20
#fail	20
#nra	20
#roft	20

Tabela 5.4 – Entidades que surgem com maior frequência em tweets contendo #JustinBieber

Entidade	Frequência
#justinbieber	1613
#JustinBieber	1379
@lojadoaltivo	354
@ProSieben	258
#JUSTINBIEBER	191
#Proform	191
http://migre.me/TJwj	191
#Justinbieber	107
#nowplaying	104
@justinbieber	99
#music	88
#tickets	80
@_Yassi_	78
#musicmonday	78
#video	78
#Dschungel	74
#Celebrity	42
#beliebers	38
#BieberFact	38
@JustBieberFact	32

@TinselTownDirt	32
@rheinzeitung	28
#WTF	28
http://tinyurl.com/343kax4	28
#Telezwerge	26
#Escutando	22
#justinBieber	22
#Restart	22
#TT	22
http://bit.ly/aARD4t	21
http://bit.ly/b2Kc1L	21
#bieberblast	20
#Eclipse	20
#somebodytolove	20

O que fica imediatamente óbvio é que os tweets de #TeaParty parecem ter uma área "sob a curva" muito maior, além de uma cauda mais longa[8] (se é que podemos chamá-la de cauda) do que os tweets de #JustinBieber. Assim, em uma análise rápida, pode parecer que o número médio de hashtags para tweets de #TeaParty é maior do que o número dos tweets de #JustinBieber. A próxima seção investiga essa suposição, mas antes de avançarmos, podemos fazer mais algumas observações sobre esses resultados. Uma avaliação qualitativa superficial desses dados parece indicar que a informação codificada nas entidades em si é mais rica para #TeaParty. Por exemplo, em entidades #TeaParty, vemos tópicos como #oilspill, #Obama, #palin, #libertarian e *@Drudge_Report*, além de outros. Em contraste, a maioria das entidades mais frequentes de #JustinBieber é simplesmente uma variação de #JustinBieber, com outras hashtags espalhadas e não concentradas em tópicos específicos. Lembre-se, entretanto, de que isso não é de todo inesperado, uma vez que #TeaParty é um tópico muito político, enquanto #JustinBiber está associado à cultura pop e ao entretenimento.

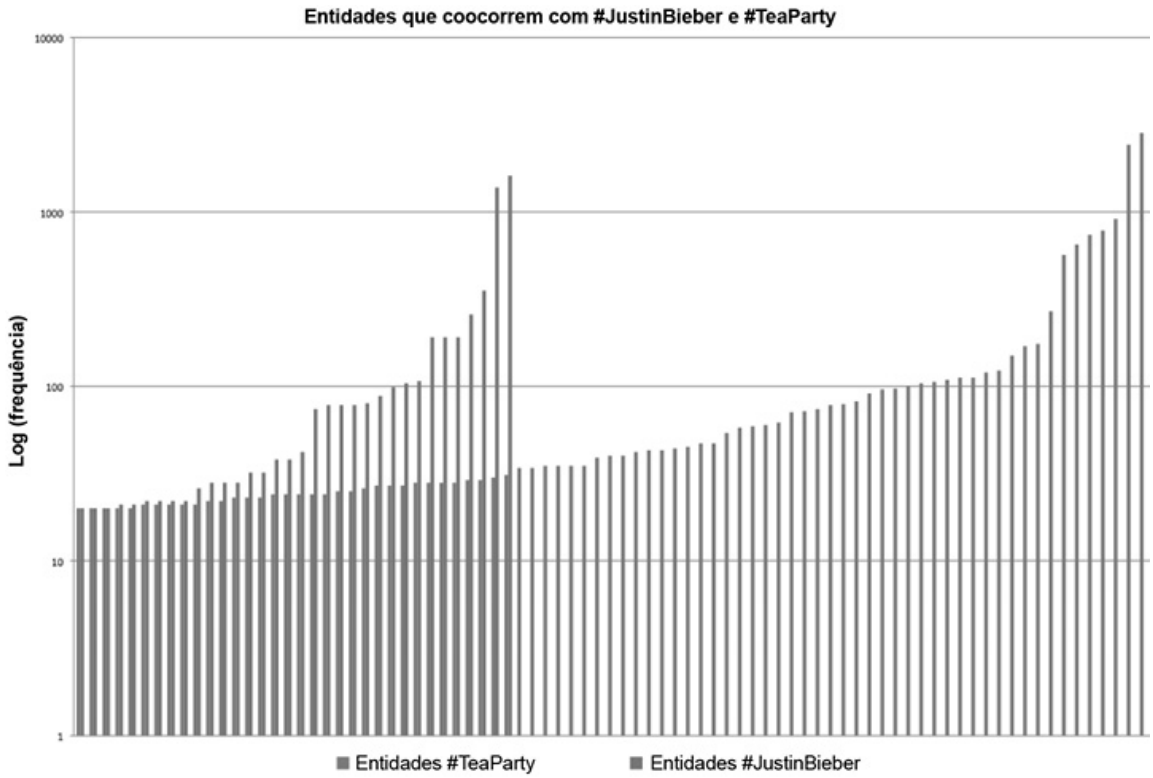


Figura 5.4 – Distribuição das entidades que coocorrem para #JustinBieber e #TeaParty.

Dentre outras observações que podem ser feitas, notamos que certas entidades de usuário (*@lojadoaltivo* e *@Pro-Sieben*) surgem como alguns dos principais resultados – em posições mais altas até mesmo do que a própria conta "oficial" *@justinbieber* – e que muitas das entidades que coocorrem com frequência com #JustinBieber são palavras ou entidades de usuário que não estão em inglês, geralmente associadas à indústria do entretenimento.

Tendo apenas arranhado a superfície de uma avaliação qualitativa, vamos agora retornar à questão anterior e descobrir se há mais hashtags por tweet para #TeaParty do que para #JustinBieber.

Na média, quais tweets têm mais hashtags, os de #JustinBieber ou os de #TeaParty?

O exemplo 5.13 fornece uma implementação funcional para contagem do número médio de hashtags por tweet, e pode ser prontamente aplicado aos bancos de dados `search-justinbieber` e `search-teaparty` sem que nenhum trabalho adicional seja necessário.

Computar os resultados para os dois bancos de dados revela que os

tweets de #JustinBieber têm uma média de cerca de 1,95 hashtags por tweet, enquanto tweets de #TeaParty têm em torno 5,15 hashtags por tweet. São aproximadamente 2,5 vezes mais hashtags para #TeaParty do que para #JustinBieber. Ainda que essa talvez não seja a descoberta mais surpreendente do mundo, é muito útil ter pontos de dados sólidos sobre os quais fundamentar futuras explorações e apoiar eventuais conjecturas: são resultados quantificáveis que podem ser monitorados cronologicamente, ou compartilhados e reavaliados por outros.

Mesmo que a diferença neste caso seja marcante, lembre-se de que os dados coletados são as informações que o Twitter nos passou como os cerca de 3.000 tweets mais recentes para cada tópico, por meio da API de busca. Eles não têm significado estatístico, ainda que sejam provavelmente um bom indicador e que possam muito bem retratar a verdade. Quer percebam ou não, usuários do Twitter que utilizam a hashtag #TeaParty são grandes adeptos das folksonomias: essas pessoas claramente têm um interesse particular em garantir que seu conteúdo seja facilmente acessível e que possa ser referenciado pelas APIs de busca e por exploradores de dados como nós.

Qual é retwittado com maior frequência: #JustinBieber ou #TeaParty?

Antes, neste capítulo, fizemos a conjectura de que tweets retwittados com grande frequência provavelmente são mais influentes e informativos, ou mesmo mais editoriais em natureza, do que tweets que não têm essa característica. Tweets como "Estou comendo um pretzel" e "Alienígenas acabaram de pousar na entrada da Casa Branca; vamos todos morrer! #fail #apocalypse" são provavelmente exemplos de conteúdo que, respectivamente, "não será" e "será" retwittado. Então, como #TeaParty se compara a #JustinBieber no que se refere a retweets? Analisar novamente a presença de *@menções*, a partir do conjunto de resultados de busca, produz resultados interessantes. Resultados truncados que mostram quais usuários retwittaram #TeaPary e #JustinBieber com maior frequência, utilizando como limite um parâmetro de frequência de 10, podem ser vistos nas tabelas 5.5 e 5.6.

Tabela 5.5 – Retwittadores mais frequentes de #TeaParty

Entidade	Frequência
@teapartyleader	10

@dhrxsol1234	11
@HCReminder	11
@ObamaBallBuster	11
@spitfiremurphy	11
@GregWHoward	12
@BrnEyeSuss	13
@Calroofer	13
@grammy620	13
@Herfarm	14
@andilinks	16
@c4Liberty	16
@FloridaPundit	16
@tlw3	16
@Kriskxx	18
@crispix49	19
@JIDF	19
@libertyideals	19
@blogging_tories	20
@Liliaep	21
@STOPOBAMA2012	22
@First_Patriots	23
@RonPaulNews	23
@TheFlaCracker	24
@thenewdeal	25
@ResistTyranny	29
@ALIPAC	32
@Drudge_Report	38
@welshman007	39

Tabela 5.6 – Retwittadores mais frequentes de #JustinBieber

Entidade	Frequência
@justinbieber	14
@JesusBeebs	16
@LeePhilipEvans	16
@JustBieberFact	32
@TinselTownDirt	32
@ProSieben	122
@lojadoaltivo	189

Se você fizer uma análise aproximada, executando o exemplo 5.4 nos cerca

de 3.000 tweets para cada tópico, descobrirá que algo como 1.335 dos tweets de #TeaParty são retweets, enquanto cerca de 763 dos tweets de #JustinBieber são retweets. Isso representa praticamente duas vezes mais retweets para #TeaParty do que para #JustinBieber. Você também observará que #TeaParty tem uma cauda muito mais longa, somando cerca de 400 retweets totais, contra 131 de #JustinBieber. Independentemente do rigor estatístico verificado, intuitivamente esses dados são indicadores muito relevantes que avaliam diferentes grupos de interesse de forma significativa. Poderíamos supor que a turma do #TeaParty retwitta mais consistentemente do que a do #JustinBieber; entretanto, dentre o pessoal do #JustinBieber que retwittou conteúdo, podemos notar claramente alguns usuários que se destacaram por retwittar com mais frequência do que outros. A figura 5.5 exibe um simples gráfico dos valores das tabelas 5.5 e 5.6. Assim como na figura 5.4, o eixo *y* é uma escala logarítmica que "achata" os valores de frequência para melhorar a legibilidade do gráfico, fazendo com que seja necessário menos espaço vertical.

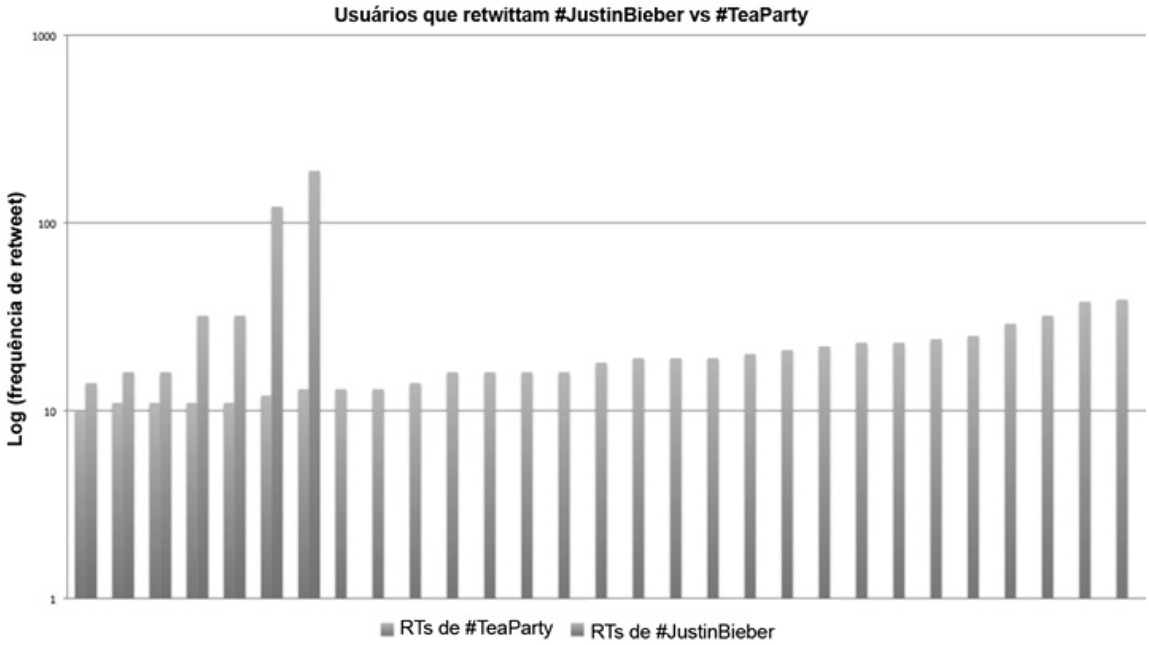


Figura 5.5 – Distribuição de usuários que retwittaram #JustinBieber e #TeaParty.

Quanta coincidência existe entre as entidades dos tweets de #TeaParty e #JustinBieber?

Uma questão final que paira sobre esses dados, e que talvez esteja lhe

roubando o sono à noite, diz respeito a quanta coincidência existe entre as entidades analisadas nos tweets de #TeaParty e #JustinBieber. Utilizando algumas das noções apresentadas no capítulo 4, estamos essencialmente perguntando qual a intersecção lógica entre dois conjuntos de entidades. Mesmo que certamente fosse possível computar esses dados adaptando código Python existente, pode ser mais fácil simplesmente capturar os resultados dos scripts que temos em dois arquivos e passar seus nomes como parâmetros em um script descartável que forneça uma ferramenta prática para computar a intersecção de qualquer arquivo delimitado por linhas. Além de produzir o resultado pretendido, essa abordagem também deixa artefatos que podem ser analisados calmamente e compartilhados com outros. Presumindo que você esteja trabalhando em um shell *nix com o script *count-entities-in-tweets.py*, veja a seguir uma abordagem para capturar as entidades da saída de #TeaParty e #JustinBieber do exemplo 5.4 e armazená-las de forma ordenada:

```
#!/bin/bash
mkdir -p out
for db in teaparty justinbieber; do
   python the_tweet__count_entities_in_tweets.py search-$db 0 | \
   tail +3 | awk '{print $2}' | sort > out/$db.entities
done
```

Depois de executar este script, você pode passar os dois nomes de arquivo ao programa Python para computar a saída (Exemplo 5.15).

Exemplo 5.15 – Computando a intersecção das linhas nos arquivos (the_tweet__compute_intersection_of_lines_in_files.py)

```
# -*- coding: utf-8 -*-
"""
Read in 2 or more files and compute the logical intersection of
the lines in them
"""

import sys

data = {}
for i in range(1, len(sys.argv)):
    data[sys.argv[i]] = set(open(sys.argv[i]).readlines())

intersection = set()
keys = data.keys()
for k in range(len(keys) - 1):
    intersection = data[keys[k]].intersection(data[keys[k - 1]])
```

```
msg = 'Common items shared amongst %s:' % ', '.join(keys).strip()
print msg
print '-' * len(msg)
for i in intersection:
    print i.strip()
```

As entidades compartilhadas por #JustinBieber e #TeaParty são um tanto previsíveis, mas nem por isso deixam de ser interessantes. O exemplo 5.16 lista os resultados de nosso exemplo.

Exemplo 5.16 – Exemplo de resultados do exemplo 5.15

```
Common items shared amongst teaparty.entities,
justinbieber.entities:
-----------------------------------------------------------------
----
#lol
#jesus
#worldcup
#teaparty
#AZ
#milk
#ff
#guns
#WorldCup
#bp
#News
#dancing
#music
#glennbeck
http://www.linkati.com/q/index
@addthis
#nowplaying
#news
#WTF
#fail
#toomanypeople
#oilspill
#catholic
```

Não deve surpreendê-lo que hashtags como #WorldCup, #worldcup e #oilspill estejam nos resultados, uma vez que estes são tópicos muito populares; entretanto, a presença de #teaparty, #glennbeck, #jesus e #catholic como parte da lista de hashtags compartilhadas pode ser um

tanto inesperada, caso você não esteja familiarizado com o movimento TeaParty. Uma análise mais prolongada poderia determinar com facilidade a força exata das correlações entre as duas buscas, contabilizando a frequência com que certas hashtags surgem em cada uma. Algo que fica imediatamente claro a partir desses resultados é que nenhuma dessas entidades usuais surge entre as 20 principais entidades associadas a #JustinBieber, o que indica que estão próximas ao final da cauda de distribuição de frequência para menções dessa hashtag (além disso, ter #WTF e #fail na lista, especialmente como um elo comum entre dois grupos distintos, é um pouco curioso, pois mostra que, infelizmente, experimentar frustrações é uma característica natural da humanidade). Se você deseja ir mais fundo, pode reutilizar o exemplo 5.7 como um exercício complementar e habilitar a indexação de texto completo nos tweets para poder realizar buscas por palavras-chave.

Visualização de toneladas de tweets

Há mais formas interessantes de visualizar dados do Twitter do que poderíamos abordar neste breve capítulo, mas isso não nos impedirá de demonstrar exercícios retratando algumas das abordagens mais óbvias e que fornecem uma boa base para sua pesquisa. Mais especificamente, estudaremos o carregamento de entidades de tweets em nuvens de tags e a visualização de "conexões" entre usuários com grafos.

Visualização de tweets com nuvens de tags

Nuvens de tags estão entre as escolhas mais óbvias para visualizar entidades extraídas de tweets. Há vários widgets interessantes com essa função que podem ser encontrados na web para cuidar do trabalho pesado, sendo que todos recebem a mesma entrada – essencialmente uma distribuição de frequência, como as que computamos no decorrer deste capítulo. Mas por que visualizar dados em uma nuvem de tags regular quando podemos empregar uma nuvem de tags giratória feita em Flash? De fato, há uma opção de código-aberto para nuvens de tags giratórias, o WP-Cumulus (*http://code.google.com/p/word-cumulus-goog-vis/wiki/UserGuide*), que produz um ótimo resultado. Para aplicá-lo, basta produzir o simples formato de entrada que ele espera e alimentar esse formato a um template contendo HTML padronizado.

O exemplo 5.17 é uma adaptação básica do exemplo 5.3, ilustrando uma rotina que emite uma simples estrutura JSON (uma lista de tuplas [termo, URL, frequência]) que pode ser transmitida a um template HTML para o WP-Cumulus. Passaremos strings vazias na seção do URL dessas tuplas, mas você pode muito bem usar sua criatividade e oferecer hiperlinks para um web service simples que exibe uma lista de tweets contendo as entidades. Lembre-se de que o exemplo 5.7 fornece praticamente tudo de que você necessita para preparar algo desse tipo, utilizando o `couchdb-lucene` para realizar uma busca de texto completa nos tweets armazenados no CouchDB. Outra opção seria criar um web service e vincular um URL que fornecesse todos os tweets contendo a entidade especificada.

Exemplo 5.17 – Geração dos dados para uma nuvem de tags interativa utilizando o WP-Cumulus (the_tweet__tweet_tagcloud_code.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import webbrowser
import json
from cgi import escape
from math import log
import couchdb
from couchdb.design import ViewDefinition
DB = sys.argv[1]
MIN_FREQUENCY = int(sys.argv[2])
HTML_TEMPLATE = '../web_code/wp_cumulus/tagcloud_template.html'
MIN_FONT_SIZE = 3
MAX_FONT_SIZE = 20
server = couchdb.Server('http://localhost:5984')
db = server[DB]
# Mapeia entidades nos tweets aos documentos em que elas surgem
def entityCountMapper(doc):
    if not doc.get('entities'):
        import twitter_text
        def getEntities(tweet):
            # Agora, extraia várias entidades dele e construa uma
estrutura familiar
            extractor = twitter_text.Extractor(tweet['text'])
```

```
            # Note que a API de produção do Twitter contém alguns
campos adicionais no
            # hash das entidades que exigiriam chamadas
adicionais à API para serem transformados
            entities = {}
            entities['user_mentions'] = []
            for um in
extractor.extract_mentioned_screen_names_with_indices():
                entities['user_mentions'].append(um)
            entities['hashtags'] = []
            for ht in extractor.extract_hashtags_with_indices():
            # Ajusta o nome do campo para corresponder à API
Twitter de produção
                ht['text'] = ht['hashtag']
                del ht['hashtag']
                entities['hashtags'].append(ht)
            entities['urls'] = []
            for url in extractor.extract_urls_with_indices():
                entities['urls'].append(url)
            return entities
        doc['entities'] = getEntities(doc)
    if doc['entities'].get('user_mentions'):
        for user_mention in doc['entities']['user_mentions']:
            yield ('@' + user_mention['screen_name'].lower(),
[doc['_id'], doc['id']])
    if doc['entities'].get('hashtags'):
        for hashtag in doc['entities']['hashtags']:
            yield ('#' + hashtag['text'], [doc['_id'],
doc['id']])
def summingReducer(keys, values, rereduce):
    if rereduce:
        return sum(values)
    else:
        return len(values)
view = ViewDefinition('index', 'entity_count_by_doc',
entityCountMapper,
                      reduce_fun=summingReducer,
language='python')
view.sync(db)
entities_freqs = [(row.key, row.value) for row in
                  db.view('index/entity_count_by_doc',
```

```
group=True)]
# Crie a saída para a nuvem de tags WP-Cumulus e ordene os termos
por frequência
raw_output = sorted([[escape(term), '', freq] for (term, freq) in
entities_freqs
                     if freq > MIN_FREQUENCY], key=lambda x: x[2])
# Implementação adaptada de
# http://help.com/post/383276-anyone-knows-the-formula-for-font-s
min_freq = raw_output[0][2]
max_freq = raw_output[-1][2]
def weightTermByFreq(f):
    return (f - min_freq) * (MAX_FONT_SIZE - MIN_FONT_SIZE) /
(max_freq
            - min_freq) + MIN_FONT_SIZE
weighted_output = [[i[0], i[1], weightTermByFreq(i[2])] for i in
raw_output]
# Substitua a estrutura de dados JSON no template
html_page = open(HTML_TEMPLATE).read() %
(json.dumps(weighted_output),)
if not os.path.isdir('out'):
    os.mkdir('out')
f = open(os.path.join('out', os.path.basename(HTML_TEMPLATE)),
'w')
f.write(html_page)
f.close()
print 'Tagcloud stored in: %s' % f.name
# Abra a página web em seu navegador
webbrowser.open("file://" + os.path.join(os.getcwd(), 'out',
os.path.basename(HTML_TEMPLATE)))
```

A seção mais notável desse código é a incorporação da fórmula que vemos a seguir (*http://help.com/post/383276-anyone-knows-the-formula-for-font-s*), para classificar as tags na nuvem:

$$\frac{(f - \text{min\_freq}) * (\text{MAX\_FONT\_SIZE} - \text{MIN\_FONT\_SIZE})}{\text{max\_freq} - \text{min\_freq}} + \text{MIN\_FONT\_SIZE}$$

Essa fórmula avalia a frequência dos termos, achatando seus valores entre `MIN_FONT_SIZE` e `MAX_FONT_SIZE`; considera-se a frequência do termo em questão, além dos valores mínimo e máximo de frequência para os dados. Há muitas variações que poderiam ser aplicadas a essa fórmula, e a

incorporação de logaritmos não é de todo incomum. O texto de Kevin Hoffman, "In Search of the Perfect Tag Cloud" (*http://files.blog-city.com/files/J05/88284/b/insearchofperfecttagcloud.pdf*), fornece uma boa visão geral das várias decisões de design envolvidas na criação de nuvens de tags, além de ser um ponto de partida perfeito caso você esteja interessado em aprender mais sobre design de nuvens de tags.

O arquivo `tagcloud_template.html` mencionado no exemplo 5.17 não apresenta nada de diferente, e está disponível com o código-fonte deste livro no GitHub (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/web_code/wp_cumulus/tagcloud_template.html*); ele não é nada além de uma simples adaptação do exemplo que acompanha o código-fonte da nuvem de tags. Algumas tags de script no cabeçalho da página cuidam de todo o trabalho pesado, e tudo que você precisa fazer é alimentar alguns dados em um template provisório, que utiliza substituição de strings para substituir o espaço reservado do `%s`.

As figuras 5.6 e 5.7 mostram nuvens de tags para entidades de tweets com frequência maior que 30, que coocorrem com #JustinBieber e #TeaParty. A diferença mais óbvia entre elas é o volume da nuvem de #TeaParty quando comparado ao da nuvem de #JustinBieber. A presença de outros tópicos associados aos termos de consulta também fica evidente. Já sabíamos disso desde a figura 5.5, mas uma nuvem de tags, além de transmitir um resultado semelhante, fornece capacidades interativas. Evidentemente, nada impede que você crie gráficos Ajax interativos com ferramentas como o Google Chart Tools (*http://code.google.com/apis/visualization/documentation/gallery.html*).

Figura 5.6 – Nuvem de tags interativa em 3D para entidades de tweets que coocorrem com #JustinBieber.

Visualização de estruturas de comunidade em resultados de busca do Twitter

Antes neste capítulo, comparamos brevemente as consultas de #JustinBieber e #TeaParty. Esta seção aprofunda essa análise, apresentando algumas visualizações feitas a partir de uma perspectiva levemente diferente. Vamos tentar visualizar as estruturas de comunidade dos usuários que empregam as entidades #TeaParty e #JustinBieber, tomando os resultados de busca coletados antes, computando as amizades entre os autores dos tweets e outras entidades que surgem nesses tweets (como @menções e #hashtags) e visualizando essas conexões. Além de oferecer insights adicionais aos nossos exemplos, essas técnicas também funcionam como pontos de partida perfeitos para outras situações nas quais você busca uma justaposição interessante. O código para este fluxo de trabalho não será mostrado aqui, pois pode ser facilmente criado reciclando código que já vimos neste capítulo e nos capítulos anteriores. Os passos abrangentes envolvidos nesse processo incluem:

- Computar o conjunto de screen names para os autores de tweets únicos e menções de usuário associadas aos bancos de dados do CouchDB, `search-teaparty` e `search-justinbieber`.
- Ccoletar os IDs de amigos para os screen names com o recurso

`/friends/ids` do Twitter.

- Transformar em screen names esses IDs de amigos com o recurso `/users/lookup` do Twitter (lembre-se de que não há uma forma direta de consultar screen names para amigos; valores de ID têm de ser coletados e depois transformados).
- Construir um `networkx.Graph` percorrendo as amizades e criando uma aresta entre dois nós, sempre que existir uma amizade em qualquer uma das direções.
- Analisar e visualizar os grafos.

Figura 5.7 – Nuvem de tags interativa em 3D para entidades de tweets que coocorrem com #TeaParty.

O resultado do script (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/*) é um arquivo de grafo salvo que você pode abrir no interpretador e analisar, como ilustrado pela sessão no exemplo 5.18. Como o script é essencialmente composto apenas por trechos de lógica reciclada de exemplos anteriores, ele não será incluído aqui, mas está disponível on-line (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/the_tweet__get_friends_for_entity_mentions_in_search_results.py*). A análise é gerada executando o script em aproximadamente 3.000 tweets para cada uma das buscas de

#JustinBieber e #TeaParty. A saída das chamadas a `nx.degree`, que retornam o grau de cada nó no grafo[2], foi omitida, mas está renderizada visualmente como um simples gráfico de coluna na figura 5.8.

Exemplo 5.18 – Uso do interpretador para realizar análise ad-hoc de grafos

```
>>> import networkx as nx
>>> teaparty = nx.read_gpickle("out/search-teaparty.gpickle")
>>> justinbieber = nx.read_gpickle("out/search-
justinbieber.gpickle")
>>> teaparty.number_of_nodes(), teaparty.number_of_edges()
(2262, 129812)
>>> nx.density(teaparty)
0.050763513558431887
>>> sorted(nx.degree(teaparty)
... output omitted ...
>>> justinbieber.number_of_nodes(),
justinbieber.number_of_edges()
(1130, 1911)
>>> nx.density(justinbieber)
>>> 0.0029958378077553165
>>> sorted(nx.degree(teaparty))
... output omitted ...
```

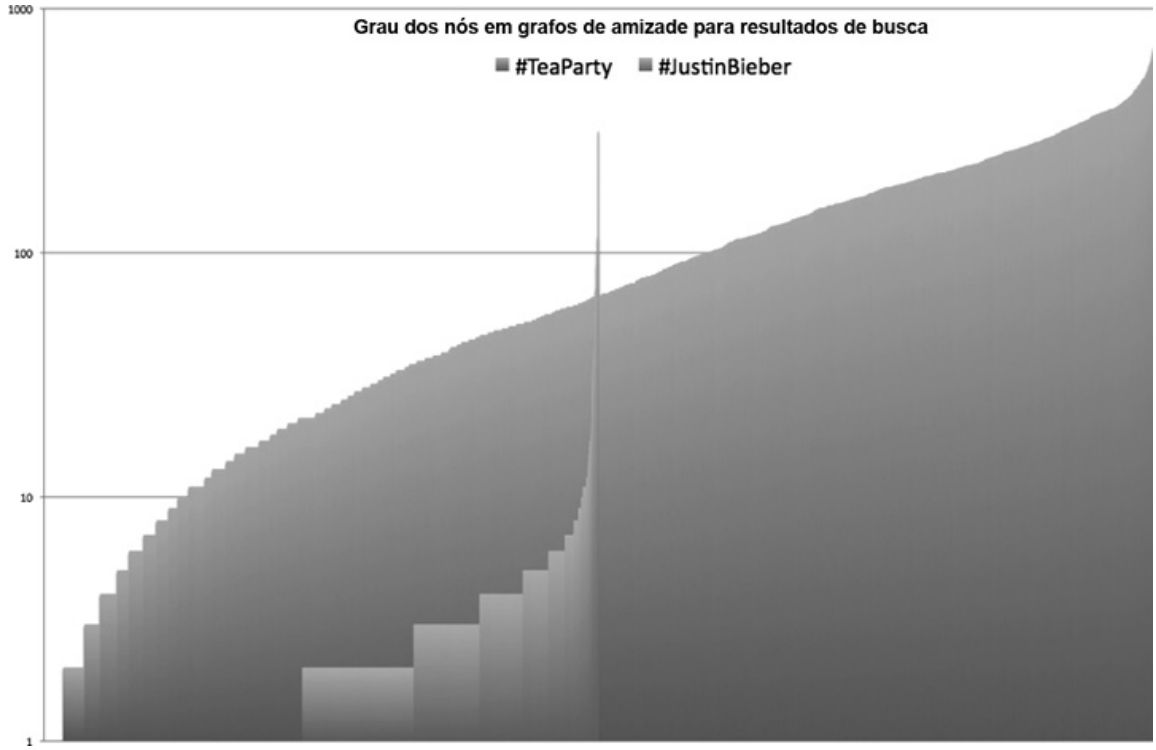


Figura 5.8 – Uma forma simples de comparar a "conectividade" dos grafos é representar o

grau ordenado de cada nó e sobrepor as curvas.

Mesmo sem qualquer representação visual, os resultados da sessão do interpretador são muito reveladores. De fato, é essencial que você seja capaz de conjeturar sobre dados como estes mesmo sem uma visualização disponível, uma vez que, com frequência, você simplesmente não será capaz de visualizar grafos complexos sem um grande investimento de tempo. Em resumo, a densidade e a razão de nós para arestas (usuários para amizades entre eles), no grafo de #TeaParty, superam enormemente as do grafo de #JustinBieber. Todavia, essa observação por si só pode não ser tão surpreendente. O que mais chama nossa atenção é a conectividade incrivelmente alta dos nós no grafo de #TeaParty. Evidentemente, sabemos que #TeaParty é um movimento político intenso, com um número de seguidores relativamente pequeno, enquanto #JustinBieber é um tópico de entretenimento muito menos concentrado, com apelo internacional. Ainda assim, a distribuição e a forma geral da curva para os resultados de #TeaParty fornecem uma visualização intuitiva e tangível da conectividade que há entre as pessoas interessadas nesse tópico. Uma representação visual em 2D do grafo não fornece tantas informações adicionais, mas o suspense deve estar sendo terrível, por isso, sem mais cerimônias – Figura 5.9.

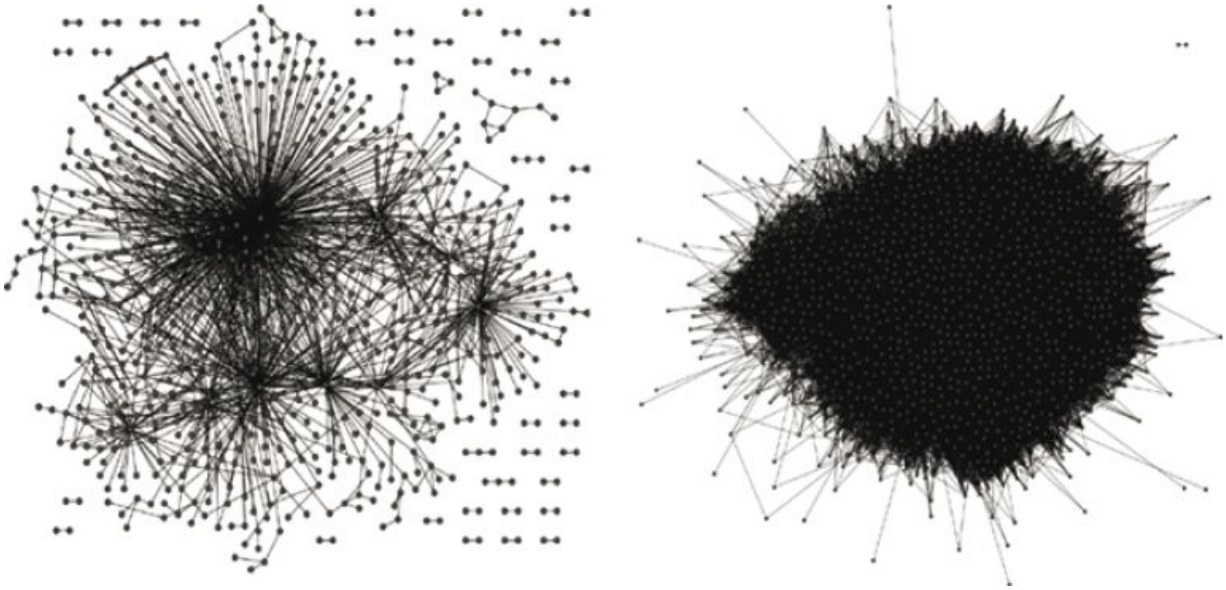


Figura 5.9 – Representação 2D evidenciando a conectividade dos usuários que twittam sobre #JutinBieber (à esquerda) e #TeaParty (à direita); cada aresta representa uma amizade que existe em ao menos uma direção.

Lembre-se de que usuários *nix podem escrever uma saída para o

Graphviz com `nx.drawing.write_dot`, mas usuários do Windows talvez tenham de gerar manualmente a saída em linguagem DOT. Veja os exemplos 1.12 e 2.4. Para grafos não-direcionados extensos, você pode aproveitar o engine SFDP (scalable force directed placement) (*http://www2.research.att.com/~yifanhu/PUB/peripheral.pdf*) do Graphviz. Um exemplo que mostra como você poderia invocá-lo em uma linha de comando, produzindo um layout útil para um grafo denso, pode ser visto a seguir:

```
$sfdp -Tpng -
Oteaparty -Nfixedsize=true -Nlabel='' -Nstyle=filled -
Nfillcolor=red -
Nheight=0.1 -Nwidth=0.1 -Nshape=circle -Gratio=fill -
Goutputorder=edgesfirst -
Gratio=fill -Gsize='10!' -Goverlap=prism teaparty.dot
```

Comentários finais

Há um número enorme de outras atividades que podem ser feitas com dados de tweets, e se você for um pouco criativo, as possibilidades serão incríveis. Apenas arranhamos a superfície desse tópico neste capítulo. Um livro inteiro poderia literalmente ser devotado ao trabalho sistemático com algumas dessas possibilidades, e muitas pequenas empresas que se concentram em análises de tweets poderiam obter lucros consideráveis vendendo respostas para certas classes de consultas ad-hoc que clientes costumam realizar. Veja algumas sugestões interessantes para as quais você pode dar continuidade:

- Defina uma métrica de similaridade e utilize-a para comparar ou analisar de modo mais profundo dois usuários ou grupos de usuários no Twitter. Você essencialmente desenvolverá um perfil, para depois medir como um usuário específico se encaixa nele. Por exemplo, por acaso certas hashtags, palavras-chave (por exemplo, LOL, OMG etc.), ou métricas semelhantes surgem mais em tweets de #JustinBieber do que em tweets mais intelectuais, como os associados a #gov20? Essa questão pode parecer óbvia, mas quais dados interessantes podemos descobrir ao calcular uma resposta quantificável?
- Quais celebridades que utilizam o Twitter têm baixa concentração de entidades em seus tweets, mas um alto volume de tweets? Por acaso, isso faz com que seus tweets não tenham importância – como os

daquelas pessoas que twittam 100 vezes por dia, sobre nada específico? Da mesma forma, quais celebridades que utilizam o Twitter têm alta concentração de hashtags em seus tweets? Quais comparações óbvias podemos fazer entre esses dois grupos?

- Se você tem uma conta no Twitter e um número razoável de tweets, considere analisar seus próprios dados e comparar seu fluxo de tweets ao desses usuários. Qual a similaridade entre o seu comportamento e o de outros usuários apresentados em um panorama mais amplo?
- Não abordamos o assunto das APIs de lista do Twitter. Será que você consegue realizar o perfil dos vários membros de uma lista e tentar descobrir redes sociais latentes? Por exemplo, seria possível encontrar uma métrica que permitisse estimar fronteiras políticas entre seguidores das listas *@timoreilly/healthcare* e *@timoreilly/gov20*?
- Pode ser interessante analisar o volume de tweets ou o conteúdo dos tweets de acordo com a hora do dia. Trata-se de um exercício simples em que você pode incorporar o SIMILE Timeline – discutido na seção "Visualização de eventos de e-mails com o SIMILE Timeline", do capítulo 3 – para fácil visualização.
- Comunicações em tempo quase real estão se tornando a regra, e o Twitter fornece uma API de streaming para acesso conveniente a grande parte de seu fluxo. Consulte a documentação on-line sobre streaming (*http://dev.twitter.com/pages/streaming_api*) e não deixe de verificar o `tweepy` (*https://github.com/joshthecoder/tweepy*), um ótimo cliente de streaming.

---

1 O módulo twitter-text-py é um port do módulo twitter-text-rb (ambos disponíveis via GitHub), que o Twitter utiliza na produção (*http://dev.twitter.com/pages/tweet_entities*).

2 A documentação da API do Twitter afirma que a timeline dos amigos é semelhante à timeline da página inicial, exceto pelo fato de que não contém retweets por considerações de compatibilidade retroativa.

3 Veja a capa da edição de maio de 2010 da revista Inc. (*http://www.inc.com/magazine/20100501/the-oracle-of-silicon-valley.html*).

4 Note que, no final de dezembro de 2010, o campo retweet_count tinha como limite máximo o valor de 100. Para este lote específico de dados, dois tweets foram retwittados exatamente 100 vezes, e 59 retwittados "100+".

5 Steven D. Levitt é o coautor de Freakonomics: A Rogue Economist Explores the Hidden Side of Everything (Harper), livro que utiliza sistematicamente dados para responder a perguntas aparentemente radicais como "O que professores de escola e lutadores de sumo têm em comum?"

6 N.T.: o Tea Party é um movimento social e político populista, conservador e de ultradireita que surgiu nos Estados Unidos em 2009 por meio de uma série de protestos coordenados tanto em nível local, quanto nacional (fonte: Wikipédia).

7 Essa questão foi parcialmente inspirada pelo interessante post "Data science democratized" (*http://radar.oreilly.com/2010/07/data-science-democratized.html*), feito no Radar, que menciona uma apresentação investigando a mesma questão (*http://www.slideshare.net/bigdatacamp/stefan-groschupf-of-datameer-gives-lightning-talk-at-big-datacamp-4667387*).

8 Uma "cauda longa" ou "pesada" refere-se a uma característica de certas distribuições estatísticas nas quais uma parte significativa (geralmente 50% ou mais) da área sob a curva existe dentro de sua cauda. Veremos novamente este conceito como parte de uma breve visão geral da lei de Zipf na seção "Hacking de dados com NLTK", no capítulo 7.

9 O grau de um nó em um grafo não-direcionado é seu número total de arestas.

CAPÍTULO 6

LinkedIn: agrupe sua rede profissional por diversão (e lucro?)

Este capítulo apresenta técnicas e considerações sobre a mineração dos dados armazenados pelo LinkedIn, um popular site de rede social concentrado em relacionamentos profissionais e de negócios. Conforme você acompanha o texto, encorajamos que crie uma rede profissional no LinkedIn, mas, mesmo que não o faça, você ainda poderá aplicar muitas das técnicas que mostraremos em outros domínios que ofereçam formas de exportar informações semelhantes às encontradas em um catálogo de endereços (address book). Mesmo que inicialmente o LinkedIn possa parecer similar às outras redes sociais, os dados que sua API fornece são de natureza inerentemente diferente. Em termos gerais, pessoas que se tornam membros do LinkedIn estão interessadas nas oportunidades de negócio que ele oferece e desejam que seus perfis tenham o melhor visual possível – o que significa garantir que forneçam amplos detalhes de natureza relativamente pessoal, indicando relacionamentos importantes de negócios, detalhes profissionais etc.

Dada a natureza um tanto particular desses dados, a API do LinkedIn é um pouco diferente das outras que vimos neste livro. Por exemplo, ainda que você possa geralmente acessar todos os detalhes interessantes sobre histórico educacional, cargos, posições ocupadas anteriormente etc. de *seus contatos*, você não pode determinar se duas pessoas estão "conectadas", e a falta de tal método não é acidental – essa decisão foi tomada, pois a equipe de gerenciamento do LinkedIn acredita que os dados de sua rede profissional são inerentemente particulares e valiosos demais para estarem abertos às mesmas possibilidades de exploração verificadas, digamos, nas informações sobre seus amigos no Twitter e no Facebook[1].

Como o LinkedIn limita seu acesso a informações referentes às suas conexões diretas, e não é muito adequado para mineração em grafos, ele

requer que você formule questões diferentes sobre os dados disponíveis. O restante deste capítulo apresenta técnicas de agrupamento (ou de *clustering*)[2] fundamentais que podem ajudá-lo a responder às seguintes questões:

- Quais de suas conexões são as mais semelhantes, de acordo com um critério como cargo ocupado?
- Quais de suas conexões já trabalharam em empresas nas quais você deseja trabalhar?
- Onde reside geograficamente a maioria de suas conexões?

Motivações para agrupamentos

Dada a riqueza dos dados do LinkedIn, a capacidade de responder questões sobre suas redes profissionais apresenta algumas oportunidades poderosas. Entretanto, quando implementamos soluções nesse sentido, há ao menos dois temas usuais que encontramos seguidas vezes:

- Muitas vezes é necessário medir a similaridade entre dois valores (geralmente valores em strings), quer eles representem cargos ocupados, nomes de empresas, interesses profissionais ou qualquer outro campo que possa ser apresentado como texto livre. O capítulo 7 introduz oficialmente algumas abordagens e considerações adicionais para medição de similaridade que você talvez também queira estudar.
- Para agrupar todos os itens de um conjunto utilizando uma métrica de similaridade, seria ideal comparar todos os seus membros. Assim, para um conjunto de n membros, você realizaria em seu algoritmo algo na ordem de n^2 computações de similaridade para o cenário de pior caso. Cientistas da computação chamam essa situação de um problema de *n ao quadrado* e, geralmente, utilizam a nomenclatura $O(n^2)$ para descrevê-lo[3]; você poderia dizer que se trata de um problema do tipo "Ordem O de n ao quadrado"[4]. Todavia, problemas $O(n^2)$ tornam-se intratáveis para grandes valores de n. Na maioria das situações, o uso do termo "intratável" significa que seriam necessários anos (ou até mesmo centenas de anos) para que uma solução fosse computada – de qualquer maneira, tempo demais para que ela tivesse utilidade.

Essas duas questões surgem inevitavelmente em qualquer discussão que

envolva o que é conhecido como *problema da correspondência aproximada*[5], tópico muito estudado (mas também complicado e trabalhoso) que se faz presente em praticamente todos os setores.

Técnicas para correspondência aproximada são um componente fundamental do cinto de ferramentas de um minerador de dados legítimo, pois em praticamente todos os setores do mercado – desde inteligência de defesa, até detecção de fraudes bancárias e trabalho de empresas locais de paisagismo – pode haver uma quantidade verdadeiramente imensa de dados relacionais semipadronizados para análise (quer as pessoas no comando percebam ou não essa necessidade). O que geralmente ocorre é que uma empresa estabelece um banco de dados para coletar algum tipo de informação, mas nem todos os campos são enumerados em um universo predefinido de respostas válidas. Quer seja porque a lógica da interface de usuário da aplicação não foi projetada corretamente, porque alguns campos não são adequados para armazenar valores estáticos predeterminados, ou ainda porque era crítico para a experiência do usuário que lhe fosse permitido digitar o que quisesse em uma caixa de texto, o resultado é sempre o mesmo: você eventualmente terá muitos dados semipadronizados, ou registros "sujos". Ainda que possa haver um total de *N* valores em string distintos para um campo específico, muitos desses valores estarão de fato relacionados ao mesmo conceito. Dados duplicados podem ocorrer por diversas razões: erros de digitação, abreviações, diferenças entre letras maiúsculas e minúsculas etc.

Mesmo que não seja óbvio, esta é exatamente a situação que encontramos na mineração de dados do LinkedIn: membros do LinkedIn são capazes de digitar suas informações profissionais como texto livre, o que resulta em uma quantidade inevitável de variação. Por exemplo, se você quisesse analisar sua rede profissional e determinar o local de trabalho da maioria de suas conexões, seria necessário considerar variações usuais nos nomes das empresas. Até mesmo os nomes mais simples apresentam algumas variações que você certamente encontrará. Por exemplo, deve ser óbvio, para a maioria das pessoas, que "Google" é uma forma abreviada de "Google, Inc.", mas mesmo essas simples variações de nomenclatura devem ser explicitamente consideradas durante tentativas de padronização. Na padronização de nomes de empresas, um bom ponto de partida é primeiro considerar sufixos como ", Inc.", ", LLC" etc. Presumindo que você tenha exportado do LinkedIn um arquivo separado

por vírgulas (CSV) com seus contatos, você pode iniciar realizando uma normalização básica e, depois, imprimir entidades selecionadas de um histograma (Exemplo 6.1).

São duas as formas primárias para acesso de seus dados do LinkedIn: exportar essas informações como dados de catálogos de endereços, incluindo informações básicas como nome, cargo, empresa e informações para contato; ou utilizar a API do LinkedIn para explorar de modo programático os detalhes completos de seus contatos. Ainda que utilizando a API possamos acessar tudo que estiver visível a um usuário autenticado navegando pelos perfis em *http://linkedin.com*, para este primeiro exercício, podemos obter todos os detalhes de cargos de que necessitamos simplesmente exportando informações de catálogo de endereços (ainda utilizaremos as APIs do LinkedIn neste capítulo, na seção "Busca de informações estendidas de perfil"). Caso não esteja óbvio como proceder para exportar suas conexões, a figura 6.1 oferece algumas dicas visuais que mostram como realizar essa tarefa. Utilizaremos o módulo `csv` da Python para parsing dos dados exportados; para garantir compatibilidade com códigos futuros, escolha a opção "Outlook CSV".

Exemplo 6.1 – Normalização simples dos sufixos das empresas, em dados de catálogo de endereços (linkedin__analyze_companies.py)

```
# -*- coding: utf-8 -*-
import sys
import nltk
import csv
from prettytable import PrettyTable
CSV_FILE = sys.argv[1]
# Manipula as abreviações conhecidas, removendo sufixos comuns
etc.
transforms = [(', Inc.', ''), (', Inc', ''), (', LLC', ''), (',
LLP', '')]
csvReader = csv.DictReader(open(CSV_FILE), delimiter=',',
quotechar='"')
contacts = [row for row in csvReader]
companies = [c['Company'].strip() for c in contacts if
c['Company'].strip() != '']
for i in range(len(companies)):
    for transform in transforms:
```

```
        companies[i] = companies[i].replace(*transform)
pt = PrettyTable(fields=['Company', 'Freq'])
pt.set_field_align('Company', 'l')
fdist = nltk.FreqDist(companies)
[pt.add_row([company, freq]) for (company, freq) in fdist.items()
if freq > 1]
pt.printt()
```

Entretanto, seria necessário um pouco mais de sofisticação para manipular algumas situações mais complexas que podem ser encontradas, como as várias manifestações diferentes de nomes de empresas, como a O'Reilly Media. Por exemplo, você poderia ver o nome dessa empresa representado como O'Reilly & Associates, O'Reilly Media, O'Reilly, Inc. [6], ou simplesmente O'Reilly. O mesmo problema se apresenta na consideração da nomenclatura de cargos. Nesses casos, a situação pode se complicar ainda mais, uma vez que títulos como esses podem apresentar muitas variações. A tabela 6.1 lista alguns cargos que você provavelmente encontrará em uma empresa de software, incluindo algumas variações naturais. Quantos cargos distintos você identifica para os 10 títulos diferentes listados?

Tabela 6.1 – Exemplo de cargos para a indústria da tecnologia

Cargo
Diretor Executivo
Presidente/CEO
Presidente & CEO
CEO
Desenvolvedor
Desenvolvedor de Software
Engenheiro de Software
Diretor Técnico (CTO)
Presidente
Engenheiro Sênior de Software

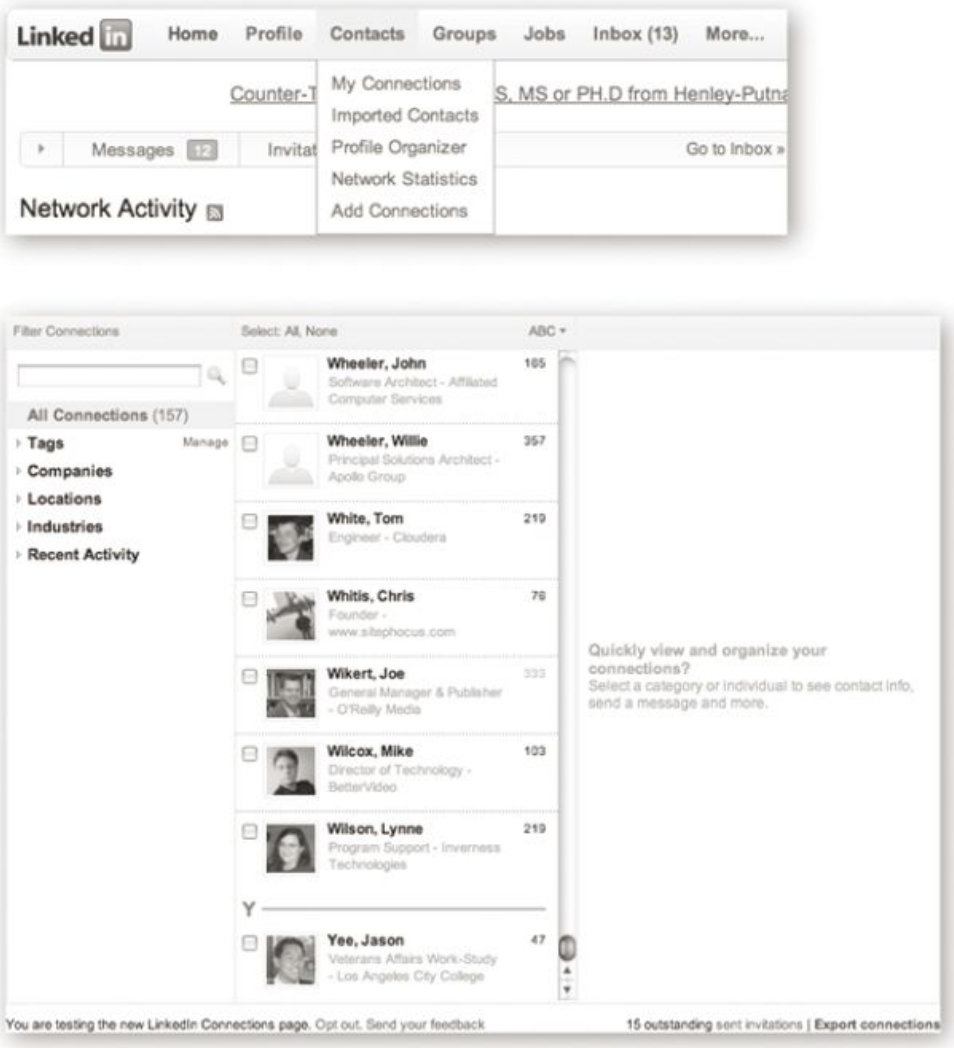


Figura 6.1 – Para exportar suas conexões, clique em Contacts, escolha My Connections do menu pop-up, e então clique no link "Export connections" na tela seguinte.

Mesmo que certamente seja possível definir uma lista de abreviações e nomes alternativos que faça a correspondência de títulos como CEO e Diretor Executivo, pode não ser muito prático definir manualmente listas que façam a correspondência de cargos como Engenheiro de Software e Desenvolvedor, para todos os domínios possíveis. Entretanto, mesmo para os campos mais confusos em um cenário de pior caso, não deve ser muito difícil implementar uma solução que condense os dados a ponto de torná-los manuseáveis por um especialista, e depois transmitir esses dados a um programa que possa aplicá-los da mesma forma que o especialista teria feito. Com frequência, esta é a abordagem preferida pelas organizações, pois permite a participação de humanos no controle de

qualidade.

Agrupamento de contatos por cargo

Agora que você pode (com sorte) valorizar adequadamente a natureza do problema da correspondência de registros, vamos coletar alguns dados reais do LinkedIn e criar alguns agrupamentos. Você pode ter muitas razões para minerar dados do LinkedIn, dentre elas o desejo de verificar se suas habilidades de relacionamento têm lhe ajudado a conhecer as "pessoas certas", ou de abordar contatos que provavelmente se enquadram em certa categoria socioeconômica com um tipo específico de consulta ou proposta de negócio. O restante desta seção demonstra sistematicamente alguns exercícios para agrupamento de seus contatos de acordo com seus cargos.

Padronização e contagem de cargos

Um ponto de partida óbvio, quando trabalhamos com um novo conjunto de dados, é contar seus elementos, e este caso não é diferente. Esta seção apresenta um padrão para transformação de cargos comuns e depois realiza uma análise de frequência básica.

Presumindo que você tenha exportado um número razoável de contatos, as nuances mais sutis que encontraremos neles poderão, de fato, surpreendê-lo. Todavia, antes de abordarmos esses detalhes, vamos preparar um pouco de código que estabeleça padrões para normalização dos dados de registro, formando um inventário básico ordenado por frequência. O exemplo 6.2 inspeciona cargos e imprime informações de frequência, tanto para os cargos em si, quanto para os tokens individuais que ocorrem neles. Caso você ainda não o tenha feito, execute `easy_install prettytable`, instalando um pacote que utilizaremos para produzir resultado tabular devidamente formatado.

Exemplo 6.2 – Padronização de cargos usuais e computação de suas frequências (linkedin__analyze_titles.py)

```
# -*- coding: utf-8 -*-
import sys
import nltk
import csv
from prettytable import PrettyTable
```

```
CSV_FILE = sys.argv[1]
transforms = [
    ('Sr.', 'Senior'),
    ('Sr', 'Senior'),
    ('Jr.', 'Junior'),
    ('Jr', 'Junior'),
    ('CEO', 'Chief Executive Officer'),
    ('COO', 'Chief Operating Officer'),
    ('CTO', 'Chief Technology Officer'),
    ('CFO', 'Chief Finance Officer'),
    ('VP', 'Vice President'),
   ]
csvReader = csv.DictReader(open(CSV_FILE), delimiter=',',
quotechar='"')
contacts = [row for row in csvReader]
# Lê uma lista de títulos e separa
# títulos combinados como #President/CEOP
# Outras variações também poderiam ser manipuladas
# como "President & CEO", "President and CEO" etc.
titles = []
for contact in contacts:
    titles.extend([t.strip() for t in contact['Job
Title'].split('/')
                   if contact['Job Title'].strip() != ''])
# Substitui abreviações usuais/conhecidas
for i in range(len(titles)):
    for transform in transforms:
        titles[i] = titles[i].replace(*transform)
# Imprime uma tabela de cargos ordenados por frequência
pt = PrettyTable(fields=['Title', 'Freq'])
pt.set_field_align('Title', 'l')
titles_fdist = nltk.FreqDist(titles)
[pt.add_row([title, freq]) for (title, freq) in
titles_fdist.items() if freq > 1]
pt.printt()
# Imprime uma tabela de tokens ordenados por frequência
tokens = []
for title in titles:
    tokens.extend([t.strip(',') for t in title.split()])
pt = PrettyTable(fields=['Token', 'Freq'])
```

```
pt.set_field_align('Token', 'l')
tokens_fdist = nltk.FreqDist(tokens)
[pt.add_row([token, freq]) for (token, freq) in
tokens_fdist.items() if freq > 1
and len(token) > 2]
pt.printt()
```

Em resumo, o código lê registros CSV e tenta normalizá-los, separando títulos combinados que utilizam uma barra (como um título de "Presidente/CEO") e substituindo abreviações conhecidas. Além disso, ele simplesmente exibe os resultados de uma distribuição de frequência, tanto dos cargos com texto completo, quanto dos tokens individuais contidos neles. Isso não é nenhuma novidade, mas serve como ótimo template inicial, além de lhe fornecer certo insight sobre a distribuição dos dados. Um exemplo de resultado pode ser visto a seguir.

Exemplo 6.3 – Exemplos de resultados do exemplo 6.2

```
+--------------------------+------+
|          Title           | Freq |
+--------------------------+------+
| Chief Executive Officer  |  16  |
| Senior Software Engineer |  12  |
| Owner                    |   9  |
| Software Engineer        |   9  |
...
+-------------+------+
|    Token    | Freq |
+-------------+------+
| Engineer    |  44  |
| Software    |  31  |
| Senior      |  27  |
| Manager     |  26  |
| Chief       |  24  |
| Officer     |  12  |
| Director    |  11  |
...
```

O que há de mais interessante é que os resultados mostram que o cargo mais usual baseado em correspondências exatas é o de "Chief Executive Officer", enquanto o token mais comum é "Engineer". Ainda que "Engineer" não seja um token presente no cargo mais comum, ele de fato surge no nome de muitos cargos (por exemplo, "Senior Software Engineer" e "Software Engineer", próximos ao topo da lista de cargos). Na

análise de dados referentes a cargos ou a catálogos de endereços, este é precisamente o tipo de insight que justifica o uso de um algoritmo de correspondência aproximada ou de agrupamento. A próxima seção continua nossa investigação.

Métricas usuais de similaridade para agrupamentos

A decisão mais importante quando tomamos um conjunto de strings – indicando cargos neste caso – e agrupamos esses dados de forma útil é decidir qual métrica de similaridade deverá ser utilizada. Há muitas métricas de similaridade de strings disponíveis, e escolher a mais apropriada depende muito da natureza de seu objetivo. Veja a seguir algumas das métricas de similaridade mais usuais que podem ser úteis na comparação de cargos:

Distância de edição

A distância de edição, também conhecida como distância de Levenshtein (*http://en.wikipedia.org/wiki/Vladimir_Levenshtein*), é uma medição simples de quantas inserções, exclusões e substituições são necessárias para converter uma string em outra. Por exemplo, o custo da conversão de "dad" (pai, em inglês) para "bad" (mau, em inglês) seria de apenas uma operação de substituição, e resultaria em um valor de 1. O NLTK fornece uma implementação desse recurso por meio da função `nltk.metrics.distance.edit_distance`. Note que a distância de edição de fato entre duas strings é muitas vezes diferente do número de operações necessárias para computá-la; a computação da distância de edição geralmente está na ordem de operações *M**N* para strings de comprimento *M* e *N*.

Similaridade n-grama

Um *n*-grama é apenas uma forma concisa de expressar cada sequência consecutiva possível de *n* tokens de um texto, e fornece a estrutura de dados que serve de base para computação de colocações. Consulte a seção "O Buzz e os bigramas", no capítulo 7, para uma discussão mais estendida sobre *n*-gramas e colocações.

Há muitas variações de similaridade *n*-grama, mas considere o caso simples em que computamos todos os bigramas possíveis para os

tokens de duas strings e, depois, pontuamos a similaridade entre as strings contando os números de bigramas comuns que elas apresentam (Exemplo 6.4).

Exemplo 6.4 – Uso do NLTK para computar bigramas no interpretador

```
>>> ceo_bigrams = nltk.bigrams("Chief Executive
Officer".split(), pad_right=True, \
... pad_left=True)
>>> ceo_bigrams
[(None, 'Chief'), ('Chief', 'Executive'), ('Executive',
'Officer'),
('Officer', None)]
>>> cto_bigrams = nltk.bigrams("Chief Technology
Officer".split(), pad_right=True, \
... pad_left=True)
>>> cto_bigrams
[(None, 'Chief'), ('Chief', 'Technology'), ('Technology',
'Officer'),
 ('Officer', None)]
>>> len(set(ceo_bigrams).intersection(set(cto_bigrams)))
2
```

Note que o uso dos argumentos de palavra-chave `pad_right` e `pad_left` permite intencionalmente a correspondência de tokens no início e no fim dos textos. Esse enchimento permite que surjam bigramas como (None, 'Chief'), correspondências importantes em títulos de cargos. O NLTK oferece muitas funções de pontuação para bigramas e trigramas, por meio das classes `BigramAssociationMeasures` e `TrigramAssociationMeasures` definidas no módulo `nltk.metrics.association`.

Distância de Jaccard

A métrica de Jaccard expressa a similaridade entre dois conjuntos e é definida por |*Conjunto1 intersecção Conjunto2*/|*Conjunto 1 união Conjunto2*|. Em outras palavras, ela se refere ao número de itens em comum, dividido pelo número total de itens distintos em dois conjuntos. Consulte a seção "Como se faz o mix das colocações: tabelas de contingência e funções de pontuação", no capítulo 7, para uma discussão mais aprofundada. Em geral, você computará a

similaridade de Jaccard utilizando *n*-gramas, incluindo unigramas (tokens individuais), para medir a similaridade de duas strings. Uma implementação da distância de Jaccard está disponível em `nltk.metrics.distance.jaccard_distance` e pode ser implementada da seguinte maneira:

```
len(X.union(Y)) - len(X.intersection(Y)))/float(len(X.union(Y))
```

onde `X` e `Y` são conjuntos de itens.

Distância de MASI

A métrica da distância de MASI[7] é uma versão ponderada da similaridade de Jaccard que ajusta a pontuação para resultar em uma distância menor quando existe uma sobreposição parcial dos conjuntos. A distância de MASI é definida no NLTK como `nltk.metrics.distance.masi_distance`, que pode ser implementado da seguinte maneira:

```
    1 - float(len(X.intersection(Y)))/float(max(len(X),len(Y)))
```

onde `X` e `Y` são conjuntos de itens. O exemplo 6.5 e os resultados de exemplo da tabela 6.2 fornecem pontos de dados que podem ajudá-lo a melhor compreender essa métrica.

Exemplo 6.5 – Uso de métricas integradas de distância do NLTK para comparar pequenos conjuntos de itens (linkedin___distances.py)

```
# -*- coding: utf-8 -*-
from nltk.metrics.distance import jaccard_distance,
masi_distance
from prettytable import PrettyTable
fields = ['X', 'Y', 'Jaccard(X,Y)', 'MASI(X,Y)']
pt = PrettyTable(fields=fields)
[pt.set_field_align(f, 'l') for f in fields]
for z in range(4):
    X = set()
    for x in range(z, 4):
        Y = set()
        for y in range(1, 3):
            X.add(x)
            Y.add(y)
            pt.add_row([list(X), list(Y),
round(jaccard_distance(X, Y), 2),
```

```
                                round(masi_distance(X, Y), 2)])
pt.printt()
```

Note que sempre que os dois conjuntos forem completamente disjuntos ou iguais – por exemplo, quando um dos conjuntos estiver totalmente subsumido[8] por outro – a distância de MASI resultará no mesmo valor da distância de Jaccard. Entretanto, quando os conjuntos estiverem sobrepostos apenas parcialmente, a distância de MASI será maior que a de Jaccard. Vale a pena avaliar se você acredita que a distância de MASI será mais efetiva do que a de Jaccard para pontuar dois cargos arbitrários e agrupá-los, caso sua similaridade (a distância entre eles) exceda certo limite. A tabela 6.2 pretende facilitar sua compreensão de como essas duas métricas se comparam.

Tabela 6.2 – Comparação das distâncias de MASI e de Jaccard para dois pequenos conjuntos de itens, como emitidos no exemplo 6.5

X	Y	Jaccard (X,Y)	MASI (X,Y)
[0]	[1]	1.0	1.0
[0]	[1, 2]	1.0	1.0
[0, 1]	[1]	0.5	0.5
[0, 1]	[1, 2]	0.67	0.5
[0, 1, 2]	[1]	0.67	0.67
[0, 1, 2]	[1, 2]	0.33	0.33
[0, 1, 2, 3]	[1]	0.75	0.75
[0, 1, 2, 3]	[1, 2]	0.5	0.5
[1]	[1]	0.0	0.0
[1]	[1, 2]	0.5	0.5
[1, 2]	[1]	0.5	0.5
[1, 2]	[1, 2]	0.0	0.0
[1, 2, 3]	[1]	0.67	0.67
[1, 2, 3]	[1, 2]	0.33	0.33
[2]	[1]	1.0	1.0
[2]	[1, 2]	0.5	0.5
[2, 3]	[1]	1.0	1.0
[2, 3]	[1, 2]	0.67	0.5
[3]	[1]	1.0	1.0
[3]	[1, 2]	1.0	1.0

Ainda que a lista de métricas interessantes possa ser muito mais extensa,

há literalmente um oceano de textos sobre elas on-line. É muito simples consultar esses recursos e experimentar várias heurísticas de similaridade, tão logo você compreenda melhor os dados que estiver minerando. A próxima seção prepara um script que agrupa cargos utilizando a métrica de similaridade de MASI.

Uma abordagem gulosa aos agrupamentos

Dada a predileção da distância de MASI pelos termos parcialmente sobrepostos, e as informações que vimos sugerindo que a sobreposição nos títulos é importante, vamos tentar agrupar títulos de cargos, comparando uns aos outros, utilizando a distância de MASI como uma extensão do exemplo 6.2. O exemplo 6.6 agrupa títulos semelhantes e então exibe seus contatos de acordo. Analise o código – especialmente o loop aninhado, invocando a função `DISTANCE` que decide pela opção gulosa – e depois acompanhe a discussão.

Exemplo 6.6 – Agrupamento de cargos utilizando uma heurística gulosa (linkedin__cluster_contacts_by_title.py)

```
# -*- codficação: utf-8 -*-
import sys
import csv
from nltk.metrics.distance import masi_distance
CSV_FILE = sys.argv[1]
DISTANCE_THRESHOLD = 0.34
DISTANCE = masi_distance
def cluster_contacts_by_title(csv_file):
    transforms = [
        ('Sr.', 'Senior'),
        ('Sr', 'Senior'),
        ('Jr.', 'Junior'),
        ('Jr', 'Junior'),
        ('CEO', 'Chief Executive Officer'),
        ('COO', 'Chief Operating Officer'),
        ('CTO', 'Chief Technology Officer'),
        ('CFO', 'Chief Finance Officer'),
        ('VP', 'Vice President'),
        ]
    seperators = ['/', 'and', '&']
```

```
    csvReader = csv.DictReader(open(csv_file), delimiter=',',
quotechar='"')
    contacts = [row for row in csvReader]
# Normaliza e/ou substitui abreviações conhecidas e constrói uma
lista de títulos usuais
    all_titles = []
    for i in range(len(contacts)):
        if contacts[i]['Job Title'] == '':
            contacts[i]['Job Titles'] = ['']
            continue
        titles = [contacts[i]['Job Title']]
        for title in titles:
            for seperator in seperators:
                if title.find(seperator) >= 0:
                    titles.remove(title)
                    titles.extend([title.strip() for title in
title.split(seperator)
                                   if title.strip() != ''])
        for transform in transforms:
           titles = [title.replace(*transform) for title in
titles]
        contacts[i]['Job Titles'] = titles
        all_titles.extend(titles)
all_titles = list(set(all_titles))
clusters = {}
    for title1 in all_titles:
        clusters[title1] = []
        for title2 in all_titles:
            if title2 in clusters[title1] or
clusters.has_key(title2) and title1 \
                in clusters[title2]:
                continue
            distance = DISTANCE(set(title1.split()),
set(title2.split()))
            if distance < DISTANCE_THRESHOLD:
                clusters[title1].append(title2)
# Planifica os agrupamentos (clusters)
    clusters = [clusters[title] for title in clusters if
len(clusters[title]) > 1]
# Reúne os contatos que estão nesses agrupamentos, agrupando-os
    clustered_contacts = {}
```

```
    for cluster in clusters:
        clustered_contacts[tuple(cluster)] = []
        for contact in contacts:
            for title in contact['Job Titles']:
                if title in cluster:
                    clustered_contacts[tuple(cluster)].append('%s
%s.'
                            % (contact['First Name'],
contact['Last Name'][0]))
    return clustered_contacts
if __name__ == '__main__':
    clustered_contacts = cluster_contacts_by_title(CSV_FILE)
    for titles in clustered_contacts:
        common_titles_heading = 'Common Titles: ' + ',
'.join(titles)
        print common_titles_heading
        descriptive_terms = set(titles[0].split())
        for title in titles:
            descriptive_terms.intersection_update(set(title.split
()))
        descriptive_terms_heading = 'Descriptive Terms: ' \
            + ', '.join(descriptive_terms)
        print descriptive_terms_heading
        print '-' * max(len(descriptive_terms_heading),
len(common_titles_heading))
        print '\n'.join(clustered_contacts[titles])
        print
```

O código inicia separando títulos combinados utilizando uma lista de conjunções comuns e, então, normaliza títulos usuais. Depois, um loop aninhado itera todos os títulos, agrupando-os de acordo com uma métrica limite de similaridade de MASI. Nesse loop ocorre a maioria das operações do código: é nele que cada título é comparado aos outros. Caso a distância entre dois títulos quaisquer, como determinada por uma heurística de similaridade, seja "suficientemente próxima", nós *gulosamente* os agrupamos. Nesse contexto, ser "guloso" significa que, na primeira oportunidade em que somos capazes de determinar que um item pode se encaixar em um agrupamento, seguimos em frente e o atribuímos, sem qualquer consideração referente à presença de uma combinação melhor, ou tentativa de considerar tal combinação, caso ela se verifique futuramente. Mesmo sendo incrivelmente pragmática, tal

abordagem produz resultados muito razoáveis. Nitidamente, escolher uma heurística de similaridade eficaz é essencial, mas, dada a natureza do loop aninhado, quanto menos vezes tivermos de invocar a função de pontuação, melhor. Falaremos mais sobre essa consideração na próxima seção, mas note que utilizamos um pouco de lógica condicional para tentar evitar a repetição de cálculos desnecessários sempre que possível.

O restante do código simplesmente consulta contatos com cargo específico, agrupando-os para exibição, mas há uma nuance interessante envolvida na computação dos agrupamentos: *você muitas vezes tem de atribuir a cada agrupamento um rótulo significativo*. A implementação que mostramos computa rótulos, verificando a intersecção apresentada por termos nos títulos dos cargos para cada agrupamento, o que parece razoável, considerando que essa é a ligação comum mais óbvia. Seus resultados certamente podem variar utilizando outras abordagens.

Os tipos de resultados que você pode esperar deste código são úteis no sentido de que agrupam indivíduos que provavelmente compartilham responsabilidades semelhantes em seus cargos. Essa informação pode ser útil por muitos motivos, quer você esteja planejando um evento que inclua um "Painel de CEOs", tentando determinar quem melhor pode ajudá-lo a tomar sua próxima decisão na carreira, ou simplesmente tentando determinar se você está *realmente* bem conectado a outros profissionais semelhantes, considerando suas próprias responsabilidades. Resultados resumidos para uma rede de exemplo são apresentados no exemplo 6.7.

Exemplo 6.7 – Exemplo de resultado para o exemplo 6.6

```
Common Titles: Chief Executive Officer,
               Chief Finance Officer,
               Chief Technology Officer,
               Chief Operations Officer
Descriptive Terms: Chief, Officer
----------------------------------------
Gregg P.
Scott S.
Mario G.
Brian F.
Damien K.
Chris A.
Trey B.
```

```
Emil E.
Dan M.
Tim E.
John T. H.
...
Common Titles: Consultant,
               Software Engineer,
               Software Engineer
Descriptive Terms: Software, Engineer
----------------------------------------
Kent L.
Paul C.
Rick C.
James B.
Alex R.
Marco V.
James L.
Bill T.
...
```

Agrupar com escalabilidade não é nada fácil

É importante notar que, *no pior caso*, o loop aninhado que executa o cálculo de `DISTANCE` do exemplo 6.6 exigiria que o cálculo fosse invocado no que já mencionamos ser uma complexidade de tempo $O(n^2)$ – em outras palavras, o cálculo seria invocado `len(all_titles)*len(all_titles)` vezes. Um loop aninhado que compara cada item individual para realizar os agrupamentos não é uma abordagem de boa escalabilidade[9] quando temos um valor muito grande de n, mas considerando que o número de cargos individuais de sua rede profissional provavelmente não será grande demais, isso não deve impor uma limitação de desempenho. Talvez esses detalhes não pareçam tão importantes – afinal, estamos falando simplesmente de um loop aninhado – mas o ponto crucial de um algoritmo $O(n^2)$ está no fato de que o número de comparações necessárias para processar um conjunto de entrada cresce exponencialmente em proporção ao número de itens do conjunto. Por exemplo, um pequeno conjunto de entrada de 100 cargos demandaria 10.000 operações de pontuação, enquanto 10.000 cargos demandariam 100.000.000 de operações de pontuação. A matemática dessas operações não produz os resultados pretendidos e eventualmente

fracassa, mesmo quando você tem hardware suficiente à disposição.

Sua reação inicial quando enfrenta o que parece ser um problema de escalabilidade exponencial provavelmente será tentar reduzir ao máximo o valor de n. Porém, na maioria dos casos, você não será capaz de reduzi-lo suficientemente para tornar sua solução escalável conforme cresce o volume dos dados de sua entrada, uma vez que você continuará trabalhando com um algoritmo $O(n^2)$. O que você realmente deve fazer é descobrir um modo de formular um algoritmo que seja da ordem de $O(k*n)$, em que k é muito menor que n e representa uma quantidade administrável de overhead, que se eleva muito mais lentamente do que a taxa de crescimento de n. Assim como em qualquer outra decisão de engenharia, há questões de desempenho e qualidade que deverão ser consideradas referentes a todos os aspectos de sua situação, e *não é nada fácil* alcançar o equilíbrio correto. De fato, muitas empresas de mineração de dados que conseguiram implementar com êxito análises escaláveis de correspondência de registros a um grau elevado de fidelidade consideram suas abordagens específicas informações proprietárias (ou seja, segredos de negócio), pois resultam em vantagens comerciais evidentes (muitas vezes, não é a habilidade de solucionar um problema o que realmente importa, mas a capacidade de implementar essa solução de modo específico e comercializável).

Para as situações nas quais um algoritmo $O(n^2)$ é simplesmente inaceitável, uma variação que você pode experimentar é reescrever os loops aninhados selecionando uma amostra randômica para a função de pontuação, o que efetivamente reduziria a complexidade para $O(k*n)$, se k fosse o tamanho da amostra. Todavia, conforme esse valor se aproxima de n, você pode esperar que o tempo de execução se aproxime do verificado em $O(n^2)$. O exemplo 6.8 exibe qual seria o visual dessa técnica de amostragem no código; as principais alterações ao código anterior estão destacadas em negrito. O resultado principal é que, para cada invocação do loop externo, estamos executando o loop interno um número muito menor e fixo de vezes.

Exemplo 6.8 – Pequena alteração, feita em uma parte do exemplo 6.6, que incorpora amostragem randômica para melhorar o desempenho

```
# ...início do trecho ...
all_titles = list(set(all_titles))
```

```
clusters = {}
for title1 in all_titles:
    clusters[title1] = []
   for sample in range(SAMPLE_SIZE):
       title2 = all_titles[random.randint(0, len(all_titles)-1)]
       if title2 in clusters[title1] or clusters.has_key(title2)
and title1 \
          in clusters[title2]:
          continue
       distance = DISTANCE(set(title1.split()),
set(title2.split()))
       if distance < DISTANCE_THRESHOLD:
       clusters[title1].append(title2)
# ...fim do trecho ...
```

Outra abordagem que pode ser considerada é a reunião de uma amostragem randômica de dados em *n* compartimentos (onde *n* é algum número geralmente menor ou igual à raiz quadrada do número de itens de seu conjunto). Depois, você realizaria agrupamentos dentro de cada um desses compartimentos individuais, e então opcionalmente faria o *merge* (ou a fusão) da saída. Por exemplo, se você tivesse um milhão de itens, um algoritmo $O(n^2)$ demandaria um trilhão de operações lógicas; por outro lado, se dividíssemos esses itens em 1.000 compartimentos contendo 1.000 itens cada e agrupássemos cada compartimento individual, seriam necessárias apenas um bilhão de operações (isso significa 1.000*1.000 comparações para cada compartimento para todos os 1.000 compartimentos). Um bilhão ainda é um número considerável, mas três ordens de grandeza menor que um trilhão, o que não pode ser desprezado.

Há muitas outras abordagens na literatura da área além das que utilizam amostragem e compartimentos e que poderiam ser muito melhores para redução da dimensionalidade de um problema. Por exemplo, idealmente você deve comparar todos os itens de um conjunto e, em última análise, a técnica escolhida para evitar uma situação de $O(n^2)$ para um grande valor de *n* dependerá de características específicas de seu caso e de insights obtidos por meio de experimentação e de conhecimentos particulares à sua situação. Conforme considera as possibilidades, lembre-se de que o campo da aprendizagem de máquina oferece muitas técnicas projetadas para combater exatamente esses tipos de problemas de escala, utilizando

diversos modelos probabilísticos e técnicas sofisticadas de amostragem. A próxima seção apresenta um algoritmo de agrupamento intuitivo e relativamente popular conhecido como algoritmo das *k*-médias, uma abordagem de uso geral para agrupamento de um espaço multidimensional. Utilizaremos essa técnica futuramente para agrupar seus contatos por localização geográfica.

Agrupamentos inteligentes permitem experiências de usuário atrativas

É tão fácil deixar-se envolver pelas técnicas de mineração de dados que até esquecemos do caso prático que nos serviu de motivação para adotá-las. Técnicas simples de agrupamento podem criar experiências de usuário incrivelmente atrativas (em muitos aspectos), produzindo resultados tão simples quanto os títulos de cargos que acabamos de analisar. A figura 6.2 demonstra uma visualização alternativa muito poderosa de seus dados por meio de um simples widget de árvore que poderia ser utilizado como parte de um painel de navegação ou display facetado para filtrar critérios de busca. Presumindo que as métricas de similaridade escolhidas tenham produzido agrupamentos significativos, um simples display hierárquico, que apresente os dados em grupos lógicos com contagem dos itens em cada grupo, pode otimizar o processo da localização de informações. Isso permitiria fluxos de trabalho intuitivos para praticamente todas as aplicações em que a análise de muitas informações fosse necessária na procura por resultados.

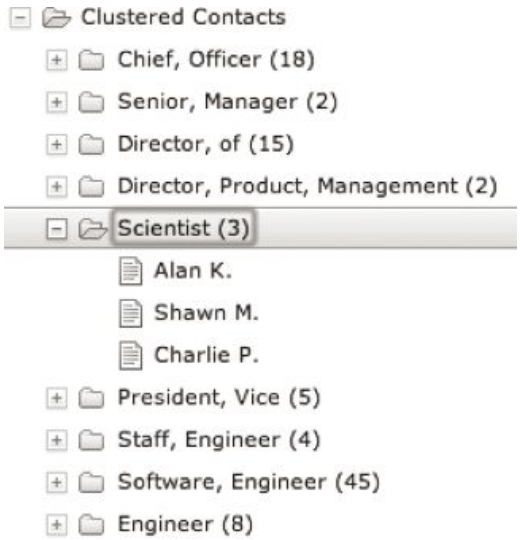


Figura 6.2 – Agrupamento inteligente dos dados possibilita displays facetados e experiências de usuário atrativas.

O código para criação de um simples display navegacional pode ser surpreendentemente simples, considerando a maturidade dos kits de ferramentas Ajax e de outras bibliotecas de UI. O exemplo 6.9 mostra uma simples página web suficiente para preencher o widget de árvore Dojo (*http://docs.dojocampus.org/dijit/Tree*) da figura 6.2 utilizando um template provisório.

Exemplo 6.9 – Template para um widget de árvore Dojo, como o da figura 6.2 – adicione os dados e veja como ele funciona

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>Clustered Contacts</title>
    <link rel="stylesheet"
        href="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/
resources/dojo.css">
    <link rel="stylesheet"
        href="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dijit
/themes/claro/claro.css">
    <script
src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.j
s"
        type="text/javascript" djConfig="parseOnLoad:true">
</script>
    <script language="JavaScript" type="text/javascript">
        dojo.require("dojo.data.ItemFileReadStore");
        dojo.require("dijit.Tree");
        dojo.require("dojo.parser");
    </script>
    <script language="JavaScript" type="text/javascript">
        var data = %s; //substituted by Python
    </script>
</head>
<body class="claro">
    <div dojoType="dojo.data.ItemFileReadStore" jsId="jobsStore"
        data="data"></div>
<div dojoType="dijit.Tree" id="mytree" store="jobsStore"
    label="Clustered Contacts" openOnClick="true">
        </div>
    </body>
```

```
</html>
```

O valor `data` substituído no template é uma simples estrutura JSON, que podemos obter com facilidade modificando a saída do exemplo 6.6, como mostra o exemplo 6.10.

Exemplo 6.10 – Transformação dos dados de clustered_contacts computados no exemplo 6.6 para produzir JSON que possa ser consumido pelo widget de árvore Dojo

```
import json
import webbrowser
import os

data = {"label" : "name", "temp_items" : {}, "items" : []}
for titles in clustered_contacts:
    descriptive_terms = set(titles[0].split())
    for title in titles:
        descriptive_terms.intersection_update(set(title.split()))
    descriptive_terms = ', '.join(descriptive_terms)

    if data['temp_items'].has_key(descriptive_terms):
        data['temp_items'][descriptive_terms].extend([{'name' :
cc } for cc
    in clustered_contacts[titles]])
    else:
        data['temp_items'][descriptive_terms] = [{'name' : cc }
for cc
            in clustered_contacts[titles]]

for descriptive_terms in data['temp_items']:
    data['items'].append({"name" : "%s (%s)" %
(descriptive_terms,
        len(data['temp_items'][descriptive_terms]),),
                         "children" : [i for i in
                          data['temp_items']
[descriptive_terms]]})

del data['temp_items']

# Abre o template e substitui os dados

if not os.path.isdir('out'):
    os.mkdir('out')

TEMPLATE = os.path.join(os.getcwd(), '..', 'web_code', 'dojo',
'dojo_tree.html')
OUT = os.path.join('out', 'dojo_tree.html')

t = open(TEMPLATE).read()
f = open(OUT, 'w')
```

```
    f.write(t % json.dumps(data, indent=4))
    f.close()
    webbrowser.open("file://" + os.path.join(os.getcwd(), OUT))
```

Há tremendo valor na capacidade de criar experiências de usuário que apresentem dados de forma intuitiva para estimular fluxos de trabalho. Algo tão simples quanto um display hierárquico criado de forma inteligente pode inadvertidamente motivar os usuários a passarem mais tempo em um site, descobrindo mais informações do que normalmente descobririam e, em última análise, reconhecendo mais valor nos serviços oferecidos.

Agrupamentos hierárquicos e das k-médias

O exemplo 6.6 apresentou uma abordagem gulosa para agrupamentos, mas há ao menos duas outras técnicas usuais que você deve conhecer: a dos agrupamentos hierárquicos e a dos agrupamentos das *k*-médias. Agrupamentos hierárquicos são superficialmente semelhantes à heurística gulosa que estamos utilizando, enquanto o agrupamento das *k*-médias é radicalmente diferente. Trataremos principalmente das *k*-médias no restante deste capítulo, mas vale a pena introduzir brevemente a teoria que fundamenta ambas essas abordagens, uma vez que você provavelmente as encontrará ao revisar a literatura da área e realizar suas pesquisas. Uma excelente implementação de ambas está disponível por meio do módulo `cluster` que pode ser instalado com o `easy_install`, por isso não tenha pressa e faça uma pausa para realizar essa tarefa.

Agrupamentos hierárquicos

Agrupamentos hierárquicos são uma técnica determinística e exaustiva, no sentido de que computam a matriz completa das distâncias entre todos os itens e depois percorrem essa matriz agrupando itens que correspondam a um limite mínimo de distância. São *hierárquicos* no sentido de que, ao percorrerem a matriz e agruparem itens, naturalmente produzem uma estrutura de árvore que expressa distâncias relativas entre os itens. Na literatura, você poderá encontrar essa técnica referenciada como *aglomerativa*, pois as folhas dessa árvore representam os itens sendo agrupados, enquanto os nós aglomeram esses itens em agrupamentos.

Essa técnica é semelhante, mas não fundamentalmente idêntica, à

abordagem utilizada no exemplo 6.6, que utiliza heurística gulosa para agrupar itens, em vez de construir sucessivamente uma hierarquia. Nesse sentido, o tempo real gasto (para que o código execute) em agrupamentos hierárquicos pode ser um pouco maior, e talvez você tenha de ajustar sua função de pontuação e seu limite de distância. Entretanto, o uso de programação dinâmica (*http://en.wikipedia.org/wiki/Dynamic_programming*), e de outras técnicas inteligentes de computação, pode resultar em uma economia significativa do tempo de execução, dependendo da implementação. Por exemplo, dado que a distância entre dois itens, como títulos de cargos, é quase certamente simétrica, você terá de computar somente uma metade da matriz de distância em vez da matriz inteira. Mesmo que a complexidade de tempo do algoritmo como um todo ainda seja $O(n^2)$, apenas $0{,}5*n^2$ unidades de trabalho estão sendo realizadas, em vez de n^2 unidades.

Se reescrevêssemos o exemplo 6.6 para utilizar o módulo `cluster`, o loop aninhado que realiza a computação de `DISTANCE` seria substituído por algo semelhante ao código do exemplo 6.11.

Exemplo 6.11 – Pequena modificação ao exemplo 6.6 que utiliza cluster.HierarchicalClustering em vez de uma heurística gulosa (linkedin___cluster_contacts_by_title_hac.py)

```
# ... início do trecho ...
# Importa os dados
from cluster import HierarchicalClustering
# Define uma função de pontuação
def score(title1, title2):
    return DISTANCE(set(title1.split()), set(title2.split()))
# Alimenta a classe com seus dados e a função de pontuação
hc = HierarchicalClustering(all_titles, score)
# Agrupa os dados de acordo com um limite de distância
clusters = hc.getlevel(DISTANCE_THRESHOLD)
# Remove agrupamentos singletons
clusters = [c for c in clusters if len(c) > 1]
# ... fim do trecho ...
```

Caso você esteja muito interessado em variações de agrupamentos hierárquicos, não deixe de verificar o método `setLinkageMethod` da classe `HierarchicalClustering`, que fornece algumas variações sutis na forma como a classe pode computar distâncias entre agrupamentos. Por

exemplo, você pode especificar se as distâncias entre os agrupamentos devem ser determinadas calculando a distância mais curta, a mais longa ou a média entre dois agrupamentos. Dependendo da distribuição de seus dados, a escolha de outro método de ligação pode potencialmente produzir resultados bem diferentes. A figura 6.3 mostra uma seção de uma rede profissional como um layout de árvore radial e um dendograma utilizando o Protovis (*http://vis.stanford.edu/protovis/*), um poderoso kit de ferramentas de visualização em HTML5. O layout de árvore radial é mais eficiente em termos de espaço e provavelmente uma escolha mais adequada para esse conjunto de dados específico, mas um dendograma (*http://en.wikipedia.org/wiki/Dendrogram*) seria uma ótima escolha se você quisesse encontrar facilmente correlações entre cada nível da árvore para um conjunto de dados mais complexo (o que corresponderia a cada nível de aglomeração em agrupamentos hierárquicos). Ambas as visualizações apresentadas aqui são essencialmente apenas representações estáticas do widget de árvore interativa da figura 6.2. Como elas mostram uma quantidade incrível de informações torna-se aparente quando você é capaz de visualizar uma simples imagem de sua rede profissional[10].

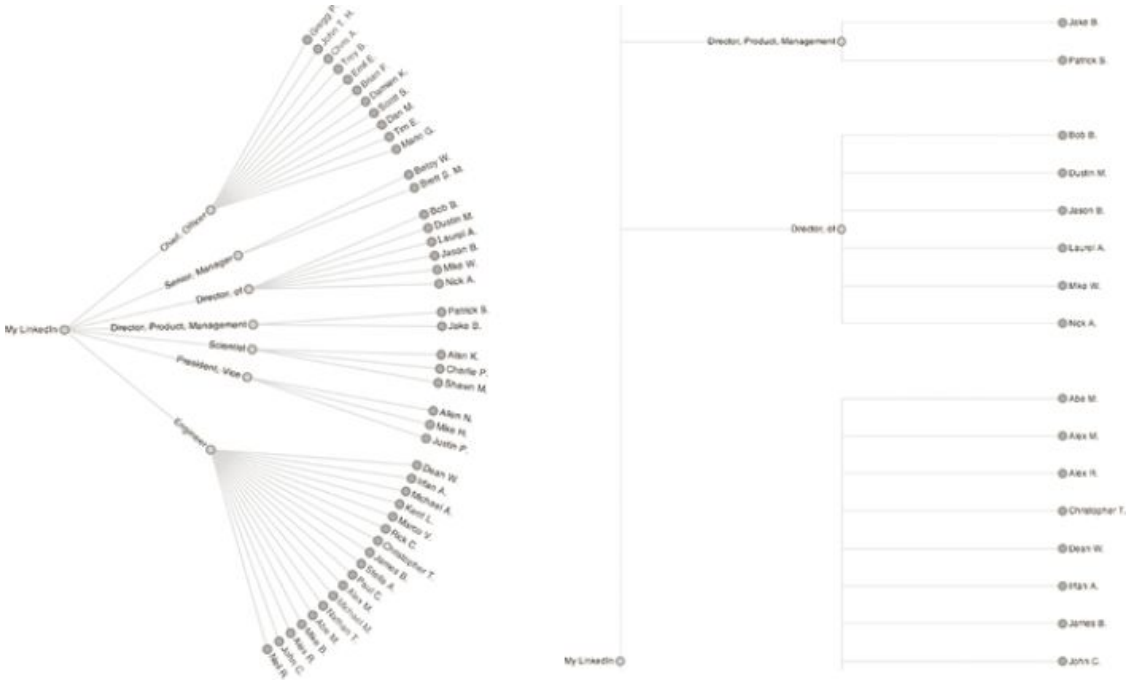


Figura 6.3 – Layout de árvore radial do Protovis de contatos agrupados por cargos ocupados (à esquerda) e uma seção dos mesmos dados apresentada como um dendograma (à direita).

Agrupamentos por k-médias

Enquanto os agrupamentos hierárquicos são uma técnica determinística e uma abordagem de tipo O(n^2), que esgota possibilidades, agrupamentos por *k*-médias, por outro lado, geralmente são executados na ordem de O(k^*n) vezes. A economia significativa de desempenho é obtida ao custo de resultados apenas aproximados, mas ainda potencialmente muito valiosos. O princípio é que você geralmente tem um espaço multidimensional contendo *n* pontos, agrupados em *k* agrupamentos por meio da seguinte série de passos:

1. Selecione randomicamente *k* pontos no espaço de dados como valores iniciais que serão utilizados para computar os *k* agrupamentos: *K*1, *K*2, ..., *K*k.
2. Atribua cada um dos *n* pontos a um agrupamento, encontrando o *K*n mais próximo – efetivamente criando *k* agrupamentos e exigindo k^*n comparações.
3. Para cada um dos *k* agrupamentos, calcule o centroide (*http://en.wikipedia.org/wiki/Centroid*), ou a média do agrupamento, e reatribua seu valor *K*i para ser esse valor. Portanto, você está computando "*k*-médias" durante cada iteração do algoritmo.
4. Repita os passos 2–3 até que os membros dos agrupamentos não se alterem entre as iterações. Em termos gerais, são necessárias relativamente poucas iterações para que haja convergência.

Como à primeira vista as *k*-médias podem não parecer muito intuitivas, a figura 6.4 exibe cada passo do algoritmo como apresentado no "Tutorial on Clustering Algorithms" (*http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/AppletKM.html*), que pode ser encontrado on-line, e que apresenta um applet[11] Java interativo. Os parâmetros de amostra utilizados envolvem 100 pontos de dados e um valor de 3 para o parâmetro *k*, o que significa que o algoritmo produzirá três agrupamentos. O mais importante em cada passo é perceber a localização dos quadrados, e quais pontos são incluídos em cada um desses três agrupamentos à medida que o algoritmo avança. São necessários apenas nove passos para que o algoritmo conclua.

Ainda que fosse possível executar *k*-médias em pontos em duas dimensões ou em duas mil dimensões, o intervalo mais costumeiro que encontramos está geralmente na ordem das dezenas, com casos mais usuais de duas ou três dimensões. Quando a dimensionalidade do espaço

em que você trabalha é relativamente pequena, a utilização das *k*-médias pode ser uma técnica de agrupamento efetiva, pois é de rápida execução e capaz de produzir resultados muito razoáveis. Você deve, entretanto, escolher um valor apropriado para *k*, o qual nem sempre é óbvio. A próxima seção demonstra como buscar informações estendidas de perfil para seus contatos, resumindo essa discussão em um exercício de acompanhamento que trata do agrupamento geográfico de sua rede profissional empregando *k*-médias.

Busca de informações estendidas de perfil

Ainda que os exercícios anteriores deste capítulo tenham sido interessantes, logo você desejará utilizar as APIs do LinkedIn para minerar toda a riqueza dos dados disponíveis. Como seria de se esperar, há um processo de opt-in básico de desenvolvedor no qual você deve se registrar para obter credenciais de autorização; visite *http://developer.linkedin.com* para ler todos os detalhes desse procedimento, mas a figura 6.5 demonstra seus traços mais importantes. Assim que você tiver as credenciais, o processo geral será muito semelhante ao de qualquer outra rede social que exija a "dança do OAuth" para acesso às APIs: você solicita credenciais, recebe um token e um segredo de volta, e então utiliza esses valores para finalmente obter o desejado token de acesso, que pode ser utilizado quando solicitações são enviadas. Evidentemente, a maioria dos detalhes menores será abstraída, pois estamos utilizando o módulo Python `linkedin` para cuidar dos detalhes mais tediosos em nosso nome. Basta executar `easy_install python-linkedin`.

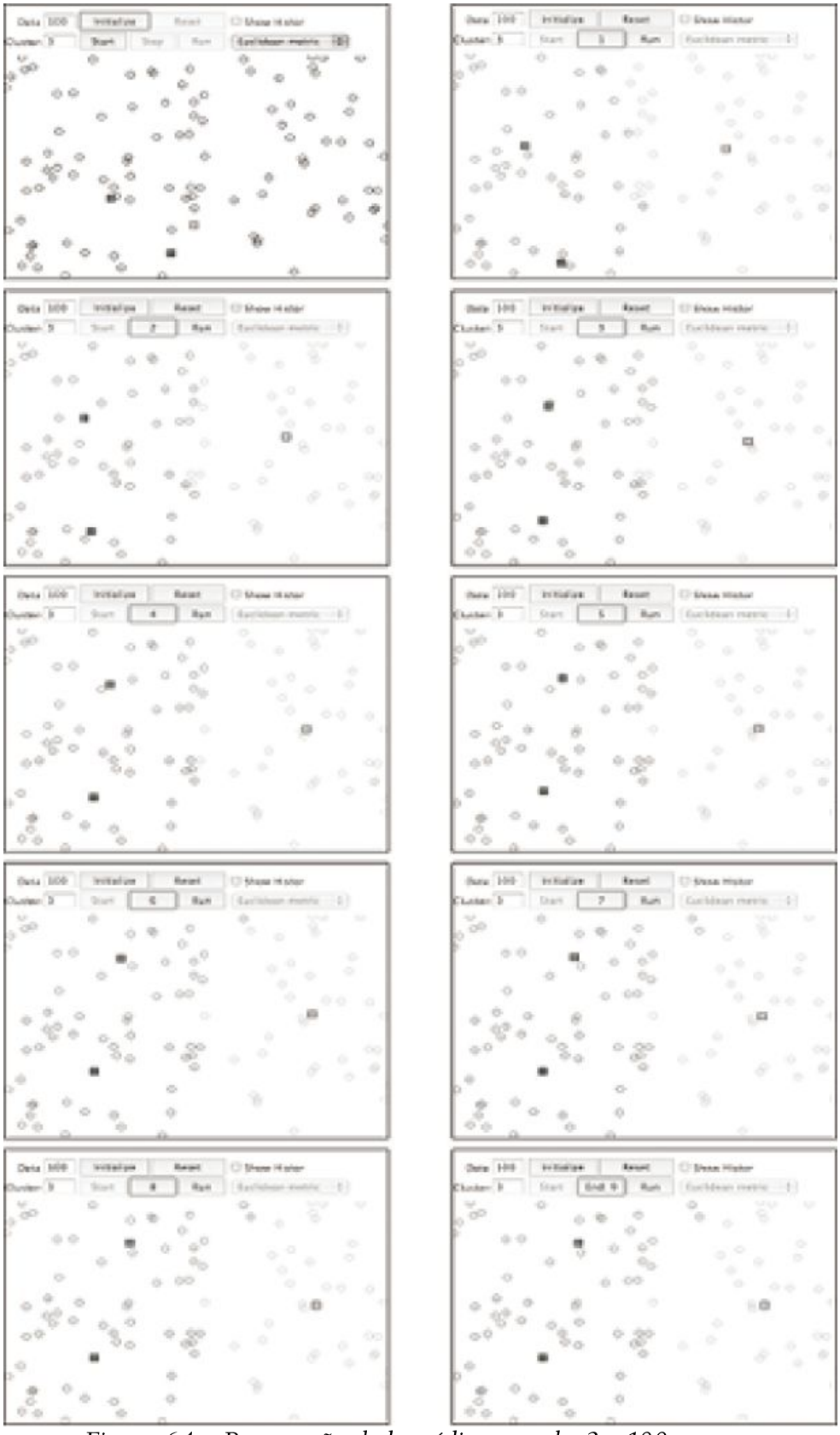


Figura 6.4 – Progressão de k-médias para k=3 e 100 pontos.

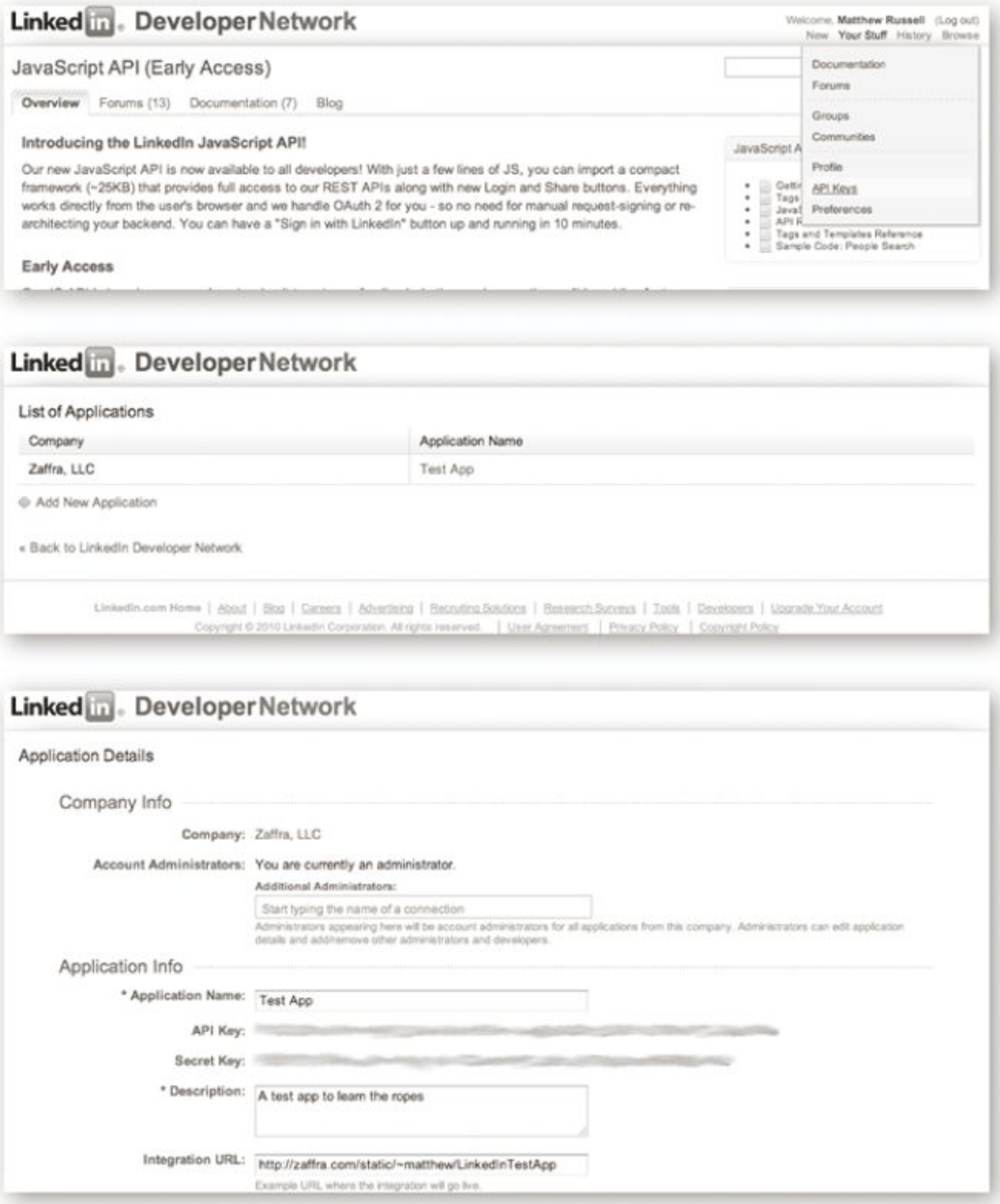


Figura 6.5 – Passos básicos envolvidos na obtenção das credenciais da API do LinkedIn: em http://developer.linkedin.com, escolha "Your Stuff"→"API Keys" (no top); então crie uma nova app (no meio), e finalmente configure seus parâmetros, tomando nota de sua chave para a API e de seu segredo (na base).

Instalado o pacote `linkedin`, o script a seguir (Exemplo 6.12) demonstra um template funcional que você pode utilizar para se conectar e acessar sua rede profissional no nível da API. O pacote `linkedin` ainda é recente, por isso já pode ter sido otimizado quando você estiver lendo este texto[12].

Exemplo 6.12 – Coleta de informações estendidas de perfil para seus contatos no LinkedIn

(linkedin__get_connections.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import webbrowser
import cPickle
from linkedin import linkedin
KEY = sys.argv[1]
SECRET = sys.argv[2]
# Faz o parsing do parâmetro oauth_verifier de
window.location.href, exibindo-o ao usuário
RETURN_URL =
'http://miningthesocialweb.appspot.com/static/linkedin_oauth_help
er.html'
def oauthDance(key, secret, return_url):
    api = linkedin.LinkedIn(key, secret, return_url)
    result = api.requestToken()
    if not result:
        print >> sys.stderr, api.requestTokenError()
        return None
    authorize_url = api.getAuthorizeURL()
    webbrowser.open(authorize_url)
    oauth_verifier = raw_input('PIN number, bro: ')
    result = api.accessToken(verifier=oauth_verifier)
    if not result:
        print >> sys.stderr, 'Error: %s\nAborting' %
api.getRequestTokenError()
        return None
    return api
# Primeiro, efetua a oauth_dance
api = oauthDance(KEY, SECRET, RETURN_URL)
# Agora, faz algo, como obter suas conexões:
if api:
    connections = api.GetConnections()
else:
    print >> sys.stderr, 'Failed to aunthenticate. You need to
learn to dance'
    sys.exit(1)
```

```
# Tenha cuidado - esse tipo de utilização da API é custoso.
# Consulte http://developer.linkedin.com/docs/DOC-1112
print >> sys.stderr, 'Fetching extended connections...'
extended_connections = [api.GetProfile(member_id=c.id, url=None,
fields=[
    'first-name',
    'last-name',
    'current-status',
    'educations',
    'specialties',
    'interests',
    'honors',
    'positions',
    'industry',
    'summary',
    'location',
    ]) for c in connections]
# Armazena os dados
if not os.path.isdir('out'):
    os.mkdir('out')
f = open('out/linkedin_connections.pickle', 'wb')
cPickle.dump(extended_connections, f)
f.close()
print >> sys.stderr, 'Data pickled to
out/linkedin_connections.pickle'
```

Note que, diferentemente do Twitter, o LinkedIn requer que você especifique seu próprio URL para captura do verificador OAuth (o token de acesso) necessário para autorização de um cliente de linha de comando, como o programa Python que estamos executando. O template de página web no exemplo 6.13 está disponível no GitHub (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/web_code/linkedin/linkedin_oauth_helper.html*) e demonstra como extrair essa informação e apresentá-la ao usuário de modo semelhante ao Twitter, caso você esteja curioso.

Exemplo 6.13 – Template de exemplo que extrai o verificador OAuth do LinkedIn, exibindo-o para você

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
```

```
<head>
    <title>LinkedIn OAuth Helper</title>
    <script
src="http://ajax.googleapis.com/ajax/libs/dojo/1.5/dojo/dojo.xd.j
s"
         type="text/javascript"></script>
    <script type="text/javascript">
         dojo.ready(function() {
             var q =
dojo.queryToObject(window.location.search.slice(1));
             dojo.byId("oauth_verifier").innerHTML =
q.oauth_verifier;
         });
</script>
</head>
<body>
    <div>
        Type the following identifier into the prompt in your
terminal:
         <strong><span id="oauth_verifier"></span></strong>
    </div>
</body>
</html>
```

Isso é basicamente tudo que é necessário para que você possa utilizar o pacote `linkedin`, mas há outras considerações importantes para utilização da API. Talvez a mais pertinente seja que os limites de taxa do LinkedIn (*http://developer.linkedin.com/docs/DOC-1112*) são um pouco mais complexos do que os que estamos acostumados a encontrar. As aplicações que você constrói têm um rate limit diário total, mas usuários individuais de suas aplicações também têm rate limits, e até você, como desenvolvedor, tem outro rate limit, na maioria das situações. Além disso, há rate limits diferentes dependendo da API específica, além de rate limits distintos, caso os usuários da aplicação estejam acessando seus próprios dados, ou dados de outras pessoas da rede. Quando você atinge ou excede o rate limit, precisa aguardar até a meia-noite no horário padrão do pacífico (Pacific Standard Time, PST) para zerar o valor. O exemplo 6.12 demonstra como obter os "miniperfis" para usuários via `GetConnections`, e como buscar detalhes completos de perfis passando argumentos para `GetProfile`. Todos esses rate limits não permitem que você acumule grandes quantidades de dados por meio de processos de longa duração,

por isso os exemplos futuros deste capítulo não empregam técnicas como a utilização de um pool de threads para acelerar a busca de dados. Se você é do tipo que gosta de gastar tudo de uma só vez, verá que não demora muito para que seu rate limit seja consumido em cerca de um minuto.

Agrupamento geográfico de sua rede

Com a capacidade de acessar informações estendidas de perfis do LinkedIn, e um conhecimento prático dos algoritmos mais usuais de agrupamento, tudo que falta é apresentar uma visualização adequada que reúna todos esses elementos. A próxima seção aplica as *k*-médias ao problema do agrupamento de seus contatos profissionais, representando-os no Google Earth. Depois, a seção seguinte apresenta uma visualização alternativa, conhecida como Dorling Cartogram (*http://vis.stanford.edu/protovis/ex/cartogram.html*), essencialmente um gráfico de bolha agrupado geograficamente, permitindo que você visualize facilmente quantos de seus contatos vivem em cada estado. Ironicamente, ela não utiliza explicitamente um algoritmo de agrupamento como as *k*-médias, mas ainda produz resultados intuitivos mapeados em um espaço 2D, exibindo uma apresentação de agrupamento geográfico com informações de frequência.

Mapeamento de sua rede profissional com o Google Earth

Um exercício interessante para observar as *k*-médias em ação pode ser utilizá-las para visualizar e agrupar sua rede profissional do LinkedIn, colocando-a em um mapa – ou no globo, se você é fã do Google Earth. Além do insight obtido pela visualização da distribuição de seus contatos, você também pode analisar agrupamentos utilizando como base esses mesmos contatos, os empregadores deles, ou as áreas distintas nas quais essas pessoas residem. Todas as três abordagens podem produzir resultados úteis considerando propósitos diferentes. Por meio da API do LinkedIn, você pode buscar informações de localização que descrevam, por exemplo, a área metropolitana em que reside cada um de seus contatos (como a "área da Grande Nashville"), e que, com um pouco de transformação, sejam adequadas para geocodificação[13] dessas localizações em coordenadas que possam ser representadas utilizando uma ferramenta como o Google Earth.

Os principais passos que devem ser seguidos para colocar isso em prática incluem:

- O parsing da localização geográfica de cada um dos perfis públicos de seus contatos. O exemplo 6.12 demonstra como buscar esse tipo de informação.
- A geocodificação das localizações, transformando-as em coordenadas. A abordagem que escolhemos é executar `easy_install geopy` e deixar que ele cuide de todo o trabalho pesado. Há um ótimo guia para iniciantes disponível on-line (*http://code.google.com/p/geopy/wiki/GettingStarted*); dependendo de sua escolha de geocodificador, talvez você tenha de solicitar uma chave de API para um provedor como o Google ou o Yahoo!.
- A alimentação das geocoordenadas à classe `KMeansClustering` do módulo `cluster` para calcular agrupamentos.
- A construção do arquivo KML que pode ser alimentado a uma ferramenta de visualização como o Google Earth.

Muitas nuances e variações interessantes tornam-se possíveis quando você tem disponível o trabalho básico do exemplo 6.14. O `linkedin__kml_utility` referenciado é muito simples e faz apenas alguma transformação no XML; você pode visualizar os detalhes no GitHub (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/linkedin__kml_utility.py*).

Lembre-se de que vimos no tópico "Representação de dados geo via microform.at e Google Maps", no capítulo 2, que você pode apontar o Google Maps para um URL que aponte para um arquivo KML, caso prefira não fazer o download e utilizar o Google Earth.

Exemplo 6.14 – Geocodificação das localizações de seus contatos do LinkedIn seguida pela exportação desses dados para KML (linkedin__geocode.py)

```
# -*- coding: utf-8 -*-

import os
import sys
import cPickle
from urllib2 import HTTPError
from geopy import geocoders
from cluster import KMeansClustering, centroid

# Uma simples função helper para construção de uma árvore XML
from linkedin__kml_utility import createKML
```

```
K = int(sys.argv[1])
# Utilize aqui sua própria chave da API, caso esteja utilizando
um serviço
# de geocodificação como o Google ou o Yahoo!
GEOCODING_API_KEY = sys.argv[2]
CONNECTIONS_DATA = sys.argv[3]
OUT = "clusters.kmeans.kml"
# Abre suas conexões salvas com informações estendidas de perfil
extended_connections = cPickle.load(open(CONNECTIONS_DATA))
locations = [ec.location for ec in extended_connections]
g = geocoders.Yahoo(GEOCODING_API_KEY)
# Algumas transformações podem ser necessárias para que os
serviços de geocodificação funcionem
# adequadamente. Veja alguns exemplos que parecem causar
problemas no Yahoo. Você provavelmente
# terá de adicionar seus próprios exemplos.
transforms = [('Greater ', ''), (' Area', ''), ('San Francisco
Bay', 'San Francisco')]
# Computa a frequência de cada localização
coords_freqs = {}
for location in locations:
    # Evita operações de I/O desnecessárias
    if coords_freqs.has_key(location):
        coords_freqs[location][1] += 1
        continue
    transformed_location = location
    for transform in transforms:
        transformed_location =
transformed_location.replace(*transform)
        while True:
            num_errors = 0
            try:
                # Esta chamada retorna um gerador
                results = g.geocode(transformed_location,
exactly_one=False)
                break
            except HTTPError, e:
                num_errors += 1
                if num_errors >= 3:
```

```
                    sys.exit()
                print >> sys.stderr, e
                print >> sys.stderr, 'Encountered an urllib2
error. Trying again...'
for result in results:
            # Cada resultado tem a forma ("Description", (X,Y))
            coords_freqs[location] = [result[1], 1]
            break
# Aqui, você poderia opcionalmente segmentar localizações por
continente ou país para evitar a
# potencial localização de uma média no meio do oceano. O
algoritmo das k-médias espera pontos
# distintos para cada contato, por isso construa uma lista
expandida para passá-lo
expanded_coords = []
for label in coords_freqs:
    ((lat, lon), f) = coords_freqs[label]
    expanded_coords.append((label, [(lon, lat)] * f)) # Inverte
lat/lon para o Google Earth
# Não é necessário desorganizar o mapa com marcações
desnecessárias...
kml_items = [{'label': label, 'coords': '%s,%s' % coords[0]} for
(label,
             coords) in expanded_coords]
# Também poderia ser interessante incluir os nomes de seus
contatos no mapa para exibição
for item in kml_items:
    item['contacts'] = '\n'.join(['%s %s.' % (ec.first_name,
ec.last_name[0])
                                  for ec in extended_connections
if ec.location
                                  == item['label']])
cl = KMeansClustering([coords for (label, coords_list) in
expanded_coords
                       for coords in coords_list])
centroids = [{'label': 'CENTROID', 'coords': '%s,%s' %
centroid(c)} for c in
             cl.getclusters(K)]
kml_items.extend(centroids)
kml = createKML(kml_items)
```

```
if not os.path.isdir('out'):
    os.mkdir('out')
f = open("out/" + OUT, 'w')
f.write(kml)
f.close()
print >> sys.stderr, 'Data pickled to out/' + OUT
```

Valores de localização retornados como parte das informações de perfil do LinkedIn geralmente seguem a forma "Greater Nashville Area" (área da Grande Nashville), e é necessária certa quantidade de transformação dos dados para extrair o nome da cidade. A abordagem apresentada aqui é imperfeita, e você pode ter de ajustá-la de acordo com o que ocorre com seus dados para obter total exatidão.

Assim como no exemplo 6.6, a maioria do trabalho necessário para chegar ao ponto em que os dados possam ser visualizados envolve apenas procedimentos básicos de processamento de dados. Os detalhes mais interessantes estão na chamada ao método `getclusters` de `KMeansClustering`, próxima do final do código. A abordagem demonstrada agrupa seus contatos por localização, reúne esses contatos em agrupamentos, e então utiliza os resultados do algoritmo de agrupamento para computar os centroides. A figura 6.6 ilustra resultados de exemplo para execução do código do exemplo 6.14.

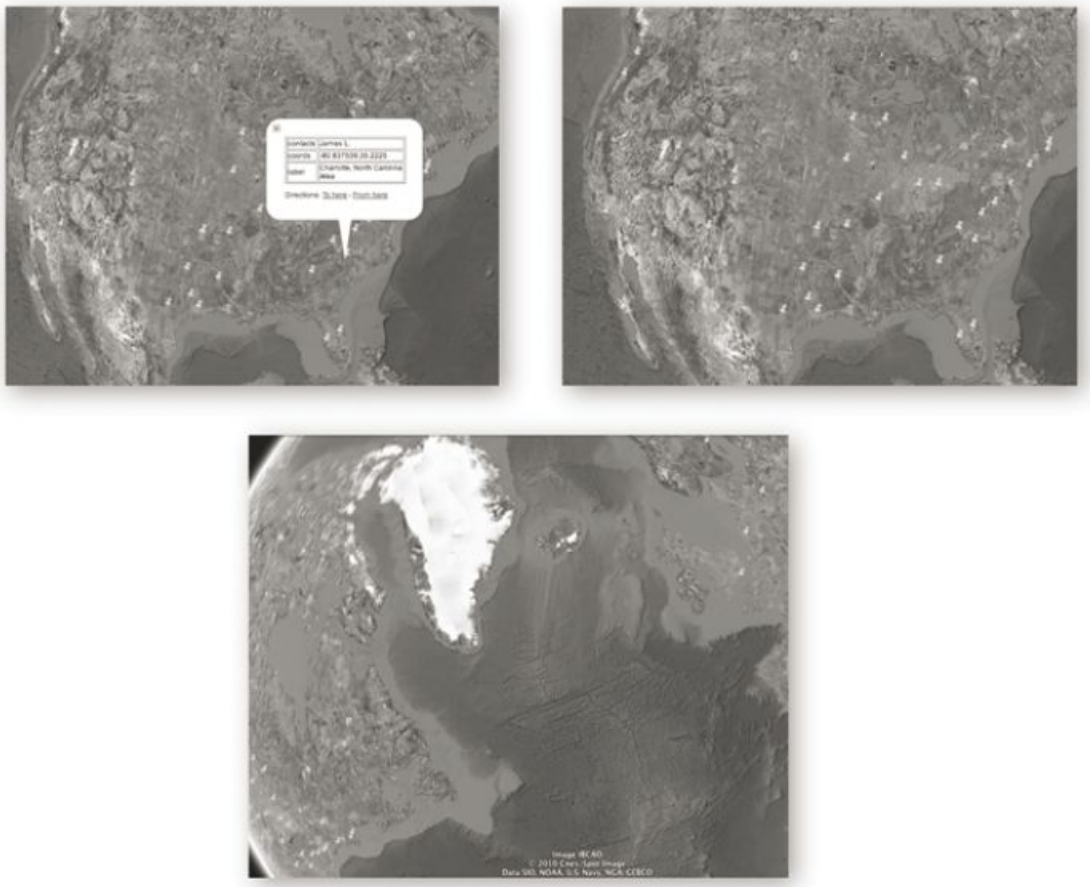


Figura 6.6 – Siga os passos partindo do canto superior esquerdo até a base: 1) agrupe seus contatos por localização, para analisar facilmente quem vive/trabalha em cada cidade; 2) localize os centroides de três agrupamentos computados por k-médias; 3) não se esqueça de que agrupamentos podem se estender por países, ou até continentes, quando estiver tentando encontrar uma localização ideal para um encontro.

A simples visualização de sua rede pode ser muito interessante, mas a computação dos centroides geográficos de sua rede profissional também é capaz de abrir possibilidades realmente intrigantes. Por exemplo, talvez você esteja interessado em computar localizações adequadas para uma série de workshops ou conferências regionais. Da mesma forma, caso você trabalhe com consultoria e conviva com uma agenda frenética de viagens, pode tentar encontrar algumas localizações ideais para o aluguel de uma residência temporária. Ou talvez você queira mapear profissionais de sua rede de acordo com as responsabilidades de seus cargos, ou com a classificação socioeconômica à qual pertencem, a partir de seus cargos e experiência. Além das inúmeras opções apresentadas pela visualização dos dados de localização de sua rede profissional, agrupamentos

geográficos também são perfeitos para muitas outras possibilidades, como para lidar com problemas de gerenciamento de uma rede de fornecimento, ou com o problema do caixeiro viajante (*travelling salesman problem*, TSP, *http://en.wikipedia.org/wiki/Travelling_salesman_problem*).

Mapeamento de sua rede profissional com Cartogramas Dorling

O Protovis (*http://vis.stanford.edu/protovis/*), kit de ferramentas avançadas de visualização em HTML5, que será apresentado no capítulo 7, inclui uma visualização chamada Dorling Cartogram, essencialmente um gráfico de bolha geograficamente agrupado. Enquanto um cartograma tradicional pode transmitir informações distorcendo as fronteiras geográficas de um estado em um mapa, um Dorling Cartogram posiciona no mapa uma figura uniforme, como um círculo, aproximadamente onde o estado em si deveria estar localizado, e codifica informações utilizando a circunferência (e muitas vezes a cor) desse círculo (Figura 6.7). Trata-se de uma ótima ferramenta de visualização uma vez que lhe permite utilizar seus instintos para determinar onde devem aparecer as informações em uma superfície de mapeamento 2D, além de ser capaz de codificar parâmetros utilizando propriedades muito intuitivas, como as áreas e as cores das figuras.

O Protovis também inclui diversas outras visualizações que transmitem informações geográficas, como heatmaps (*http://vis.stanford.edu/protovis/ex/heatmap.html*), mapas de símbolos (*http://vis.stanford.edu/protovis/ex/symbol.html*), e mapas de chloropleth (*http://vis.stanford.edu/protovis/ex/choropleth.html*). Consulte o artigo "A Tour Through the Visualization Zoo" (*http://queue.acm.org/detail.cfm?id=1805128*), para um estudo geral destas e de muitas outras visualizações que podem lhe ser úteis.

Depois disso tudo, supondo que você tenha acompanhado o exemplo 6.14 e geocodificado com êxito seus contatos a partir dos dados de localização fornecidos pelo LinkedIn, basta um ajuste mínimo no script para produzir uma saída levemente diferente que possibilite uma visualização na forma de um Dorling Cartogram do Protovis[14]. Uma versão modificada do exemplo canônico do Dorling Cartogram está disponível no GitHub (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/linkedin__create_dorling_cartogram_output.py*) e deve produzir uma visualização semelhante à da figura 6.7. O código de exemplo conecta os pontos para produzir uma visualização útil, mas há muito mais que você poderia fazer para aprimorar o resultado, como adicionar manipuladores de eventos para exibir informações de conexão

quando um estado específico é clicado. Como apontamos no alerta logo após o exemplo 6.14, a abordagem de geocodificação implementada aqui é necessariamente imperfeita e pode demandar alguns ajustes.

Figura 6.7 – Cartograma Dorling que utiliza a área do círculo para transmitir o número de colegas de trabalho que reside em cada estado; cores poderiam ter sido utilizadas para denotar outras variáveis, como a taxa de desemprego associada ao estado ou a rotatividade de emprego de seus colegas, como determinadas pela mineração de informações estendidas de perfil.

Comentários finais

Este capítulo abordou muitos tópicos, apresentando e aplicando técnicas fundamentais de agrupamento aos dados de sua rede profissional do LinkedIn de várias formas diferentes. Assim como nos outros capítulos, vimos apenas a ponta do iceberg; há muitas outras práticas interessantes que não foram aqui apresentadas e que podem ser utilizadas sobre seus dados do LinkedIn. Não tenha pressa e explore as informações estendidas de perfil que você tem à disposição. Pode ser interessante tentar correlacionar o local de trabalho das pessoas com informações sobre onde elas estudaram e/ou analisar se as pessoas costumam se mudar para outras áreas. Porém, lembre-se de que seu cinto de ferramentas contém muitas outras ferramentas que não necessariamente têm de incluir dados geográficos ou agrupamentos. Caso você continue analisando dados geográficos, um dos muitos projetos que podem ser verificados é um emergente framework de código aberto, o geodict (*http://petewarden.typepad.com/searchbrowser/2010/10/geodict-an-open-source-tool-for-extracting-locations-from-text.html*).

1 A posição do LinkedIn foi transmitida em uma discussão entre o autor e D.J. Patil, Cientista-Chefe, Diretor Executivo de Segurança, e Diretor Sênior de Análise de Projetos no LinkedIn.

2 N.T.: clustering é uma técnica de mineração de dados que realiza agrupamentos automáticos de dados de acordo com o grau de semelhança. O critério de semelhança faz parte da definição do problema e, dependendo, do algoritmo (fonte: Wikipédia).

3 Em termos técnicos, às vezes é mais preciso descrever a complexidade de tempo de um problema como $\Theta(n^2)$, pois $O(n^2)$ representa um pior caso, enquanto $\Theta(n^2)$ representa uma associação mais indicativa do desempenho esperado. Em outras palavras, $\Theta(n^2)$ diz que não se trata apenas de um problema n^2 no pior caso, mas também de um problema n^2 no melhor caso. Entretanto, nosso objetivo neste capítulo não é analisar a complexidade de tempo de execução dos algoritmos (tópico difícil por si só), por isso utilizaremos a notação $O(n^2)$, mais comum, descrevendo o pior caso.

4 N.T.: a notação do Grande-O ou notação assintótica (Big-O notation, ou Big Omicron Notation, em inglês) é uma notação matemática utilizada para analisar o comportamento assintótico de funções, muito empregada na análise de algoritmos em ciência da computação (fonte: Wikipédia).

5 Também chamada de "correspondência difusa", de clustering, ou ainda de duplicação, dentre muitos outros nomes. A tentativa de realizar uma correspondência aproximada de campos múltiplos (registros) é comumente referida como o problema do "relacionamento de registros".

6 Se você acredita que as coisas estão começando a complicar, imagine o trabalho da Dun & Bradstreet (*http://www.dnb.com/us/*, o "Quem é quem" das informações empresariais), empresa agraciada com o desafio de manter um diretório global, identificando empresas representadas em idiomas espalhados por todo o planeta.

7 MASI significa "Measuring Agreement on Set-Valued Items". Consulte *http://www.cs.columbia.edu/~becky/pubs/lrec06masi.pdf*.

8 `def subsumed(X,Y): return X.union_update(Y) == X or Y.union_update(X) == Y`

9 N.T.: em telecomunicações e em engenharia de software, escalabilidade é uma característica desejável em todo sistema, rede ou processo que indica habilidade para manipular uma porção crescente de trabalho de forma uniforme, ou capacidade de crescimento (fonte: Wikipédia).

10 O linkedin__cluster_contacts_by_title_hac.py (*https://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/linkedin__cluster_contacts_by_title_hac.py*) fornece uma adaptação mínima do exemplo Exemplo 6.6 que demonstra como produzir saída que poderia ser consumida pelo Protovis.

11 N.T.: um applet é um software aplicativo executado no contexto de outro programa (como um navegador web), e que geralmente executa funções bem específicas (fonte: Wikipédia).

12 Em outubro 2010, o LinkedIn estava pondo em prática uma implementação do OAuth 2 (*http://blog.linkedin.com/2010/10/06/linkedin-oauth2/*). O módulo linkedin da versão 1.6 implementa o OAuth 1.0a, como descrito em *http://developer.linkedin.com/docs/*.

13 N.T.: geocodificação é o processo de localização de coordenadas geográficas a partir

de outros dados geográficos, como endereços de rua, ou códigos postais (fonte: Wikipédia).

14 A visualização em Dorling Cartogram do Protovis é atualmente implementada para manipular somente localizações nos Estados Unidos.

CAPÍTULO 7

Google Buzz: TF-IDF, similaridade de cosseno e colocações

Este breve capítulo inicia nossa jornada pela mineração de textos[1], e representa uma espécie de ponto de inflexão neste livro. Capítulos anteriores trataram principalmente da análise de dados estruturados ou semiestruturados, como registros codificados em microformatos, relacionamentos entre pessoas, ou hashtags especialmente marcadas de tweets. Este capítulo, por outro lado, inicia a transformação e a compreensão das informações textuais dos documentos por meio da introdução de fundamentos da teoria da Recuperação de Informação (ou RI), como TF-IDF, similaridade de cosseno, e detecção de colocação. Como você pode ter percebido pelo título do capítulo, o Google Buzz servirá inicialmente como nossa principal fonte de dados, pois é inerentemente social, fácil de ser minerado[2], e tem muito potencial para a web social (*http://radar.oreilly.com/2010/02/google-buzz-re-invents-gmail.html*). Perto do final deste capítulo, também veremos o que é necessário para explorar seus dados do Gmail. Nos capítulos futuros, investigaremos a mineração de dados de blogs e de outras fontes de texto livre, além de apresentar formas adicionais de analíticas de texto, como a extração de entidades e a geração automática de abstracts. Não há motivo real para introduzir primeiro o Buzz e depois os blogs (tópico do capítulo 8), a não ser o fato de que o Buzz preenche um nicho muito interessante entre a funcionalidade do Twitter e a dos blogs, de modo que a ordem que aplicamos facilita o acompanhamento de capa a capa dos tópicos deste livro. Em última análise, as técnicas de mineração que você aprenderá, em qualquer capítulo específico, poderão ser aplicadas com a mesma facilidade aos outros capítulos.

Sempre que possível, não reinventaremos algo que já existe, nem implementaremos ferramentas de análise criadas do zero, mas faremos algumas análises mais profundas quando surgirem tópicos especialmente

importantes, essenciais para a compreensão da mineração de texto. O Natural Language Toolkit (NLTK), poderosa tecnologia que vimos em alguns exemplos no capítulo 1, fornece muitas das ferramentas que veremos. A riqueza de suas APIs pode assustá-lo de início, mas não se preocupe: a análise de textos é um campo de estudo incrivelmente diverso e complexo, mas há muitos fundamentos poderosos capazes de realmente facilitar seu trabalho, sem exigir um investimento significativo. Este capítulo e os seguintes pretendem analisar precisamente esses fundamentos.

Uma introdução completa ao NLTK está além do escopo deste livro, mas você pode revisar o texto completo de Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit (O'Reilly) on-line (*http://www.nltk.org/book*). Cópias em papel também estão disponíveis em *http://oreilly.com/catalog/9780596516499*.

Buzz = Twitter + Blogs (???)

Qualquer pessoa com uma conta do Gmail também tem uma conta do Buzz. Do ponto de vista do produto, o Google Buzz parece se encaixar em algum ponto entre a funcionalidade do Twitter e a dos blogs (*http://radar.oreilly.com/2010/03/google-buzz-and-the-conversati.html*). Uma *atividade* no Buzz pode ser tão longa quanto um post de um blog, mas parece mais informal, como um tweet. Da mesma forma que o Twitter, o Buzz é tipicamente utilizado para transmissão rápida de ideias com um mínimo de formatação, e fluxos de atividade podem ser estabelecidos entre amigos e seguidores. De fato, muitos usuários do Buzz personalizam suas configurações para que sua atividade no Twitter seja representada em seus feeds do Buzz (Figura 7.1). Quando da redação deste livro, o Buzz apresentava uma API mais compacta que a do Twitter, e rate limits pareciam ainda não estar em vigor. Enquanto forem válidas, estas duas características podem simplificar o trabalho envolvido na mineração de dados do Buzz. Será interessante acompanhar sua evolução e descobrir se ele se tornará mais semelhante ao Twitter e/ou aos blogs. Por ora, entretanto, vamos iniciar a coleta e a mineração de alguns de seus dados. Vale a pena fazer uma pausa para se familiarizar com a forma de funcionamento do Buzz, seja por meio de um feed público ou utilizando o link de navegação "Buzz" em sua conta do Gmail. O seu feed público do Buzz (como o de todos os usuários) deve estar disponível em um URL

com a seguinte forma: *http://www.googleapis.com/buzz/v1/activities/your_gmail_account_id/@public*. Por exemplo, o feed público do Buzz de Tim O'Reilly está disponível em: *http://www.googleapis.com/buzz/v1/activities/timoreilly/@public*.

O Buzz expõe uma API RESTful (*http://code.google.com/apis/buzz/v1/using_rest.html*), envolvendo três recursos principais: pessoas, atividades e comentários. "Atividades" do Buzz são mais ou menos sinônimos de "posts", uma vez que nós as consideraremos como recursos em texto. Entretanto, lembre-se de que uma das características mais interessantes do Buzz é a forma como ele pretende integrar perfeitamente em seu fluxo de atividade outros tipos de mídia, como fotos e vídeos, de outros produtos e serviços do Google.

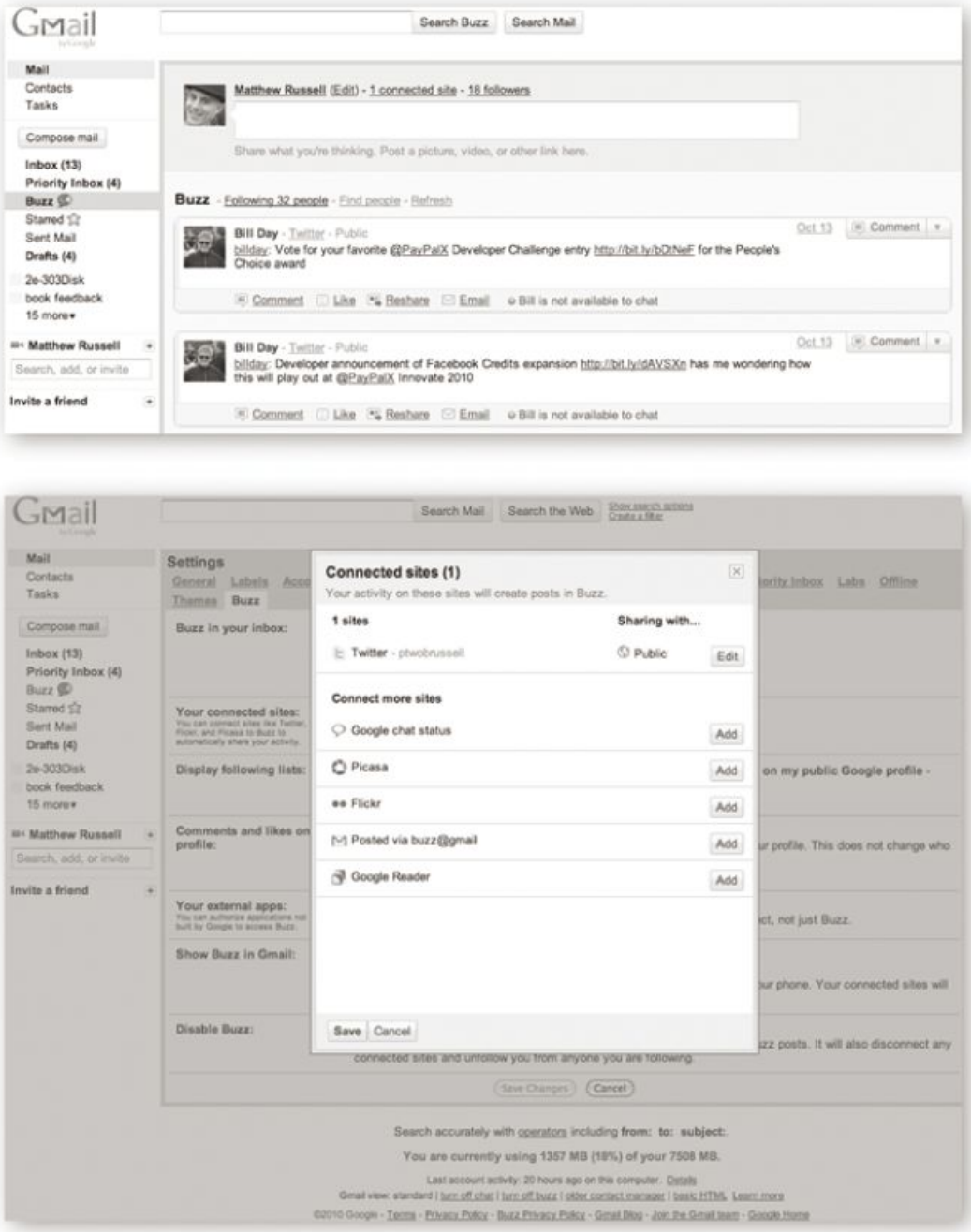


Figura 7.1 – Configurações de conta do Gmail permitem que você integre facilmente atividades do Twitter e de outros serviços ao seu feed do Buzz.

A documentação da API do Google Buzz (*http://code.google.com/apis/buzz/docs/*) descreve adequadamente as formas RESTful pelas quais você pode interagir com o Buzz, caso seu interesse seja mais casual. Todavia, o livro RESTful Web Services (*http://oreilly.com/catalog/9780596529260*), da O'Reilly, é um ótimo

recurso caso você esteja interessado em uma análise mais detalhada do REST e queira ver exemplos práticos que manipulem situações não tão óbvias.

Há um ótimo cliente Buzz feito em Python, o `buzz`, disponível em *http://code.google.com/p/buzz-python-client/*. Você pode instalá-lo com `easy_install buzz-python-client`.

Agora, vamos buscar alguns dados do feed público do Buzz de Tim O'Reilly. Ainda que o módulo `buzz` exponha suporte OAuth, utilizado caso você necessite de recursos que requeiram autenticação, neste capítulo utilizaremos apenas informações que podem ser obtidas a partir de perfis públicos, por isso esse recurso não será necessário. Há muitas possibilidades interessantes para mineração de dados de relacionamento social disponíveis no Google Buzz, mas nos concentraremos nos dados de texto livre disponíveis nos posts, em vez de reinventar ou continuar desenvolvendo as abordagens de amigos/seguidores apresentadas nos capítulos que trataram do Twitter. Consulte esses capítulos caso esteja interessado em interagir com o Buzz dessa forma.

Dados do Buzz são expostos como um feed, e você deve supor que contêm marcação, entidades HTML escapadas etc. O código seguinte apresenta uma sessão de exemplo que demonstra como utilizar o módulo `buzz` para buscar e analisar textos simples, adequada para mineração de até 100 posts públicos. Ainda que este primeiro exemplo ilustre a utilização do CouchDB como mídia de armazenamento, faremos uma transição para além dele, em favor do armazenamento dos dados utilizando *pickling*, para que possamos aperfeiçoar nossas técnicas de mineração de dados, em vez de discutir considerações de armazenamento. Caso você esteja interessado em uma utilização mais agressiva do CouchDB, assim como em add-ons úteis como o `couchdb-lucene`, consulte os capítulos anteriores deste livro.

O exemplo 7.1 cria uma estrutura de dados conveniente que consiste no título, no conteúdo, nos comentários e no link para cada post do Buzz, armazenando-a em um banco de dados do CouchDB. É fácil não dar a devida atenção a esse elemento, mas note a importância de `cleanHtml`. Ele cuida de remover a marcação e converter as entidades HTML para que o conteúdo de cada post fique o mais limpo e legível possível – consideração muito importante quando suas rotinas de análise não

esperam encontrar entidades HTML e elementos do tipo.

Exemplo 7.1 – Coleta de dados do Google Buzz (buzz___get_buzz_posts_and_comments.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import buzz
from BeautifulSoup import BeautifulStoneSoup
from nltk import clean_html
import couchdb
import json
USER = sys.argv[1]
MAX_RESULTS = 100
# Funções helper para remoção do html e conversão de entidades
escapadas.
# Retorna UTF-8
def cleanHtml(html):
    return BeautifulStoneSoup(clean_html(html),
            convertEntities=BeautifulStoneSoup.HTML_ENTITIES).con
tents[0]
client = buzz.Client()
posts_data = client.posts(type_id='@public',
                          user_id=USER,
                          max_results=MAX_RESULTS).data
posts = []
for p in posts_data:
    # Buscar um grande número de comentários para muitos posts
pode demorar um pouco.
    # O código de thread pool de mailboxes__CouchDBBulkReader.py
poderia
    # ser adaptado e utilizado aqui.
    comments = [{'name': c.actor.name, 'content':
cleanHtml(c.content)} for c in
                p.comments().data]
    link = p.uri
    post = {
        'title': cleanHtml(p.title),
        'content': cleanHtml(p.content),
        'comments': comments,
        'link': link,
        }
```

```
    posts.append(post)
# Caso você prefira, armazene em um arquivo local como dados JSON
if not os.path.isdir('out'):
    os.mkdir('out')
filename = os.path.join('out', USER + '.buzz')
f = open(filename, 'w')
f.write(json.dumps(posts))
f.close()
print >> sys.stderr, "Data written to", f.name
# Ou armazene em algum lugar como o CouchDB da seguinte
maneira...
# server = couchdb.Server('http://localhost:5984')
# DB = 'buzz-' + USER
# db = server.create(DB)
# db.update(posts, all_or_nothing=True)
```

Agora que você tem as ferramentas para buscar e armazenar dados do Buzz, vamos iniciar nossa análise.

Exploração de dados com o NLTK

O NLTK foi escrito de modo a permitir que você explore muito facilmente os dados e formule algumas impressões iniciais, sem que para isso necessite de um grande investimento. Entretanto, antes de avançar, acompanhe primeiro a sessão do interpretador, no exemplo 7.2, para entender melhor algumas das poderosas funcionalidades prontas para uso que o NLTK oferece. Não se esqueça de que você pode utilizar a função `help` para obter mais informações sempre que necessário. Por exemplo, `help(nltk)` fornece a documentação para o pacote NLTK. Lembre-se também de que nem todas as funcionalidades do interpretador estão destinadas à incorporação em software de produção, uma vez que a saída é escrita por meio da saída-padrão e não pode ser capturada em uma estrutura de dados como uma lista. Nesse sentido, métodos como `nltk.text.concordance` são considerados "funcionalidade de demonstração". Falando nisso, muitos dos módulos do NTLK têm uma função de demonstração que você pode chamar para entender como utilizar a funcionalidade oferecida. O código-fonte para essas demonstrações representa um ótimo ponto de partida para que você aprenda como utilizar a nova API. Por exemplo, você poderia executar

`nltk.text.demo()` no interpretador para obter insights adicionais quanto às capacidades oferecidas pelo módulo `nltk.text`. Veremos como funciona esse recurso de demonstração nas páginas futuras.

Os exemplos deste capítulo, incluindo a sessão do interpretador a seguir, utilizam o método split para tokenizar o texto. O capítulo 8 apresenta abordagens mais sofisticadas para tokenização que podem produzir um melhor resultado.

Exemplo 7.2 – Exploração de dados do Google Buzz no interpretador com o NLTK. O exemplo 7.1 produz um arquivo USER.buzz no diretório de saída a partir do qual você executou o script

```
>>> import nltk
>>> import json
>>> buzz_data = json.loads(open("out/timoreilly.buzz").read())
>>> all_content = " ".join([ p['content'] for p in buzz_data ])
>>> len(all_content)
42930
>>> tokens = all_content.split()
>>> text = nltk.Text(tokens)
>>> text.concordance("open")
Building index...
Displaying 22 of 22 matches:
 operating system. Andy told him why open systems matter. "If
google did not a
mulus payments for meaningful use of open healthcare records. It
could also be
 planning meeting at the Times (it's open to the public). It's
great to see ho
 me as the most influential voice in open source.
http://www.mindtouch.com/blo
ff, I don't tweet or blog much about open source any more -
though the subject
eply influenced by my thinking about open source. I came to
articulate my idea
 about the long term implications of open source and how it would
reshape the
ons have also taken me away from the open source mainstream. It's
only recentl
ecently that members of the hardcore open source and free
software advocacy co
ntally changing the context in which open source exists, and that
it hasn't do
er in favor of understanding how the open source community
actually communicat
y remind people to attend OSCON, the open Source Convention,
```

```
where the long te
he long term thinking that drives my open source position is
leading us to cov
n is leading us to cover topics like open source and cloud
computing, open sou
ike open source and cloud computing, open source hardware, and
many other topi
ks than you can count), I do have an open source bully pulpit. It
just isn't t
Reilly to Keynote at Computerworld's open Source Business
Conference http://bi
hasn't built a personal brand in the open source space such that
his company i
trum auction and their launch of the open Handset Alliance are
among the compa
ng their phone even further from the open web. In the end, if the
iPhone fails
s have weaker adoption dynamics than open systems, not because
Eric Schmidt be
he Gov't has put in place around the open government directive.
Any pushing yo
>>> text.collocations()
Building collocations list
open source; Economist Innovation; 1/4 cup; long term; Innovation
event.; Web 2.0; I'd love; free software; Annalee Saxenian; Nexus
One.; came back; Cauliflower Pancakes; Eric Schmidt; Open Source;
certainly true; olive oil; Andy Rubin; Mr. O'Reilly; O'Reilly
said.;
tech support
>>> fdist = text.vocab()
Building vocabulary index...
>>> fdist["open"]
19
>>> fdist["source"]
14
>>> fdist["web"]
3
>>> fdist["2.0"]
3
>>> len(tokens)
6942
>>> len(fdist.keys()) # tokens únicos
2750
```

```
>>> [w for w in fdist.keys()[:100] \
... if w.lower() not in nltk.corpus.stopwords.words('english')]
[u'-', u"it's", u'open', u'like', u'people', u'also', u'would',
u"It's", u"don't",
 u'source', u'Apple', u'one', u'Google', u'government', u"I'd",
u'Microsoft',
 u'many', u'great', u'innovation', u'make', u'think',
u'companies', u'even', u'get',
 u'good', u'lot', u'made', u'new', u'story', u'technology', u')',
u"I'm", u'big',
 u'going', u'long', u'love']
>>> [w for w in fdist.keys() if len(w) > 15 and not
w.startswith("http")]
[u'\u201c#airportsecurity', u'Caravaggiomania,',
u'administrations,',
 u'anti-competitive.', u'apparentlycompared', u'entrepreneur-
friendly.',
 u'entrepreneur-unfriendly.', u'flyontime.us/m/lines/security',
u'government/small',
 u'interoperability', u'over-psychologized', u'super-
impressive.',
 u'technology-minded']
>>> len([w for w in fdist.keys() if w.startswith("http")])
35
>>> for rank, word in enumerate(fdist): print rank, word,
fdist[word]
0 the 317
1 to 199
2 of 180
3 a 150
4 that 129
5 and 122
6 in 110
7 is 79
... output truncated ...
```

Talvez você tenha de executar `nltk.download('stopwords')` para fazer o download dos dados de stopwords do NTLK, caso ainda não os tenha instalado. Se possível, recomendamos que você execute apenas `nltk.download()` para instalar todos os dados do NLTK.

O último comando na sessão do interpretador lista as palavras da distribuição de frequência, ordenadas por frequência. Não é nenhuma surpresa que palavras usuais como "the", "to" e "of" – todas *stopwords* (também conhecidas como palavras de parada, ou de ligação) – sejam as

mais frequentes, mas, depois delas, há uma grande queda na frequência dos termos e a distribuição apresenta uma cauda muito longa. Estamos trabalhando com um pequeno exemplo de dados do Buzz, mas esta mesma propriedade segue válida para qualquer análise de frequência de linguagem natural. A lei de Zipf (*http://en.wikipedia.org/wiki/Zipf's_law*), famosa lei empírica de linguagem natural, afirma que a frequência de uma palavra em um *corpus* (conjunto de documentos) é inversamente proporcional à sua classificação na tabela de frequência. Isso significa que se o termo de maior frequência em um corpus corresponde a N% das palavras totais, o segundo termo mais frequente deve corresponder a (N/2)% das palavras, o terceiro a (N/3)% etc. Quando representada em um gráfico, tal distribuição (mesmo para um pequeno exemplo de dados) mostra uma curva que acompanha cada eixo (Figura 7.2). Uma observação importante é a de que a maioria da área em uma distribuição desse tipo encontra-se sob sua cauda, o que para um corpus de tamanho suficiente, que corresponde a um exemplo razoável de uma linguagem, é um dado considerável. Se você representasse esse tipo de distribuição em um gráfico em que cada eixo estivesse escalonado por um algoritmo, a curva se aproximaria de uma linha reta para um tamanho representativo de exemplo.

A lei de Zipf fornece muitos insights referentes à apresentação da frequência de distribuição das palavras que surgem em um corpus, além de algumas regras práticas que podem ser úteis quando queremos estimar frequências. Por exemplo, se você sabe que há um milhão de palavras (não-únicas) em um corpus, e supõe que a palavra mais frequente (geralmente "the", em um texto em inglês) corresponde a 7% desse número[3] pode derivar o número total de cálculos lógicos que um algoritmo realizaria se você considerasse uma amostra específica dos termos da distribuição de frequência. Às vezes, esse tipo de aritmética simples é suficiente para verificar a possibilidade de suposições referentes a um longo tempo de execução, ou confirmar se certas computações em um conjunto de dados de tamanho considerável chegam a ser viáveis.

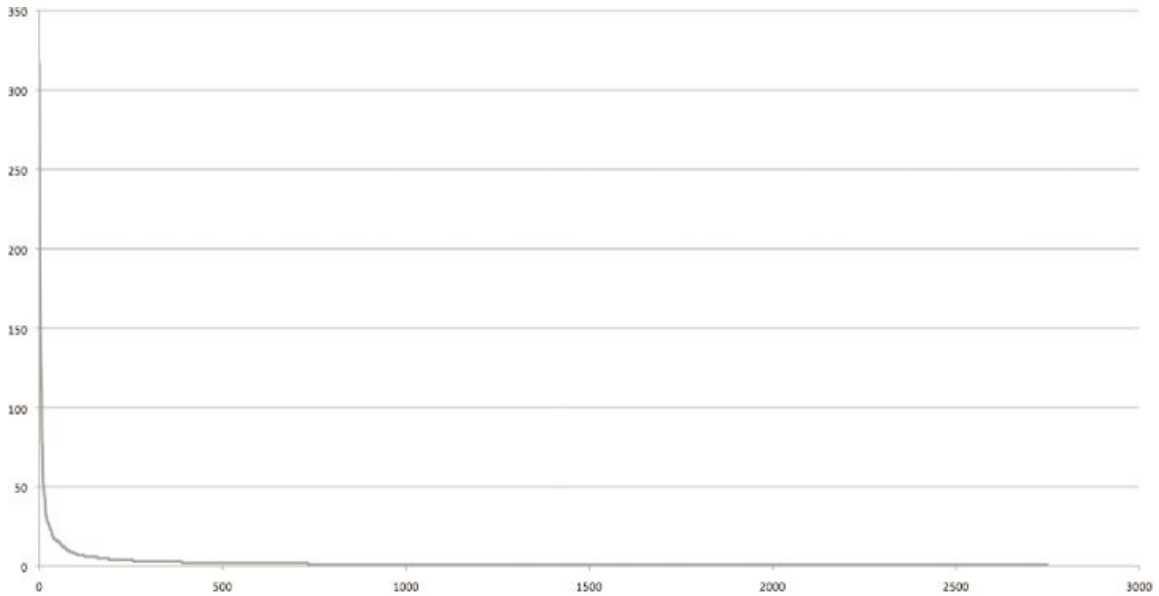


Figura 7.2 – Distribuição de frequência para termos que surgem em um pequeno exemplo de dados do Buzz. Note que ela acompanha cada eixo.

Fundamentos da mineração de texto

Ainda que abordagens rigorosas ao processamento de linguagem natural (*Natural Language Processing*, ou NLP), incluindo elementos como segmentação de sentenças, tokenização, agrupamento de palavras e detecção de entidades, sejam necessárias para que possamos entender completamente os dados textuais, é interessante apresentar primeiramente alguns fundamentos da teoria da Recuperação de Informação. O restante deste capítulo apresenta alguns aspectos fundamentais dessa teoria, incluindo a TF-IDF, a métrica da similaridade do cosseno, e parte da teoria que explica a detecção de colocação. O capítulo 8 oferece uma discussão mais detalhada sobre NLP.

Se você quiser se aprofundar na teoria da RI, o texto completo de Introduction to Information Retrieval está disponível on-line (*http://nlp.stanford.edu/IR-book/information-retrieval-book.html*), fornecendo todas as informações necessárias sobre esse tópico.

Uma introdução-relâmpago à TF-IDF

Recuperação de informação é um campo extenso, com muitas especialidades. Esta discussão concentra-se na TF-IDF, uma das técnicas mais importantes para recuperação de documentos relevantes a partir de um corpus. TF-IDF significa *term frequency-inverse document frequency* (frequência do termo-frequência inversa do documento) e pode ser utilizada para consultar um corpus calculando pontuações normalizadas

que expressem a importância relativa dos termos nos documentos. Matematicamente, a TF-IDF pode ser expressa como o produto da frequência do termo e da frequência inversa do documento, *tf_idf = tf*idf*, em que o termo `tf` representa a importância de um termo em um documento específico e `idf` representa a importância de um termo relativa ao corpus como um todo. A multiplicação desses termos produz uma pontuação que contabiliza ambos os fatores e que tem sido parte integral de todos os principais mecanismos de busca em algum momento de sua existência. Para um entendimento mais intuitivo sobre como funciona a TF-IDF, vamos analisar cada um dos cálculos envolvidos na computação da pontuação geral.

Por simplicidade, suponha que temos um corpus com três documentos de exemplo e que os termos são calculados verificando quebras quando ocorrem espaços em branco (Exemplo 7.3).

Exemplo 7.3 – Estruturas de dados de exemplo utilizadas por todo este capítulo

```
corpus = {
      'a' : "Mr. Green killed Colonel Mustard in the study with
the candlestick. \
             Mr. Green is not a very nice fellow.",
      'b' : "Professor Plumb has a green plant in his study.",
      'c' : "Miss Scarlett watered Professor Plumb's green plant
while he was away \
             from his office last week."
}
terms = {
        'a' : [ i.lower() for i in corpus['a'].split() ],
        'b' : [ i.lower() for i in corpus['b'].split() ],
        'c' : [ i.lower() for i in corpus['c'].split() ]
        }
```

A frequência de um termo poderia ser simplesmente representada como o número de vezes em que ele ocorre no texto, mas é mais comum que seja normalizada considerando o número total de termos no texto, para que a pontuação geral considere também a extensão do documento relativa à frequência de um termo. Por exemplo, o termo "green" (depois de normalizado para letras minúsculas) ocorre duas vezes no `corpus['a']` e apenas uma vez no `corpus['b']`. Assim, `corpus['a']` produziria uma pontuação mais alta se a frequência fosse o único critério de pontuação. Entretanto, se você normalizar o comprimento do documento,

`corpus['b']` terá uma frequência levemente maior para "green" (1/9) do que `corpus['a']` (2/19), uma vez que `corpus['b']` é mais curto – ainda que "green" ocorra mais frequentemente no `corpus['a']`. Dessa forma, uma técnica comum para a pontuação de uma consulta composta como "Mr. Green" é somar a pontuação da frequência dos termos para cada um dos termos de consulta em cada documento, e retornar os documentos classificados pela pontuação somada da frequência dos termos.

O parágrafo anterior não é tão confuso quanto parece; por exemplo, uma consulta em nosso corpus de exemplo por "Mr. Green" retornaria as pontuações normalizadas que podem ser vistas na tabela 7.1 para cada documento.

Tabela 7.1 – Exemplo de pontuações da frequência de termos para "Mr. Green"

Documento	tf(Mr.)	tf(Green)	Soma
corpus['a']	2/19	2/19	4/19 (0,2105)
corpus['b']	0	1/9	1/9 (0,1111)
corpus['c']	0	1/16	1/16 (0,0625)

Para este exemplo artificial, um esquema cumulativo de pontuação para frequência de termos analisa e retorna `corpus['a']` (documento que esperávamos que fosse retornado), uma vez que `corpus['a']` é o único que contém o token composto "Mr. Green". Entretanto, muitos problemas poderiam surgir, uma vez que o modelo de pontuação da frequência dos termos analisa cada documento como uma coleção não-ordenada de palavras. Por exemplo, consultas por "Green Mr.", ou "Green Mr. Foo", retornariam exatamente a mesma pontuação verificada para a consulta por "Mr. Green", ainda que nenhum desses tokens compostos estivesse presente nas frases de exemplo. Além disso, há muitos cenários que poderíamos facilmente formular para ilustrar os resultados consideravelmente pobres apresentados pela técnica de classificação de frequência dos termos, explorando o fato de que a pontuação gramatical à direita dos termos não é manipulada da forma correta, e de que o contexto ao redor dos tokens de interesse não é considerado nos cálculos.

Considerar apenas a frequência dos termos parece ser um problema usual quando pontuamos um documento de cada vez, pois tal procedimento não leva em conta palavras muito frequentes, comuns em muitos documentos. Ou seja, todos os termos têm o mesmo peso, independentemente de sua importância de fato. Por exemplo, "the green

plant" contém a stopword "the", que distorce a pontuação geral da frequência dos termos em favor do `corpus['a']` uma vez que "the" surge duas vezes nesse documento, assim como "green". Em contraste, no `corpus['c']` "green" e "plant" surgem apenas uma vez. Como consequência, as pontuações podem ser representadas como mostra a tabela 7.2, com `corpus['a']` classificado como mais relevante do que `corpus['c']`, ainda que sua intuição possa indicar resultados diferentes. Felizmente, todavia, `corpus['b']` ainda tem a melhor classificação.

Tabela 7.2 – Pontuações de exemplo da frequência dos termos para "the green plant"

Documento	tf(the)	tf(green)	tf(plant)	Soma
corpus['a']	2/19	2/19	0	4/19 (0,2105)
corpus['b']	0	1/9	1/9	2/9 (0,2222)
corpus['c']	0	1/16	1/16	1/8 (0,125)

Kits de ferramentas como o NLTK fornecem listas de stopwords que podem ser utilizadas para filtrar termos como "the", "a", "and" etc., mas lembre-se de que pode haver termos que escapam até mesmo das melhores listas de stopwords, e que são relativamente usuais em domínios especializados. A métrica de frequência inversa do documento é um cálculo que fornece uma métrica genérica de normalização para um corpus, e que considera a presença de termos usuais em um conjunto de documentos verificando o número total de documentos em que um termo de consulta se faz presente. A intuição por trás dessa métrica é a de que ela produzirá um valor mais alto caso um termo seja relativamente incomum, o que ajuda a lidar com o problema das stopwords que acabamos de investigar. Por exemplo, uma consulta por "green" no corpus dos documentos de exemplo deveria retornar uma pontuação de frequência inversa do documento mais baixa, em comparação com uma consulta por "candlestick", pois "green" surge em todos os documentos, enquanto "candlestick" em apenas um. Matematicamente, a única nuance de interesse no cálculo da frequência inversa do documento é a de que um logaritmo é utilizado para "achatar" o resultado em um intervalo comprimido (Figura 7.3), uma vez que sua aplicação usual seria na multiplicação desse valor pela frequência de um termo como fator de escalonamento.

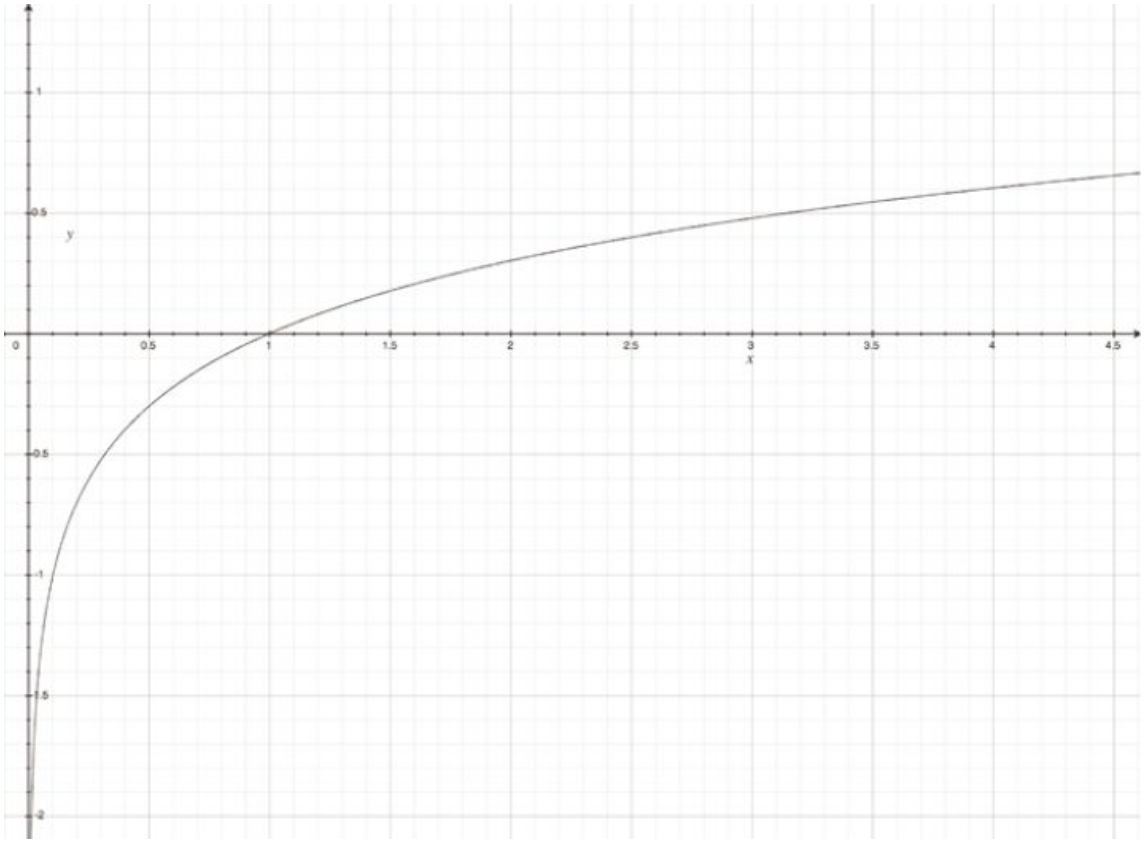


Figura 7.3 – O logaritmo "achata" o intervalo de valores transformando-o em um espaço mais comprimido.

Com isso, fechamos o ciclo e projetamos uma forma de computar uma pontuação para uma consulta de termos múltiplos que considere a frequência dos termos presentes em um documento, o comprimento do documento em que qualquer termo específico se faz presente, e a unicidade geral dos termos nos documentos de todo o corpus. Em outras palavras, `tf-idf = tf*idf`. O Exemplo 7.4 é uma implementação simplista dessa discussão que deve ajudar a solidificar os conceitos descritos. Não tenha pressa e analise-o com calma; depois, veremos algumas consultas de exemplo.

Exemplo 7.4 – Execução de TF-IDF em dados de exemplo (buzz__tf_idf.py)

```
# -*- coding: utf-8 -*-
import sys
from math import log
QUERY_TERMS = sys.argv[1:]
def tf(term, doc, normalize=True):
    doc = doc.lower().split()
    if normalize:
```

```
        return doc.count(term.lower()) / float(len(doc))
    else:
        return doc.count(term.lower()) / 1.0

def idf(term, corpus):
    num_texts_with_term = len([True for text in corpus if
term.lower()
                               in text.lower().split()])
    # O cálculo da tf-idf envolve a multiplicação utilizando um
valor de tf menor que 0, por isso
    # é importante retornar um valor maior que 1 para pontuação
consistente. A multiplicação de
    # dois valores menores que 1 retorna um valor menor que cada
um deles
    try:
        return 1.0 + log(float(len(corpus)) /
num_texts_with_term)
    except ZeroDivisionError:
        return 1.0

def tf_idf(term, doc, corpus):
    return tf(term, doc) * idf(term, corpus)

corpus = \
    {'a': 'Mr. Green killed Colonel Mustard in the study with the
candlestick. \
Mr. Green is not a very nice fellow.',
     'b': 'Professor Plumb has a green plant in his study.',
     'c': "Miss Scarlett watered Professor Plumb's green plant
while he was away \
from his office last week."}

# Pontua consultas calculando uma pontuação tf-idf cumulativa
para cada termo na consulta

query_scores = {'a': 0, 'b': 0, 'c': 0}
for term in [t.lower() for t in QUERY_TERMS]:
    for doc in sorted(corpus):
        print 'TF(%s): %s' % (doc, term), tf(term, corpus[doc])
    print 'IDF: %s' % (term, ), idf(term, corpus.values())
    print

    for doc in sorted(corpus):
        score = tf_idf(term, corpus[doc], corpus)
        print 'TF-IDF(%s): %s' % (doc, term), score
        query_scores[doc] += score
    print
```

```
print "Overall TF-IDF scores for query '%s'" % ('
'.join(QUERY_TERMS), )
for (doc, score) in sorted(query_scores.items()):
    print doc, score
```

Ainda que estejamos trabalhando em uma escala diminuta, os cálculos envolvidos funcionam da mesma forma para grandes conjuntos de dados. A tabela 7.3 é uma adaptação consolidada da saída do programa para três consultas de exemplo que envolvem quatro termos distintos:

- "green"
- "Mr. Green"
- "the green plant"

Mesmo que os cálculos da IDF para os termos considerem todo o corpus, eles são exibidos para cada documento, de modo que você pode facilmente verificar as pontuações da TF-IDF analisando uma única linha e multiplicando dois números. Uma vez mais, vale a pena dedicar alguns minutos a esse tópico e compreender os dados da tabela para assimilar como os cálculos funcionam. É incrível o poder da TF-IDF, mesmo não considerando a proximidade ou ordenação das palavras em um documento.

Tabela 7.3 – Cálculos envolvidos em consultas TF-IDF de exemplo, como computadas pelo exemplo 7.4

Documento	tf(mr.)	tf(green)	tf(the)	tf(plant)
corpus['a']	0,1053	0,1053	1,1053	0
corpus['b']	0	0,1111	0	0,1111
corpus['c']	0	0,0625	0	0,0625
	idf(mr.)	**idf(green)**	**idf(the)**	**idf(plant)**
corpus['a']	2,0986	1,0	2,099	1,4055
corpus['b']	2,0986	1,0	2,099	1,4055
corpus['c']	2,0986	1,0	2,099	1,4055
	tf-idf(mr.)	**tf-idf(green)**	**tf-idf(the)**	**tf-idf(plant)**
corpus['a']	0,1053	0,1053	0,1053	0
corpus['b']	0	0,1111	0	0,1111
corpus['c']	0	0,0625	0	0,0625

Os mesmos resultados para cada consulta são mostrados na tabela 7.4, com os valores da TF-IDF somados para cada documento.

Tabela 7.4 – Valores da TF-IDF somados para consultas de exemplo como computadas pelo exemplo 7.4

Consulta	corpus['a']	corpus['b']	corpus['c']
green	0,1053	0,1111	0,0625
Mr. Green	0,1053 + 0,1053 = 0,2105	0 + 0,1111 = 0,1111	0 + 0,0625 = 0,0625
the green plant	0,1053 + 0,1053 + 0 = 0,2105	0 + 0,1111 + 0,1111 = 0,2222	0 + 0,0625 + 0,0625 = 0,1250

De um ponto de vista qualitativo, os resultados da consulta fazem sentido. O documento `corpus['b']` é o vencedor para a consulta "green", com `corpus['a']` logo atrás. Nesse caso, o fator decisivo foi o fato de que o comprimento de `corpus['b']` é menor do que o de `corpus['a']`: a pontuação TF normalizada favorece `corpus['b']` em razão de sua ocorrência individual de "green", ainda que "Green" apareça em `corpus['a']` duas vezes. Como "green" surge em todos os três documentos, o efeito do termo IDF nos cálculos é idêntico. Note, entretanto, que se tivéssemos retornado 0,0, em vez de 1,0, para os cálculos IDF, como feito em algumas implementações, as pontuações da TF-IDF para "green" teriam sido 0,0 para todos os três documentos. Dependendo da situação específica, pode ser melhor retornar 0,0 para as pontuações IDF em vez de 1,0. Por exemplo, se você tivesse 100.000 documentos e "green" estivesse presente em todos eles, você certamente teria de considerá-la uma stopword e remover totalmente seus efeitos sobre uma consulta.

Um ponto mais delicado, que merece observação, é o de que a implementação fornecida no exemplo 7.4 ajusta a pontuação IDF adicionando um valor de 1,0 ao cálculo do logaritmo. Isso é feito para ilustração e porque estamos lidando com um conjunto trivial de documentos. Sem o ajuste de 1,0 ao cálculo, a função `idf` poderia retornar valores menores que 1,0, o que resultaria na multiplicação de duas frações no cálculo da TF-IDF. Como a multiplicação de duas frações resulta em um valor menor que ambas, teríamos um caso perigoso que poderia passar despercebido muito facilmente nos cálculos. Lembre-se de que o raciocínio por trás dos cálculos da TF-IDF é o de que gostaríamos de ser capazes de multiplicar dois termos produzindo consistentemente pontuações TF-IDF maiores para consultas mais relevantes do que para consultas menos relevantes.

Consulta de dados do Buzz com a TF-IDF

Vamos aplicar a TF-IDF aos dados do Buzz que coletamos antes e ver seu desempenho como uma ferramenta para consulta de dados. O NLTK fornece algumas abstrações que podemos utilizar, em vez de criarmos nossas próprias, por isso há muito pouco a fazer agora que você compreende a teoria. O código no exemplo 7.5 supõe que você salvou os dados do Buzz como um arquivo JSON, e permite que você passe vários termos de consulta utilizados para pontuar os documentos por relevância.

Exemplo 7.5 – Consulta de dados do Google Buzz com TF-IDF (buzz___tf_idf_nltk.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import nltk
# Carrega os dados não-estruturados de onde quer que você os
tenha salvo
BUZZ_DATA = sys.argv[1]
buzz_data = json.loads(open(BUZZ_DATA).read())
QUERY_TERMS = sys.argv[2:]
all_posts = [post['content'].lower().split() for post in
buzz_data]
# Fornece abstrações tf/idf/tf_idf
tc = nltk.TextCollection(all_posts)
relevant_posts = []
for idx in range(len(all_posts)):
    score = 0
    for term in [t.lower() for t in QUERY_TERMS]:
        score += tc.tf_idf(term, all_posts[idx])
    if score > 0:
        relevant_posts.append({'score': score, 'title':
buzz_data[idx]['title'],
                               'link': buzz_data[idx]['link']})
# Ordena por pontuação e exibe os resultados
relevant_posts = sorted(relevant_posts, key=lambda p: p['score'],
reverse=True)
for post in relevant_posts:
    print post['title']
    print '\tLink: %s' % (post['link'], )
```

```
    print '\tScore: %s' % (post['score'], )
```

Os resultados de exemplo da consulta para "gov 2.0" nos posts do Buzz de Tim O'Reilly podem ser vistos no exemplo 7.6.

Exemplo 7.6. Resultados de exemplo do exemplo 7.5

```
Flattered but somewhat bemused by the metrics that name me as the
most influential ...
    Link:
http://www.google.com/buzz/107033731246200681024/C62oDoX5oYZ
    Score: 0.0224532639011
Interesting headline on AlleyInsider this morning. "Whoa! Did The
Nook Have Better ...
    Link:
http://www.google.com/buzz/107033731246200681024/hJLh5pf62Hf
    Score: 0.0128109579108
The Importance of Defaults I've been pushing the idea that one of
the big lessons ...
    Link:
http://www.google.com/buzz/107033731246200681024/C1Yh6MLmDeS
    Score: 0.0075401637989
```

Dado um termo de busca, ser capaz de verificar três posts do Buzz classificados por relevância, dentro de uma amostra de praticamente 50, é certamente um passo na direção certa. Tente formular outras consultas, sem esquecer que os valores absolutos das pontuações não são de fato tão importantes – mais valiosa é a habilidade de localizar e ordenar documentos por relevância. Em seguida, pondere as inúmeras formas pelas quais você poderia ajustar ou otimizar essa métrica e torná-la ainda mais efetiva (afinal de contas, todo hacker de dados têm de completar esse exercício em algum momento de sua vida). Uma melhoria óbvia que fica como exercício opcional para o leitor é o uso de raízes (*stems*, em inglês) para os verbos, de forma que variações nas sentenças, funções gramaticais etc. alcancem a mesma raiz e possam ser computadas de modo mais preciso em cálculos de similaridade. O módulo `nltk_stem` fornece implementações de fácil utilização para diversos algoritmos comuns de lematização (*stemming*).

Localização de documentos semelhantes

Assim que você tiver consultado e descoberto documentos de interesse, um de seus próximos objetivos pode ser encontrar documentos

semelhantes (Figura 7.4). Enquanto a TF-IDF fornece meios de analisar um corpus com base em termos de pesquisa, a similaridade de cosseno é uma das técnicas mais comuns para comparação de documentos, atividade essencial para que possamos encontrar documentos semelhantes. Para compreender a similaridade de cosseno é necessária uma breve introdução sobre modelos de espaço vetorial, o tópico de nossa próxima seção.

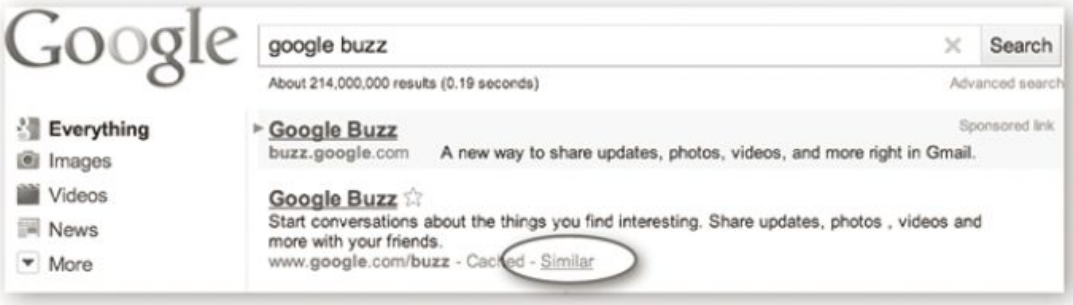


Figura 7.4 – Assim que você tiver encontrado o documento que procura, muitos mecanismos de busca oferecem um recurso do tipo "Find Similar" (encontrar semelhante).

A teoria por trás dos modelos de espaço vetorial e da similaridade de cosseno

Ainda que tenhamos enfatizado o fato de que a TF-IDF modela documentos como conjuntos não-ordenados de palavras, outra forma conveniente de modelar documentos é utilizar um modelo conhecido como *espaço vetorial*. A teoria básica por trás de um modelo de espaço vetorial é a de que você tem um grande espaço multidimensional que contém um vetor para cada documento, e a distância entre dois vetores indica a similaridade dos documentos correspondentes. Uma das características mais belas dos modelos de espaço vetorial é o fato de que você também pode representar uma consulta como um vetor e encontrar os documentos mais relevantes, localizando os vetores de documentos com a menor distância para o vetor da consulta. Ainda que seja virtualmente impossível fazer justiça a este tópico em uma seção tão curta, é importante que você tenha um entendimento básico dos modelos de espaço vetorial caso esteja minimamente interessado na mineração de textos ou no campo da RI. Se você não estiver interessado na teoria e quiser avançar diretamente para os detalhes de implementação, sinta-se à vontade para pular para a próxima seção.

Esta seção supõe um entendimento básico de trigonometria. Caso suas habilidades nessa área estejam um pouco enferrujadas, considere esta uma ótima oportunidade de refrescar seus conhecimentos de matemática do ensino médio.

Primeiramente, pode ser útil esclarecer o que queremos dizer exatamente com o termo "vetor", uma vez que há muitas variações sutis associadas a essa expressão, em diferentes campos de estudo. Em termos gerais, um vetor é uma lista de números que expressa tanto uma direção relativa a uma origem, quanto uma magnitude, a distância dessa origem. Um vetor pode muito naturalmente ser representado como um segmento de linha entre a origem e um ponto em um espaço N-dimensional desenhado como uma linha entre a origem e o ponto. Para ilustrar, imagine um documento definido apenas por dois termos ("Open", "Web"), com um vetor correspondente de (*0,45*, *0,67*), em que os valores no vetor são atribuídos tal qual pontuações TF-IDF para os termos. Em um espaço vetorial, esse documento poderia ser representado em duas dimensões por um segmento de linha que se estendesse da origem, em (*0*,*0*), até o ponto (*0,45*, *0,67*). Em referência a um plano x/y, o eixo x representaria "Open", o eixo y seria "Web" e o vetor de (*0*,*0*) até (*0,45*, *0,67*) seria o documento em questão. Documentos interessantes geralmente contêm no mínimo centenas de termos, mas os mesmos princípios fundamentais podem ser aplicados para modelar documentos nesses espaços de dimensões mais elevadas; é apenas uma situação um pouco mais difícil de visualizar.

Tente fazer a transição da visualização de um documento representado por um vetor com dois componentes, para um representado por três dimensões, como ("Open", "Web" e "Government"). Nesse processo, procure aceitar o fato de que mesmo sendo difícil visualizá-lo, ainda é possível fazer com que um vetor represente dimensões adicionais, difíceis de serem esboçadas ou visualizadas. Se você puder fazer isso, não terá dificuldades para entender que as mesmas operações de vetores que podem ser aplicadas a um espaço bidimensional também podem ser aplicadas a um espaço de 10 dimensões, ou de 367 dimensões. A figura 7.5 mostra um vetor de exemplo em um espaço tridimensional.

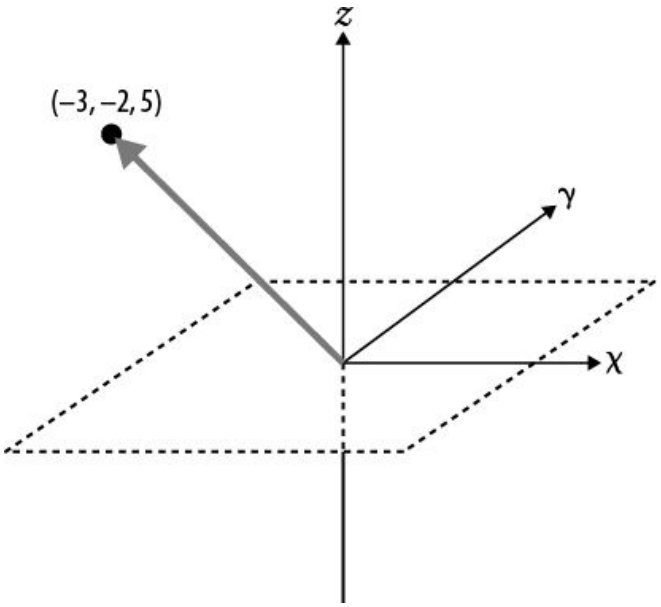


Figura 7.5 – Vetor de exemplo com o valor (–3, –2, 5) representado em um espaço 3D.

Uma vez que é possível modelar documentos como vetores centrados em termos, representando cada termo no documento por sua pontuação TF-IDF correspondente, a tarefa então é determinar qual métrica melhor representa a similaridade entre dois documentos. Em última análise, o cosseno do ângulo entre dois vetores é uma métrica válida para sua comparação, e conhecida como a *similaridade de cosseno* dos vetores. Ainda que essa talvez não seja uma métrica tão intuitiva, anos de pesquisa científica demonstraram que a computação da similaridade de cosseno de documentos representada como vetores de termos é uma métrica muito eficaz (ainda assim, ela sofre de muitos dos problemas da TF-IDF; consulte a seção "Antes que você tente construir um mecanismo de busca...", também deste capítulo, para uma breve sinopse). Construir uma prova rigorosa dos detalhes por trás da métrica de similaridade de cosseno estaria além do escopo deste livro, mas o essencial é que o cosseno do ângulo entre quaisquer dois vetores indica sua similaridade e é equivalente ao produto escalar de seus vetores unitários. Intuitivamente pode lhe ajudar pensar que quanto mais próximos dois vetores estiverem um do outro, menor será o ângulo entre eles e, portanto, maior o cosseno desse ângulo. Dois vetores idênticos teriam um ângulo de 0 grau e uma métrica de similaridade de 1,0, enquanto dois vetores octogonais teriam um ângulo de 90 graus e uma métrica de similaridade de 0,0. O esboço seguinte procura fornecer uma demonstração:

$\overrightarrow{doc1} \cdot \overrightarrow{doc2} = \|doc1\| \cdot \|doc2\| \cdot \cos\Theta$	Dado (por trigonometria)
$\frac{\overrightarrow{doc1} \cdot \overrightarrow{doc2}}{\|doc1\| \cdot \|doc2\|} = \cos\Theta$	Pela divisão
$\hat{doc1} \cdot \hat{doc2} = \cos\Theta$	Pela definição de "vetor unitário"
$\hat{doc1} \cdot \hat{doc2} =$ Similiridade (doc1, doc2)	Pela substituição (suponha: $\cos\Theta$ = Similaridade (doc1, doc2)

Lembrando-se de que um vetor unitário tem um comprimento de 1,0 (por definição), a principal vantagem da computação da similaridade de documentos com vetores unitários está no fato de que eles já estão normalizados, evitando o que poderiam ser variações significativas de comprimento. Depois dessa breve lição matemática, você deve estar ansioso para ver tudo em ação. É exatamente isso o que ocorre na próxima seção.

Agrupamentos de posts utilizando similaridade de cosseno

Se você não pretende utilizar nada da seção anterior, lembre-se ao menos de que, para computar a similaridade entre dois documentos, você precisa apenas produzir um vetor de termos para cada documento e computar o produto escalar dos vetores unitários para esses documentos. Convenientemente, o NLTK expõe a função `nltk.cluster.util.cosine_distance(v1,v2)` para computar a similaridade de cosseno, fazendo com que seja relativamente simples comparar documentos. Como mostra o exemplo 7.7, todo o trabalho necessário ocorre na produção dos vetores de termos apropriados; em resumo, essa função computa vetores de termos para determinado par de documentos atribuindo pontuações TF-IDF para cada componente dos vetores. Entretanto, como o vocabulário exato dos dois documentos provavelmente não é idêntico, espaços reservados com valor de 0,0 devem ser utilizados em cada vetor no lugar de palavras ausentes em um dos documentos, mas presentes no outro. O resultado final é que você acaba com dois vetores de mesmo comprimento, com componentes ordenados de forma idêntica, e que podem ser utilizados para realizar operações de vetores.

Por exemplo, suponha que documento1 contenha os termos (*A*, *B*, *C*) e tenha o vetor correspondente de pesos TF-IDF (*0,10*, *0,15*, *0,12*), enquanto

documento2 contenha os termos (*C*, *D*, *E*) com o vetor correspondente de pesos TF-IDF (*0,05*, *0,10*, *0,09*). O vetor derivado para documento1 seria (*0,10*, *0,15*, *0,12*, *0,0*, *0,0*), e o vetor derivado para documento2 seria (*0,0*, *0,0*, *0,05*, *0,10*, *0,09*). Cada um desses vetores poderia ser passado à função `cosine_distance` do NLTK, que produz a similaridade de cosseno. Internamente, `cosine_distance` utiliza o módulo `numpy` para computar de modo *muito* eficiente o produto escalar dos vetores unitários, alcançando um resultado. Ainda que o código nesta seção reutilize os cálculos de TF-IDF introduzidos previamente, a função exata de pontuação poderia ser qualquer métrica de utilidade. A TF-IDF (ou alguma variação sua), entretanto, é consideravelmente comum em muitas implementações e oferece um ótimo ponto de partida.

O exemplo 7.7 ilustra uma abordagem em que utilizamos a similaridade de cosseno para encontrar o documento mais semelhante a cada documento parte de um corpus de dados do Buzz. Isso poderia ser aplicado da mesma forma a qualquer outro tipo de dados não-estruturados, como posts de blogs, livros etc.

Exemplo 7.7 – Localização de documentos semelhantes utilizando a similaridade de cosseno (buzz__cosine_similarity.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import nltk

# Carrega os dados textuais de onde quer que você os tenha salvo
BUZZ_DATA = sys.argv[1]
buzz_data = json.loads(open(BUZZ_DATA).read())
all_posts = [post['content'].lower().split() for post in
buzz_data]

# Fornece abstrações tf/idf/tf_idf para pontuação
tc = nltk.TextCollection(all_posts)

# Computa uma matriz termo-documento de tal forma que
td_matrix[doc_title][term]
# retorne uma pontuação tf-idf para o termo no documento
td_matrix = {}
for idx in range(len(all_posts)):
    post = all_posts[idx]
    fdist = nltk.FreqDist(post)
```

```
    doc_title = buzz_data[idx]['title']
    link = buzz_data[idx]['link']
    td_matrix[(doc_title, link)] = {}
    for term in fdist.iterkeys():
        td_matrix[(doc_title, link)][term] = tc.tf_idf(term,
post)
# Construa vetores de forma que pontuações de termos estejam nas
mesmas posições...
distances = {}
for (title1, link1) in td_matrix.keys():
    distances[(title1, link1)] = {}
    (max_score, most_similar) = (0.0, (None, None))
    for (title2, link2) in td_matrix.keys():
      # Não altere as estruturas de dados originais
      # uma vez que estamos em um loop e necessitamos deles
muitas vezes
        terms1 = td_matrix[(title1, link1)].copy()
        terms2 = td_matrix[(title2, link2)].copy()
        # Preenche as "lacunas" em cada mapeamento, de forma que
vetores de mesmo comprimento
        # possam ser computados
        for term1 in terms1:
            if term1 not in terms2:
                terms2[term1] = 0
        for term2 in terms2:
            if term2 not in terms1:
                terms1[term2] = 0
        # Cria vetores a partir do mapeamento dos termos
        v1 = [score for (term, score) in sorted(terms1.items())]
        v2 = [score for (term, score) in sorted(terms2.items())]
        # Computa a similaridade entre os documentos
        distances[(title1, link1)][(title2, link2)] = \
            nltk.cluster.util.cosine_distance(v1, v2)
        if link1 == link2:
            continue
        if distances[(title1, link1)][(title2, link2)] >
max_score:
            (max_score, most_similar) = (distances[(title1,
link1)][(title2,
```

```
                                                link2)], (title2,
link2))
    print '''Most similar to %s (%s)
\t%s (%s)
\tscore %s
''' % (title1, link1,
            most_similar[0], most_similar[1], max_score)
```

Se essa discussão pareceu interessante, lembre-se de que *o ponto mais importante é que a consulta de um espaço vetorial representa exatamente a mesma operação feita para computar a similaridade entre documentos, exceto pelo fato de que, em vez de comparar somente vetores de documento, você compara seu vetor de consulta e os vetores de documento*. Em termos de implementação, isso significa construir um vetor contendo seus termos de consulta e compará-lo a cada documento no corpus. Esteja avisado, entretanto, de que a abordagem da comparação direta de um vetor de consulta a todos os vetores de documento possíveis não é uma boa sugestão, mesmo para um corpus de tamanho modesto. Seriam necessárias ótimas decisões de engenharia envolvendo a utilização apropriada de índices se quiséssemos obter uma solução de boa escalabilidade. Isso é algo que você deve considerar antes de decidir criar o próximo grande mecanismo de busca.

Representação da similaridade utilizando visualizações em grafos

Assim como praticamente tudo que estudamos neste livro, há certamente mais do que uma única forma disponível para visualizar similaridade entre itens. A abordagem apresentada nesta seção utiliza estruturas de tipo-grafo, nas quais um link entre os documentos codifica uma medida de sua similaridade. Esta situação é uma ótima oportunidade para apresentar as visualizações do Protovis (*http://vis.stanford.edu/protovis/*), um kit de ferramentas de visualização de alto desempenho, feito em HTML5, desenvolvido pelo Stanford Visualization Group. O Protovis foi especificamente projetado considerando os interesses dos cientistas de dados, oferecendo uma sintaxe declarativa familiar e alcançando um meio termo entre interfaces de alto e baixo-nível. Uma adaptação mínima do exemplo 7.7 é suficiente para emitir um conjunto de nós e arestas que pode ser utilizado para produzir visualizações da galeria de nossos exemplos de dados (*http://vis.stanford.edu/protovis/ex/*). Um loop

aninhado pode computar a similaridade entre os dados de exemplo do Buzz, neste capítulo, e os relacionamentos entre os itens podem ser determinados de acordo com um simples critério de limite estatístico. Os detalhes associados à transformação dos dados e aos ajustes feitos nos templates do Protovis não serão apresentados aqui, mas o código correspondente está disponível para download on-line (*http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/buzz__cosine_similarity_protovis_output.py*).

O código produz o diagrama em arco, apresentado na figura 7.6, e o diagrama de matriz, na figura 7.7. O diagrama em arco produz arcos entre os nós caso haja relacionamento entre eles. Além disso, também escalona os nós de acordo com seus graus, ordenando-os de modo a minimizar confusões e a facilitar a visualização de quais deles têm o maior número de conexões. Títulos são exibidos verticalmente, mas poderiam ter sido omitidos, pois uma tool tip é exibida quando o mouse passa sobre um nó. Ao clicar em um nó você abre uma nova janela no navegador que aponta para a atividade do Buzz representada por ele. Elementos adicionais, como manipuladores de eventos, coloração dos arcos com base em pontuações de similaridade e ajustes dos parâmetros para essa visualização não seriam difíceis de implementar (além de serem uma forma divertida de ocupar seu tempo em um dia chuvoso).

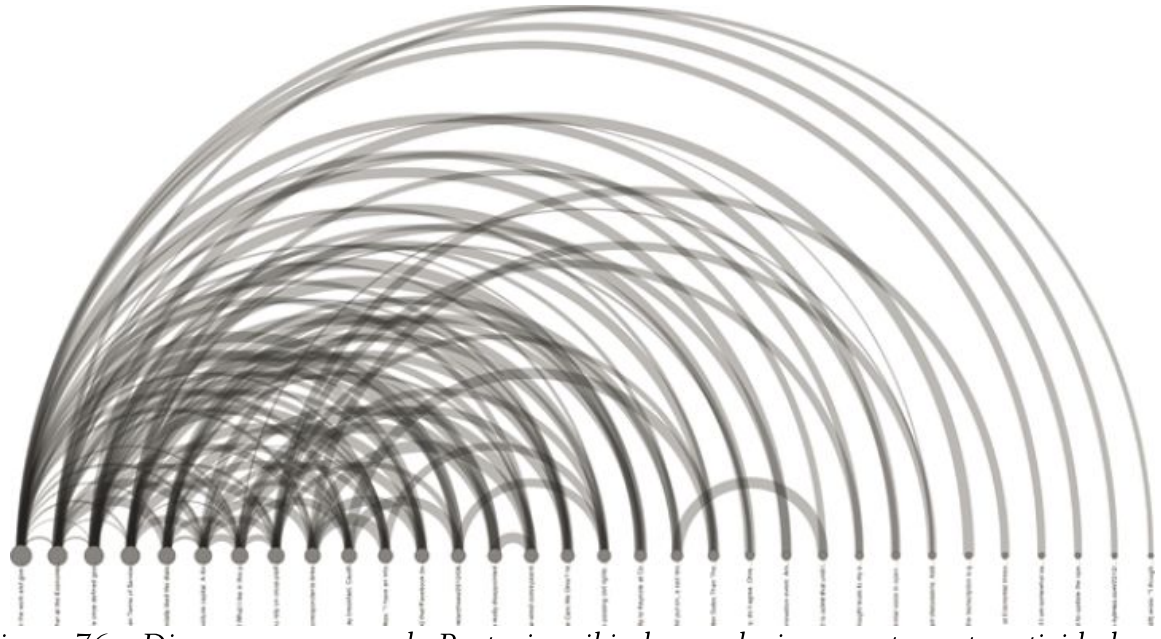


Figura 7.6 – Diagrama em arco do Protovis exibindo os relacionamentos entre atividades do Buzz.

Uma exibição complementar a um diagrama em arco é um diagrama de matriz, em que a intensidade da cor de cada célula varia como uma função da força da correlação da pontuação de similaridade; uma célula mais escura representa maior similaridade. A diagonal da matriz da figura 7.7 foi omitida uma vez que não estamos interessados em fazer com que nossa atenção se volte a essas células, ainda que as pontuações de similaridade para a diagonal pudessem ser perfeitas. Passar o mouse sobre uma célula exibe uma tool tip, indicando a pontuação de similaridade. Uma vantagem dos diagramas de matriz é o fato de que não há potencial para sobreposições desorganizadas entre arestas que representam relacionamentos. Uma desvantagem comumente mencionada de diagramas em arco é a de que eles são difíceis de acompanhar, mas, nesta situação específica, esse processo não é tão importante e, além do mais, seria possível acompanhar o grafo, se você assim o quisesse. Dependendo de suas preferências, e dos dados que está tentando visualizar, qualquer uma, ou até mesmo ambas essas abordagens podem ser apropriadas.

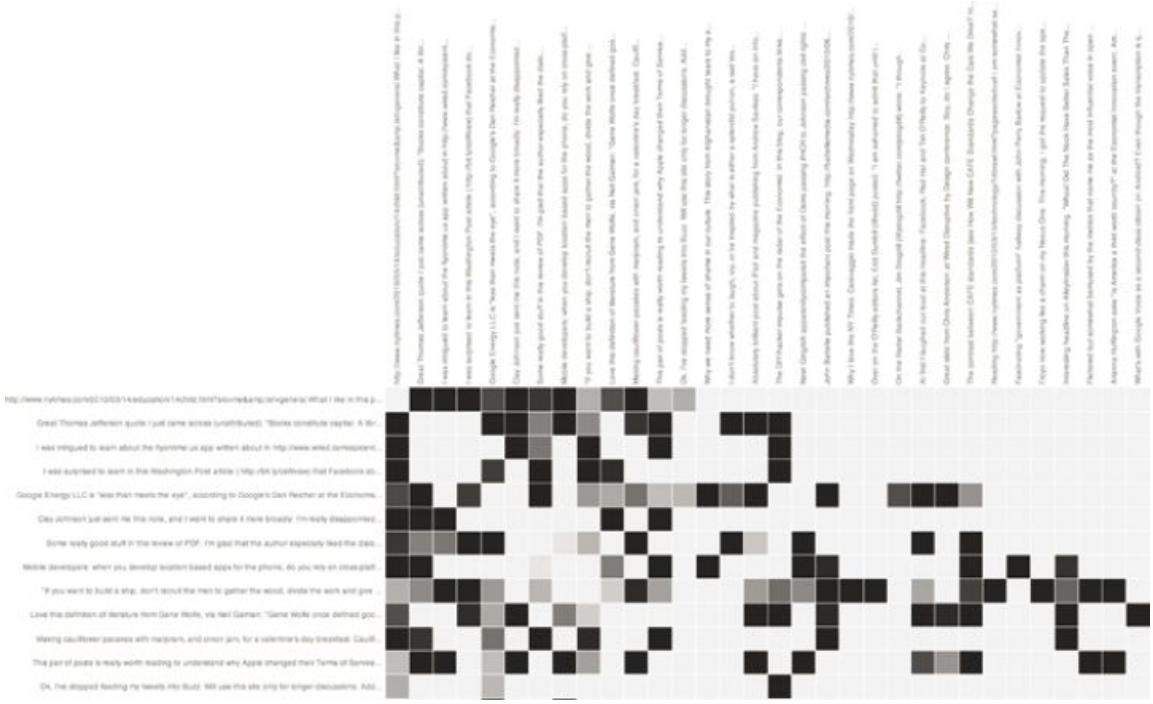


Figura 7.7 – Diagrama de matriz do Protovis exibindo relacionamentos entre atividades do Buzz.

O Buzz e os bigramas

Como mencionamos antes, um problema muitas vezes negligenciado em processamento de textos não-estruturados é a quantidade tremenda de

informações que se tornam disponíveis quando somos capazes de visualizar mais do que somente um token de cada vez, pois muitos dos conceitos que expressamos são frases e não apenas palavras individuais. Por exemplo, se alguém lhe dissesse que alguns dos termos mais comuns de um post são "código", "aberto" e "governo", você seria capaz de dizer se o texto trata de "código aberto", "governo aberto", ambos ou nenhum desses assuntos? Caso tivesse conhecimento prévio acerca do autor ou do conteúdo, você provavelmente poderia formular um palpite mais preciso, mas se dependesse totalmente de uma máquina para *classificar* a natureza de um documento como sendo sobre desenvolvimento colaborativo de software, ou sobre governo transformacional, seria necessário retornar ao texto e determinar de alguma forma quais das palavras ocorrem mais frequentemente antes de "aberto" – por exemplo, você teria de encontrar as *colocações* que terminam com o token "aberto".

Lembre-se do capítulo 6, em que vimos que um *n*-grama é apenas uma forma concisa de expressar todas as sequências possíveis de *n* tokens de um texto, e que *n*-gramas fornecem a estrutura de dados fundamental para computar colocações. Há sempre (*n*-1) *n*-gramas para qualquer valor de *n*, e se você considerasse todos os bigramas (2-gramas) para a sequência de tokens `["Mr.", "Green", "killed", "Colonel", "Mustard"]`, teria quatro possibilidades: `[("Mr.", "Green"), ("Green", "killed"), ("killed", "Colonel"), ("Colonel", "Mustard")]`. Seria necessária uma amostra maior do que apenas nossa frase de exemplo para determinar as colocações, mas, supondo que você tenha em mãos conhecimento prévio ou textos adicionais, o próximo passo será analisar estatisticamente os bigramas para determinar quais deles provavelmente são colocações.

Vale notar que o armazenamento necessário para persistir um modelo *n*-grama requer espaço para *(T–1)*n* tokens (o que é praticamente *T*n*), em que *T* é o número de tokens em questão e *n* é definido pelo tamanho do *n*-grama desejado. Como exemplo, suponha que um documento possua 1.000 tokens, demandando cerca de 8 KB de armazenamento. Para armazenar todos os bigramas do texto seria necessário aproximadamente o dobro do armazenamento original, ou 16 KB, uma vez que você estaria armazenando 999*2 tokens, mais o overhead. Armazenar todos os trigramas para o texto (998*3 tokens, mais o overhead) exigiria aproximadamente o triplo do armazenamento original, ou 24 KB. Assim, sem projetar estruturas especializadas de dados, ou esquemas de compressão, o custo de armazenamento para *n*-gramas pode ser estimado como *n* vezes o requisito original de armazenamento para qualquer valor de n.

n-gramas são muito simples, mas também consideravelmente poderosos como técnica de agrupamento de palavras concorrentes. Se você computar

todos os *n*-gramas para até mesmo um pequeno valor de *n*, provavelmente descobrirá alguns padrões interessantes no próprio texto, sem que para isso seja necessário nenhum trabalho adicional. Por exemplo, considerando os bigramas para um texto suficientemente longo, você provavelmente descobrirá nomes próprios, como "Mr. Green" e "Colonel Mustard", conceitos como "código aberto" ou "governo aberto", e assim por diante. Padrões semelhantes emergem quando você considera trigramas e *n*-gramas para valores de *n* diferentes de três. De fato, a computação de bigramas dessa forma produz essencialmente os mesmos resultados da função `collocations` que executamos em uma sessão anterior do interpretador, exceto pelo fato de que algumas análises estatísticas adicionais consideram o uso de palavras raras. Assim como você aprendeu em uma sessão do interpretador vista antes neste capítulo (Exemplo 7.2.), o NLTK cuida de grande parte do esforço relacionado à computação de *n*-gramas, descobrindo colocações a partir do texto, o contexto em que um token foi utilizado etc. Veja o exemplo 7.8.

Exemplo 7.8 – Uso do NLTK para computar bigramas e colocações para uma frase

```
>>> import nltk
>>> nltk.ngrams("Mr. Green killed Colonel Mustard in the study
with the \
... candlestick. Mr. Green is not a very nice fellow.".split(),
2)
[('Mr.', 'Green'), ('Green', 'killed'), ('killed', 'Colonel'),
 ('Colonel', 'Mustard'), ('Mustard', 'in'), ('in', 'the'),
 ('the', 'study'), ('study', 'with'), ('with', 'the'),
 ('the', 'candlestick.'), ('candlestick.', 'Mr.'), ('Mr.',
'Green'),
 ('Green', 'is'), ('is', 'not'), ('not', 'a'), ('a', 'very'),
 ('very', 'nice'), ('nice', 'fellow.')]
>>> txt = nltk.Text("Mr. Green killed Colonel Mustard in the
study with the\
... candletick. Mr. Green is not a very nice fellow.".split())
>>> txt.collocations()
Building collocations list
Mr. Green
```

Lembre-se de que a única ressalva quanto ao uso das funcionalidades integradas de demonstração, como a de `nltk.Text.collocations`, é a de que essas funções não retornam estruturas de dados que podem ser armazenadas e manipuladas. Sempre que você encontrar tal situação,

analise o código-fonte; geralmente ele poderá ser facilmente adaptado aos seus propósitos. O exemplo 7.9 ilustra como você poderia computar as colocações e os índices de concordância para um conjunto de tokens e manter controle sobre os resultados.

Exemplo 7.9 – Uso do NLTK para computar colocações de forma semelhante à funcionalidade de demonstração de nltk.Text.collocations (buzz___collocations.py)

```
# -*- coding: utf-8 -*-

import sys
import json
import nltk

# Carrega o texto legível por seres humanos de onde quer que você
o tenha salvo

BUZZ_DATA = sys.argv[1]
N = 25
buzz_data = json.loads(open(BUZZ_DATA).read())

all_tokens = [token for post in buzz_data for token in
post['content'
             ].lower().split()]

finder = nltk.BigramCollocationFinder.from_words(all_tokens)
finder.apply_freq_filter(2)
finder.apply_word_filter(lambda w: w in
nltk.corpus.stopwords.words('english'))
scorer = nltk.metrics.BigramAssocMeasures.jaccard
collocations = finder.nbest(scorer, N)

for collocation in collocations:
    c = ' '.join(collocation)
    print c
```

Em resumo, a implementação acompanha livremente a função de demonstração `collocations` do NLTK, filtrando bigramas que não concorrem um número mínimo de vezes (duas, nesse caso) e, então, aplicando uma métrica de pontuação para classificar os resultados. Nessa situação, a função de pontuação é o reconhecido Índice de Jaccard, como definido por `nltk.metrics.BigramAssocMeasures.jaccard`. Uma tabela de contingência é utilizada pela classe `BigramAssocMeasures` para classificar a concorrência dos termos em qualquer bigrama dado, como comparada às possibilidades de outras palavras que poderiam surgir no bigrama. Os detalhes do cálculo de tabelas de contingência e valores Jaccard representam um tópico avançado, mas a próxima seção, "Como se faz o

mix das colocações: tabelas de contingência e funções de pontuação", fornece uma discussão mais estendida sobre esses detalhes, críticos para um bom entendimento da detecção de colocação.

Nesse meio tempo, entretanto, vamos analisar o exemplo 7.10. Esse exemplo mostra uma saída do feed do Buzz de Tim que deve evidenciar que o retorno de bigramas pontuados é uma atividade de poderio imensamente maior em comparação com o retorno de apenas tokens, muito em razão do contexto adicional que oferece, capaz de conferir significado aos termos.

Exemplo 7.10 – Resultados de exemplo para o exemplo 7.9

```
annalee saxenian
nexus one.
cafe standards
certainly true
eric schmidt
olive oil
open source
1/4 cup
free software
andy rubin
front page
mr. o'reilly
o'reilly said.
steve jobs
tech support
long term
web 2.0
"mr. o'reilly
personal brand
came back
cloud computing,
meaningful use
```

Considerando que não empregamos nenhuma heurística ou tática especial que inspecionasse o texto em busca de nomes próprios de acordo com o uso de maiúsculas, é de fato surpreendente que tantos nomes próprios e frases comuns tenham sido verificados nos dados. Ainda há certo ruído inevitável nos resultados, pois não fizemos nada para limpar a pontuação dos tokens, mas para o pouco de trabalho que tivemos, os resultados são muito interessantes. Esta pode ser uma boa oportunidade

para mencionar que, mesmo que capacidades consideráveis de processamento de linguagem natural tivessem sido aplicadas, ainda poderia ser difícil eliminar todo o ruído dos resultados da análise textual. Uma boa sugestão é se acostumar com esse ruído e encontrar heurísticas que possam controlá-lo até que você esteja disposto a realizar um investimento significativo e obter os resultados perfeitos que apenas um leitor humano seria capaz de produzir no texto.

A principal observação que você deve formular nesse momento é a de que, mesmo com muito pouco esforço e tempo investidos, fomos capazes de utilizar outra técnica relativamente simples e descobrir algumas informações valiosas a partir de dados de texto livre, e de que *os resultados parecem ser bem representativos do que já suspeitávamos*. Isso é encorajador, uma vez que sugere que a aplicação das mesmas técnicas ao Buzz de qualquer outra pessoa (ou a qualquer outro tipo de texto não estruturado) poderia potencialmente também ser muito informativa, oferecendo uma rápida análise dos principais itens discutidos. Tão importante quanto isso é o fato de que ainda que os dados neste caso provavelmente confirmem algumas informações que talvez já soubéssemos sobre Tim O'Reilly, você também deve ter aprendido alguns dados novos – por exemplo o fato de que ele aparentemente compartilha receitas ocasionalmente, como evidenciado pela presença dos termos "olive oil" (azeite de oliva) e "1/4 cup" (1/4 de xícara) na listagem dos resultados. Enquanto poderia ser fácil utilizar a concordância, uma expressão regular, ou até mesmo o método `find` da Python para tipos de string, se quiséssemos encontrar posts relacionados a "olive oil", vamos, em vez disso, aproveitar o código que desenvolvemos no exemplo 7.5 e utilizar a TF-IDF para consultar essa expressão. O que temos como retorno? A pesquisa nos diz:

```
Making cauliflower pancakes with marjoram, and onion jam, for a
valentine's day...
    Link:
http://www.google.com/buzz/10703373124620068102 4/h1tciv2psSP
    Score: 0.0437666231649
```

Aí está. A consulta por "olive oil" nos conduz a uma receita de deliciosas panquecas de couve-flor. Efetivamente, partimos de um conhecimento nominal (se tanto) do texto, nos concentramos em alguns tópicos interessantes, utilizando análise de coleções, e buscamos no texto um

desses tópicos utilizando TF-IDF. Não há motivo para que você não possa utilizar a similaridade de cosseno nesse momento, e encontrar o post mais semelhante ao que trata das panquecas de couve-flor (ou ao assunto que você pretende investigar).

Como se faz o mix das colocações: tabelas de contingência e funções de pontuação

Esta seção mergulha em alguns dos detalhes mais técnicos sobre como funciona a BigramCollocationFinder – função de pontuação Jaccard do exemplo 7.9. Se esta é sua primeira leitura do capítulo, ou se você não está interessado nesses detalhes, sinta-se à vontade para pular esta seção e retornar futuramente.

Uma estrutura de dados comum utilizada para computar métricas relacionadas a bigramas é a *tabela de contingência*. O propósito de uma tabela de contingência é expressar de modo compacto as frequências associadas às várias possibilidades de ocorrência de termos diferentes de um bigrama. Analise as entradas em negrito na tabela 7.5, em que *token1* expressa a existência de *token1* no bigrama, e *~token1* expressa que *token1* não existe no bigrama.

Tabela 7.5 – Exemplo de tabela de contingência – valores em itálico representam "marginais", e valores em negrito representam contagens de frequência de variações do bigrama

	token1	~token1	
token2	**frequência(token1, token2)**	**frequência(~token1, token2)**	frequência(*, token2)
~token2	**frequência(token1, ~token2)**	**frequência(~token1, ~token2)**	
	frequência(token1, *)		frequência(*, *)

Ainda que haja alguns detalhes associados ao significado das células para cada cálculo, com sorte não será difícil entender que as quatro células do meio na tabela expressam as frequências associadas à apresentação de vários tokens no bigrama. Os valores nessas células são capazes de computar métricas de similaridade diferentes que podem ser utilizadas para pontuar e classificar bigramas em ordem de provável significância, assim como o caso do Índice de Jaccard que apresentamos antes, e que analisaremos em breve. Primeiro, entretanto, vamos discutir brevemente como os termos da tabela de contingência são computados.

A forma pela qual as diversas entradas na tabela de contingência são computadas está diretamente relacionada às estruturas de dados pré-computadas ou disponíveis. Caso você admita que tem à disposição

apenas uma distribuição de frequência para os vários bigramas no texto, a forma de calcular *frequência(token1, token2)* será uma pesquisa direta; mas e quanto a *frequência(~token1, token2)*? Sem mais nenhuma informação disponível, você teria de verificar cada bigrama individual pela presença de *token2* no segundo slot e subtrair *frequência(token1, token2)* desse valor (faça uma pausa para se convencer de que isso é verdade, caso não lhe pareça óbvio).

Entretanto, se você admitir que além de uma distribuição de frequência disponível que conta as ocorrências de cada token individual no texto (os unigramas do texto), você também tem uma distribuição de frequência dos bigramas, há um atalho *muito menos custoso* que pode ser utilizado nesse sentido, envolvendo duas pesquisas e uma operação aritmética. Subtraia o número de vezes em que *token2* ocorreu como um unigrama do número de vezes em que o bigrama *(token1, token2)* ocorreu, e você terá o número de vezes em que o bigrama *(~token1, token2)* ocorreu. Por exemplo, se o bigrama *("mr.", "green")* ocorreu três vezes e o unigrama *("green")* sete, podemos concluir que o bigrama *(~"mr.", "green")* ocorreu quatro vezes (em que *~"mr."* significa literalmente "qualquer token que não 'mr.'"). Na tabela 7.5, a expressão *frequência(*, token2)* representa o unigrama *token2* e é referida como uma marginal, pois está representada na margem da tabela como um atalho. O valor para *frequência(token1,*)* funciona da mesma forma, ajudando a computar *frequência(token1, ~token2)*, e a expressão *frequência(*, *)* se refere a qualquer unigrama possível, sendo equivalente ao número total de tokens no texto. Dadas *frequência(token1, token2)*, *frequência(token1, ~token2)*, e *frequência(~token1, token2)*, o valor de *frequência\(*, *)* é necessário para calcular *frequência(~token1, ~token2)*.

Ainda que essa discussão sobre tabelas de contingência possa parecer um tanto tangencial, é fundamental para a compreensão das diferentes funções de pontuação. Por exemplo, considere o Índice de Jaccard. Conceitualmente, ele expressa a similaridade de dois conjuntos e é definido por:

$$\frac{|\text{Conjunto1} \cap \text{Conjunto2}|}{|\text{Conjunto1} \cup \text{Conjunto2}|}$$

Em outras palavras, trata-se do número de itens em comum entre dois conjuntos, dividido pelo número total de itens distintos nos conjuntos

combinados. Vale fazer uma pausa e ponderar esse simples, mas eficiente, cálculo. Se *Conjunto1* e *Conjunto2* fossem idênticos, tanto sua união quanto sua intersecção seriam equivalentes, resultando em uma razão de 1,0. Se ambos os conjuntos fossem completamente diferentes, o numerador da razão seria 0, resultando em um valor de 0,0. Além disso, há todos os outros cenários possíveis. O Índice de Jaccard, aplicado a um bigrama específico, expressa a razão entre a frequência de um determinado bigrama e a soma das frequências em que qualquer bigrama contendo um termo do bigrama de interesse ocorre. Uma interpretação dessa métrica poderia ser a de que quanto maior a razão, mais provável que *(token1, token2)* ocorra no texto e, portanto, mais provável que a colocação "token1 token2" expresse um termo significativo.

A seleção da função de pontuação mais apropriada é geralmente determinada de acordo com seu conhecimento acerca das características dos dados, com um pouco de intuição e, às vezes, com uma pitada de sorte. A maioria das métricas de associação definidas em `nltk.metrics.associations` é discutida no capítulo 5 do livro *Foundations of Statistical Natural Language Processing* (MIT Press), de Christopher Manning e Hinrich Schuetze, disponível on-line (*http://nlp.stanford.edu/fsnlp/promo/colloc.pdf*); elas servem como referência útil para as descrições que veremos a seguir. Uma discussão prolongada dessas métricas está fora do escopo deste livro, mas o capítulo promocional que acabamos de mencionar fornece um retrato detalhado com exemplos mais profundos. O Índice de Jaccard, o coeficiente de Dice e a razão de probabilidade são ótimos pontos de partida, caso você tenha de criar seu próprio detector de colocação. Além de alguns outros termos essenciais, essas técnicas são descritas na lista a seguir:

Frequência bruta

Como implica seu nome, frequência bruta é a razão que expressa a frequência de um *n*-grama específico dividida pela frequência de todos os *n*-gramas. Ela é útil para analisar a frequência geral de determinada colocação em um texto.

Índice de Jaccard

O Índice de Jaccard é uma razão que mede a similaridade entre conjuntos. Aplicado às colocações, ele é definido como a frequência de

uma colocação específica dividida pelo número total de colocações que contêm ao menos um termo da colocação de interesse. O Índice de Jaccard é útil para determinar a probabilidade com a qual os termos verificados formam uma colocação, assim como para classificar a probabilidade de colocações prováveis. Utilizando notação consistente com as explanações anteriores, essa fórmula poderia ser definida matematicamente da seguinte maneira:

$$\frac{\text{frequência(token1, token2)}}{\text{frequência(token1, token2)} + \text{frequência(~token1, token2)} + \text{frequência(token1, ~token2)}}$$

Coeficiente de Dice

O coeficiente de Dice é extremamente semelhante ao Índice de Jaccard. A diferença fundamental é que ele atribui peso duas vezes maior às concordâncias entre conjuntos. Sua definição matemática é a seguinte:

$$\frac{2 * \text{frequência(token1, token2)}}{\text{frequência(*, token2)} + \text{frequência(token1, *)}}$$

Matematicamente, podemos mostrar muito facilmente que:

$$\text{Dice} = \frac{2 * \text{Jaccard}}{1 + \text{Jaccard}}$$

Talvez você prefira utilizar essa métrica em vez do Índice de Jaccard, quando for necessário influenciar a pontuação em favor de situações nas quais a colocação "token1 token2" ocorre.

Razão de probabilidade

Esta métrica é mais uma abordagem ao teste de hipóteses, utilizada para medir a independência entre os termos que podem formar uma colocação. Já foi mostrado que ela, em geral, representa uma abordagem mais apropriada para descoberta de colocações do que o teste do qui-quadrado, além de ser adequada ao trabalho com dados que incluem muitas colocações pouco frequentes. Os cálculos específicos envolvidos na computação das estimativas de probabilidade para as colocações, como implementados pelo NLTK, admitem uma distribuição binomial (*http://en.wikipedia.org/wiki/Binomial_distribution*), em que os parâmetros que governam a distribuição são calculados com base no número de ocorrências das colocações e dos termos constituintes.

Qui-quadrado

Assim como a pontuação do teste-t, esta métrica é comumente utilizada para testar a independência entre duas variáveis, e pode ser empregada para medir se dois tokens são colocações com base no teste de significância estatística do qui-quadrado de Pearson. Em termos gerais, as diferenças obtidas entre a aplicação do teste-t e do teste do qui-quadrado não são significativas. A vantagem do teste do qui-quadrado é que, diferentemente dos testes-t, ele não supõe a presença de uma distribuição normal; por esse motivo, testes do qui-quadrado são mais utilizados.

Pontuação-t de Student

Tradicionalmente, a pontuação-t de Student tem sido utilizada para testes de hipóteses e, na forma como são aplicadas à análise de *n*-gramas, pontuações-t podem ser utilizadas para testar a hipótese de que dois termos são colocações. O procedimento estatístico para esse cálculo utiliza uma distribuição-padrão, como de costume para testes-t. Uma vantagem dos valores da pontuação-t, quando comparados aos das frequências brutas, é que uma pontuação-t considera a frequência de um bigrama relativa à de seus componentes. Essa característica facilita a classificação da força das colocações. Uma crítica ao teste-t é que ele necessariamente supõe que a distribuição de probabilidade para as colocações é normal, o que nem sempre se verifica.

Informação Mútua Parcial

A Informação Mútua Parcial (Pointwise Mutual Information, ou PMI) é a medida de quanta informação pode ser obtida acerca de uma palavra específica se você também souber o valor de uma palavra vizinha. Em outras palavras, quanto uma palavra pode nos dizer sobre outra. Ironicamente (no contexto da atual discussão), os cálculos envolvidos na computação da PMI fazem com que ela pontue palavras de alta frequência com valores menores do que os de palavras de baixa frequência, o que é o oposto do efeito desejado. Assim, esta técnica é uma boa forma de medir a independência, mas não a dependência dos termos (por exemplo, é uma escolha não muito adequada para

pontuação de colocações). Também já foi mostrado que ela tem dificuldades ao trabalhar com dados esparsos, e que outras técnicas, como a razão de probabilidade, costumam apresentar um desempenho melhor nesses casos.

Exploração de seus dados do Gmail

O Google Buzz é uma ótima fonte de dados textuais limpos que podemos minerar, mas é apenas um dos muitos pontos de partida disponíveis. Como este capítulo coloca em evidência tecnologias do Google, esta seção fornece uma breve visão geral mostrando como você pode explorar seus dados do Gmail para minerar o texto das mensagens de sua caixa de entrada. Caso você ainda não tenha lido o capítulo 3, lembre-se de que ele trata da análise de e-mails, mas que se concentra principalmente nos recursos de dados estruturados dessas mensagens, como para identificar participantes em discussões. As técnicas destacadas nesse capítulo podem ser aplicadas facilmente às mensagens do Gmail, e vice versa.

Acesso ao Gmail com OAuth

No início de 2010, o Google anunciou acesso OAuth para IMAP e SMTP no Gmail (*http://googlecode.blogspot.com/2010/03/oauth-access-to-imapsmtp-in-gmail.html*). Trata-se de um fato significativo, pois esse anúncio abriu oficialmente as portas do "Gmail como uma plataforma", permitindo que desenvolvedores externos construíssem apps capazes de acessar dados do Gmail, sem que você tivesse de fornecer seu nome de usuário e senha. Esta seção não pretende explicar as nuances específicas de como funciona o Xoauth, implementação do Google para o OAuth (*http://code.google.com/apis/accounts/docs/OAuth.html*); consulte "Não, você não pode saber minha senha", no capítulo 4, para uma introdução concisa ao OAuth. Em vez disso, procuramos ensinar-lhe como trabalhar com esses recursos, para que você possa acessar seus dados do Gmail, seguindo apenas alguns simples passos:

- Selecione a opção "Ativar IMAP" na aba "Encaminhamento e POP/IMAP" de suas configurações de conta no Gmail.
- Visite a página wiki do Google Mail Xoauth Tools (*http://code.google.com/p/google-mail-xoauth-*

tools/wiki/XoauthDotPyRunThrough), faça o download do utilitário de linha de comando `xoauth.py`, e siga as instruções para gerar um token e um segredo OAuth para um consumidor "anônimo"[4].

- Instale o `python-oauth2` (*http://github.com/simplegeo/python-oauth2/*) via `easy_install oauth2` e utilize o template do exemplo 7.11 para estabelecer uma conexão.

Exemplo 7.11 – Template para conexão ao IMAP utilizando OAuth (buzz__gmail_template.py)

```
# -*- coding: utf-8 -*-

import sys
import oauth2 as oauth
import oauth2.clients.imap as imaplib

# Consulte http://code.google.com/p/google-mail-xoauth-
tools/wiki/
# XoauthDotPyRunThrough para mais detalhes sobre o xoauth.py

OAUTH_TOKEN = sys.argv[1] # obtido com o xoauth.py
OAUTH_TOKEN_SECRET = sys.argv[2] # obtido com o xoauth.py
GMAIL_ACCOUNT = sys.argv[3] # exemplo@gmail.com

url = 'https://mail.google.com/mail/b/%s/imap/' %
(GMAIL_ACCOUNT, )

# Valores-padrão para o Xoauth do GMail
consumer = oauth.Consumer('anonymous', 'anonymous')
token = oauth.Token(OAUTH_TOKEN, OAUTH_TOKEN_SECRET)

conn = imaplib.IMAP4_SSL('imap.googlemail.com')
conn.debug = 4 # defina com o nível desejado de debug
conn.authenticate(url, consumer, token)

conn.select('INBOX')

# acesse seus dados de INBOX
```

Assim que você for capaz de acessar seus dados de e-mail, o próximo passo será buscar e analisar alguns dados dessas mensagens.

Busca e parsing de mensagens de e-mails

O protocolo IMAP é uma criatura meticulosa e complexa, mas a boa notícia é que você não precisa saber muito sobre ele para pesquisar e buscar mensagens de e-mail. Exemplos compatíveis como o `imaplib` estão amplamente disponíveis on-line (*http://www.doughellmann.com/PyMOTW/imaplib/*), e uma das operações

mais comuns que você pode desejar fazer é pesquisar por mensagens. Há várias formas pelas quais você é capaz de construir uma consulta IMAP. Um exemplo de como você pesquisaria por mensagens de um usuário específico é `conn.search(None, '(FROM "me")')`, em que `None` é um parâmetro opcional para o conjunto de caracteres e `'(FROM "me")'` é um comando de busca para encontrar mensagens que você mesmo enviou (o Gmail reconhece "me" como o usuário autenticado). Um comando para pesquisa de mensagens contendo "foo" no campo de assunto seria `'(SUBJECT "foo")'`, e há muitas possibilidades adicionais sobre as quais você pode ler na Seção 6.4.4 do RFC 3501 (*http://www.faqs.org/rfcs/rfc3501.html*), que define a especificação do IMAP. O `imaplib` retorna uma resposta à pesquisa na forma de uma tupla que consiste em um código de status e em uma string de IDs de mensagem separados por espaços, encapsulados em uma lista, tal como `('OK', ['506 527 566'])`. Você pode fazer o parsing desses valores de ID para buscar mensagens de e-mail compatíveis com o RFC 822 (*http://www.faqs.org/rfcs/rfc822.html*), mas infelizmente é necessário um pouco de trabalho adicional para fazer o parsing do conteúdo das mensagens de uma forma utilizável. Felizmente, com algumas adaptações mínimas, você pode reutilizar o código do exemplo 3.3, que empregou o módulo `email` para fazer o parsing de mensagens em um formato mais adequado, e cuidar do trabalho necessário para que possamos obter texto utilizável a partir de cada mensagem. O exemplo 7.12 ilustra esse processo.

Exemplo 7.12 – Fluxo de trabalho simples para extração do corpo das mensagens do Gmail retornadas por uma pesquisa (buzz___search_and_parse_mail.py)

```
# -*- coding: utf-8 -*-
import oauth2 as oauth
import oauth2.clients.imap as imaplib
import os
import sys
import email
import quopri
import json
from BeautifulSoup import BeautifulSoup
# Consulte http://code.google.com/p/google-mail-xoauth-
tools/wiki/
# XoauthDotPyRunThrough para detalhes sobre o xoauth.py
```

```
OAUTH_TOKEN = sys.argv[1] # obtido com o xoauth.py
OAUTH_TOKEN_SECRET = sys.argv[2] # obtido com o xoauth.py
GMAIL_ACCOUNT = sys.argv[3] # exemplo@gmail.com
Q = sys.argv[4]
url = 'https://mail.google.com/mail/b/%s/imap/' % (GMAIL_ACCOUNT,
)
# Autentica com o OAuth
# Valores-padrão para a implementação do xoauth do GMail
consumer = oauth.Consumer('anonymous', 'anonymous')
token = oauth.Token(OAUTH_TOKEN, OAUTH_TOKEN_SECRET)
conn = imaplib.IMAP4_SSL('imap.googlemail.com')
conn.debug = 4
conn.authenticate(url, consumer, token)
# Seleciona uma pasta de interesse
conn.select('INBOX')
# Adapta os scripts de "Caixas de e-mail: elas nunca saem de
moda"
def cleanContent(msg):
    # Decodifique a mensagem do formato "quoted printable"
    msg = quopri.decodestring(msg)
   # Remova as tags HTML, se presentes
    soup = BeautifulSoup(msg)
    return ''.join(soup.findAll(text=True))
def jsonifyMessage(msg):
    json_msg = {'parts': []}
    for (k, v) in msg.items():
        json_msg[k] = v.decode('utf-8', 'ignore')
    # Os campos To, CC, e Bcc, se presentes, podem ter múltiplos
itens
    # Note que nem todos esses campos são necessariamente
definidos
    for k in ['To', 'Cc', 'Bcc']:
        if not json_msg.get(k):
            continue
        json_msg[k] = json_msg[k].replace('\n', '').replace('\t',
'').replace('\r'
                , '').replace(' ', '').decode('utf-8',
'ignore').split(',')
try:
```

```
    for part in msg.walk():
        json_part = {}
        if part.get_content_maintype() == 'multipart':
                continue
            json_part['contentType'] = part.get_content_type()
            content = part.get_payload(decode=False).decode('utf-
8', 'ignore')
            json_part['content'] = cleanContent(content)
            json_msg['parts'].append(json_part)
    except Exception, e:
        sys.stderr.write('Skipping message - error encountered
(%s)' % (str(e), ))
    finally:
        return json_msg

# Consuma uma consulta do usuário. Este exemplo ilustra a busca
por assunto

(status, data) = conn.search(None, '(SUBJECT "%s")' % (Q, ))
ids = data[0].split()

messages = []
for i in ids:
    try:
        (status, data) = conn.fetch(i, '(RFC822)')
        messages.append(email.message_from_string(data[0][1]))
    except Exception, e:
      'Print error fetching message %s. Skipping it.' % (i, )

jsonified_messages = [jsonifyMessage(m) for m in messages]

# Separa o conteúdo de texto de cada mensagem, de modo que ele
possa ser analisado

content = [p['content'] for m in jsonified_messages for p in
m['parts']]

# Nota: o conteúdo ainda pode estar bem desorganizado, contendo
muitas quebras de linha
# e outros elementos

if not os.path.isdir('out'):
    os.mkdir('out')

filename = os.path.join('out', GMAIL_ACCOUNT.split("@")[0] +
'.gmail.json')
f = open(filename, 'w')
f.write(json.dumps(jsonified_messages))
f.close()
```

```
print >> sys.stderr, "Data written out to", f.name
```

Assim que você tiver feito o parsing do texto do corpo de uma mensagem do Gmail, pode ser necessário certo trabalho adicional para limpá-lo, até que ele fique pronto para operações avançadas de NLP, ou para exibição, como ilustraremos no capítulo 8; entretanto, não é necessário muito esforço para deixá-lo pronto para a análise de colocação. De fato, os resultados do exemplo 7.12 podem ser alimentados praticamente de maneira direta no exemplo 7.9 para produzir uma lista de colocações dos resultados de busca. Um exercício de visualização muito interessante seria criar um grafo que representasse a força dos relacionamentos entre as mensagens de acordo com o número de bigramas que elas têm em comum, como determinado por uma métrica personalizada.

Antes que você tente construir um mecanismo de busca…

Mesmo que este capítulo possa ter fornecido muitos insights valiosos sobre como extrair informações úteis de textos não-estruturados, mal arranhamos a superfície de conceitos mais fundamentais, tanto em termos de teoria, quanto de considerações de engenharia. Recuperação de informação é literalmente um setor multibilionário, e podemos apenas imaginar a quantidade de investimento combinado que é feita tanto na teoria quanto nas implementações que operam em escala para sustentar mecanismos de busca como o Google e o Yahoo!. Esta seção é uma tentativa modesta de mostrar a você algumas das limitações inerentes da TF-IDF, da similaridade de cosseno, e dos outros conceitos apresentados neste capítulo, com a esperança de que isso ajudará na formação de um panorama geral desse cenário.

Ainda que a TF-IDF seja uma ferramenta poderosa e fácil de utilizar, a implementação específica que fizemos dela tem algumas limitações importantes que convenientemente foram ignoradas, mas que você deve considerar. Uma das mais fundamentais é o fato de que ela trata um documento como um conjunto de palavras, o que significa que a ordem dos termos, tanto no documento quanto na consulta em si, não importa. Por exemplo, consultar “Green Mr.” retornaria os mesmos resultados de “Mr. Green” se não tivéssemos implementado elementos de lógica para

considerar a ordem dos termos na consulta, ou para interpretar a consulta como uma frase, em vez de um par de termos independentes. Obviamente, a ordem em que os termos ocorrem é muito importante.

Mesmo que você efetue uma análise de *n*-gramas para considerar colocações e a ordem dos elementos, ainda há o problema de que a TF-IDF supõe que todos os tokens com o mesmo valor de texto têm o mesmo significado. Evidentemente, entretanto, isso nem sempre é verdade. Qualquer homônimo de sua escolha é um exemplo disso, e eles não são poucos; até palavras que realmente têm o mesmo significado podem apresentar conotações de sentidos levemente diferentes dependendo do contexto exato em que são utilizadas. Uma diferença fundamental entre a tecnologia tradicional de pesquisa de palavras-chave baseada nos princípios da TF-IDF e a utilizada por um mecanismo de busca semântica mais avançado é a de que este mecanismo permite que você fundamente seus termos de pesquisa em um significado específico por meio da definição de um contexto. Por exemplo, você é capaz de especificar se o termo que está buscando deve ser interpretado como uma pessoa, uma localidade, uma organização ou outro tipo específico de entidade. A capacidade de fundamentar termos de busca em contextos específicos é uma área de pesquisa muito ativa no momento.

A similaridade de cosseno também sofre com muitos dos mesmos problemas da TF-IDF. Ela também não considera o contexto do documento ou a ordem dos termos na análise de *n*-gramas, e supõe que termos que ocorrem próximos uns dos outros no espaço vetorial são necessariamente semelhantes, o que certamente nem sempre se verifica. O contraponto óbvio são os homônimos, que podem ter significados consideravelmente diferentes, mas são interpretados como o mesmo termo, pois apresentam os mesmos valores de texto. Nossa implementação específica da similaridade de cosseno também depende da pontuação TF-IDF como forma de computar a importância relativa das palavras nos documentos, assim os erros da TF-IDF têm um efeito cascata.

Você também já deve ter percebido que pode haver muitos detalhes desagradáveis que precisam ser administrados na análise de textos não-estruturados e que acabam sendo muito importantes para implementações avançadas. Por exemplo, comparações de strings diferenciam letras maiúsculas de minúsculas, por isso é importante

normalizar os termos de modo que as frequências possam ser calculadas da forma mais precisa possível. Ainda assim, simplesmente normalizar cegamente os termos aplicando letras minúsculas também pode potencialmente complicar a situação, uma vez que a opção utilizada em certas palavras e frases pode ser muito importante. "Mr. Green" e "Web 2.0" são dois exemplos que merecem consideração. No caso do primeiro nome, manter a letra maiúscula em "Green" pode ser potencialmente vantajoso, pois fornece uma dica apropriada para um algoritmo de consulta, indicando que o termo não se refere a um adjetivo e que provavelmente faz parte de uma locução substantiva. Voltaremos brevemente a tratar desse tópico no capítulo 8, quando discutirmos a NLP, uma vez que em última análise perde-se o *contexto* em que "Green" é utilizado quando consideramos o texto apenas um conjunto de palavras, enquanto um processo de parsing mais avançado com NLP poderia preservar esses dados.

Outra consideração, mais referente à nossa implementação específica do que a uma característica geral da TF-IDF, é o fato de que o uso que fazemos do `split` para tokenizar o texto pode deixar, como parte dos termos, vestígios de pontuação que serão capazes de afetar frequências de tabulação. Por exemplo, no exemplo funcional que vimos antes, `corpus['b']` termina com o token "study", diferente do token "study" que ocorre em `corpus['a']` (o token mais provável a ser pesquisado). Nesse caso, o ponto final à direta do token afeta ambos os cálculos da TF e da IDF.

Talvez seja interessante executar a lematização das palavras, para que variações usuais de uma dada palavra sejam essencialmente tratadas como o mesmo termo, em vez de termos diferentes. Consulte o pacote nltk.stem para muitas implementações interessantes de lematização.

Por fim, há muitas considerações de engenharia que devem ser ponderadas caso você decida implementar uma solução que será aplicada em uma situação de produção. O uso de índices e de armazenamento em cache são considerações críticas para obtenção de intervalos razoáveis de consulta em conjuntos de dados de tamanho considerável. A habilidade de analisar quantidades verdadeiramente significativas de dados textuais em sistemas de processamento em lotes, como o Hadoop[5], mesmo em infraestrutura de nuvem com custos razoáveis, como a Elastic Compute Cloud, da Amazon, pode ter um alto custo e exigir que você disponha do orçamento de uma corporação de tamanho médio.

Comentários finais

Este capítulo apresentou alguns dos fundamentos da teoria da RI: TF-IDF, similaridade de cosseno e colocações. Dado o imenso poder de provedores de busca como o Google, é fácil esquecer que tais técnicas fundamentais sequer existem. Entretanto, ao compreendê-las, você tem acesso a muitas das suposições e limitações presentes no status quo atualmente aceito dos mecanismos de busca, além de diferenciar claramente as técnicas avançadas centradas em entidades que estão surgindo (o capítulo 8 apresenta uma mudança fundamental de paradigma, distanciando-se das ferramentas deste capítulo, e deve evidenciar ainda mais tais diferenças, caso você ainda não tenha lido esse material). Se você pretende aplicar as técnicas deste capítulo à web (em geral), pode ser interessante verificar o Scrapy (*http://scrapy.org*), um framework estabelecido e de fácil utilização para raspagem e rastreamento de dados.

---

1 Este livro evita discutir detalhes específicos referentes à diferença exata que pode haver entre expressões comuns como "mineração de texto", "análise de dados não-estruturados (UDA)" ou "recuperação de informação", tratando todos esses termos como representativos essencialmente da mesma atividade.

2 O que não deixa de ser engraçado, uma vez que o Google é reconhecido por suas capacidades de busca e este capítulo concentra-se em técnicas de busca fundamentais centradas em documentos.

3 A palavra "the" corresponde a 7% dos tokens no Brown Corpus (*http://en.wikipedia.org/wiki/Brown_Corpus*) e representa um ponto de partida razoável para um corpus, caso você ainda não saiba nada sobre ele.

4 Se você está explorando seus próprios dados do Gmail, não há problemas em utilizar as credenciais de consumidor anônimo geradas pelo xoauth.py; no futuro, você sempre poderá se registrar e criar uma aplicação-cliente "confiável" (*http://code.google.com/apis/accounts/docs/RegistrationForWebAppsAuto.html*), quando se tornar apropriado fazê-lo.

5 Se o Hadoop lhe parece interessante, pode valer a pena conferir o Dumbo (*http://github.com/klbostee/dumbo/wiki/building-and-installing*), projeto que permite escrever e executar programas Hadoop em Python.

CAPÍTULO 8

Blogs et al.: Processamento de linguagem natural (e muito mais)

Este capítulo é uma tentativa modesta de apresentar o Processamento de linguagem natural (Natural Language Processing, ou NLP) e de aplicá-lo aos dados não-estruturados dos blogs. Seguindo o espírito dos capítulos anteriores, este capítulo tenta oferecer o nível mínimo de detalhes para que você formule um sólido conhecimento geral de um tópico inerentemente complexo, enquanto, ao mesmo tempo, oferece detalhes técnicos o suficiente para que você seja capaz de iniciar imediatamente a mineração de seus dados. Ainda que tenhamos até aqui utilizado todos os atalhos possíveis para facilitar o entendimento de nossas explicações, e empregado uma abordagem de tipo-Pareto – mostrando 20% das habilidades cruciais que você pode utilizar para realizar 80% do trabalho – os atalhos que adotaremos neste capítulo merecem um destaque especial, uma vez que o NLP é um tópico consideravelmente complexo. Nenhum capítulo de nenhum livro, nem mesmo uma pequena coleção de livros, poderia fazer justiça a este tópico. Este capítulo é uma introdução pragmática que deve fornecer informações suficientes para que você seja capaz de realizar algumas coisas incríveis, como gerar automaticamente abstracts a partir de documentos e extrair listas de entidades importantes. Ainda assim, evitaremos nos aprofundar demasiadamente em tópicos que certamente exigiriam várias dissertações para serem explicados.

Mesmo que não seja absolutamente necessário ler o capítulo 7 antes deste capítulo, é altamente recomendado que você o faça. Um bom entendimento do Processamento de linguagem natural pressupõe uma compreensão e um conhecimento funcional de algumas das forças e fraquezas fundamentais da TF-IDF, dos modelos de espaço vetorial etc. Nesse sentido, estes dois capítulos têm um vínculo um pouco mais forte do que a maioria dos outros capítulos deste livro. A fonte de dados específica que utilizaremos são os blogs, mas, assim como dissemos no

capítulo anterior, qualquer outra fonte poderia ser utilizada. Escolhemos os blogs, pois eles são o componente principal da web social (*http://radar.oreilly.com/2010/10/why-blogging-still-matters.html*) e inerentemente adequados à mineração de textos. Além disso, a linha divisória entre posts de blogs e artigos está se tornando cada vez menos significativa (*http://www.slate.com/id/2271184/pagenum/all/*).

NLP: uma introdução ao estilo Pareto

A seção de abertura deste capítulo é essencialmente uma discussão explicativa que procura ilustrar as dificuldades do NLP e fornecer um entendimento sólido sobre como ele se difere das técnicas apresentadas em capítulos anteriores. A seção seguinte, entretanto, aborda aspectos práticos, apresentando alguns exemplos de código para que você possa iniciar suas atividades.

Sintaxe e semântica

Você deve se lembrar, como visto no capítulo 7, de que talvez a fraqueza mais significativa da TF-IDF e da similaridade de cosseno é que esses modelos não requerem inerentemente uma compreensão *semântica* mais profunda dos dados. Pelo contrário, os exemplos do capítulo 7 empregaram uma sintaxe apenas básica que separava tokens na presença de espaços em branco, quebrando um documento opaco, em um conjunto de tokens, e utilizaram métricas de frequência e de similaridade estatística simples para determinar quais tokens eram provavelmente os mais importantes nos dados. Ainda que seja possível realizar procedimentos incríveis com essas técnicas, elas não fornecem nenhuma noção sobre o significado de fato de cada token no contexto em que ele ocorre no documento. Basta uma frase contendo um homógrafo (*http://en.wikipedia.org/wiki/Homograph*)[1], como "fish" (peixe, mas também o verbo pescar, em inglês) ou "bear" (urso, mas também o verbo sustentar, tolerar, em inglês), para demonstrar esse fato; qualquer uma dessas expressões poderia ser um substantivo ou um verbo.

Aplicar NLP é algo inerentemente complexo e pode ser difícil fazê-lo com um mínimo de qualidade; empregá-lo com perfeição a um grande conjunto de idiomas usuais talvez seja o maior desafio do século. Afinal, um domínio completo do NLP seria praticamente sinônimo do

desempenho máximo no Teste de Turing (*http://en.wikipedia.org/wiki/Turing_test*) e, para um observador cuidadoso, um programa de computador que conseguisse esse feito demonstraria um grau incrível de inteligência humana. Enquanto recursos estruturados ou semiestruturados são essencialmente coleções de registros, com algum significado pressuposto adicionado a cada campo que pode ser imediatamente analisado, há considerações mais sutis que devem ser manipuladas quando falamos de dados de linguagem natural, mesmo para as tarefas aparentemente mais simples. Por exemplo, suponha que é dado a você um documento e pede-se que conte o número de frases nele. Essa é uma tarefa trivial se você for um ser humano e tiver um entendimento mínimo de seu idioma. Porém, a realidade é outra para uma máquina, que exigirá um conjunto complexo e detalhado de instruções para completar a mesma tarefa.

A boa notícia é que máquinas são capazes de detectar com muita rapidez, e precisão praticamente perfeita, o término das frases, desde que trabalhem com dados relativamente bem-formados. Entretanto, mesmo que você tenha detectado corretamente todas as frases, ainda há muito que você provavelmente não sabe sobre como as palavras e expressões são utilizadas nelas. Por exemplo, considere a expressão clássica dos anos 1990, "That's the bomb" (que pode ser traduzida como "essa é a bomba", ou então, em sentido figurado, como "isso é o máximo", em inglês) (*http://www.nydailynews.com/archives/entertainment/1997/06/08/1997-06-08_that_s_the_bomb__kids__slang.html*)[2]. Trata-se de uma frase que pode ser analisada com facilidade, consistindo praticamente em nada mais do que um sujeito, um predicado e um objeto. Entretanto, sem contexto adicional, mesmo que você tenha informações perfeitas sobre os componentes da frase, não tem como saber o significado da palavra "bomb" – ela pode significar tanto "algo muito divertido", quanto um explosivo capaz de imensa destruição. O que queremos dizer é que, mesmo com informações perfeitas sobre a estrutura de uma frase, você deve compreender ao que se refere o pronome "that"[3] e formular algumas deduções para decidir se ele faz ou não referência a um explosivo perigoso. Assim, generalizando, podemos dizer que o NLP trata fundamentalmente de, a partir de um documento simples, consistindo em uma coleção ordenada de símbolos que aderem a uma *sintaxe* correta e a uma *gramática* bem-definida, deduzir a *semântica* associada relacionada a

esses símbolos.

Um breve exercício de raciocínio

Vamos retornar à detecção de frases, o primeiro passo da maioria dos processos de NLP, para ilustrar um pouco a complexidade que essa tarefa representa. Pode ser fácil superestimar a utilidade de heurísticas simples com base em regras, por isso é importante acompanhar um exercício para que você perceba quais são alguns dos problemas centrais desse processo, e não perca seu tempo reinventando a roda.

Sua primeira tentativa para solucionar o problema de detecção de frases pode ser simplesmente contar os pontos finais, de interrogação e de exclamação do período. Esta é a heurística mais óbvia que temos de início, mas é relativamente rudimentar e tem o potencial de produzir uma margem extremamente alta de erros. Considere a seguinte acusação (bem inequívoca):

> "Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow."

A simples tokenização do período, dividindo o texto de acordo com a pontuação (especificamente pontos finais), produziria o seguinte resultado:

```
>>> txt = "Mr. Green killed Colonel Mustard in the study with the
\
... candlestick. Mr. Green is not a very nice fellow."
>>> txt.split(".")
['Mr', 'Green killed Colonel Mustard in the study with the
candlestick',
 'Mr', 'Green is not a very nice fellow', '']
```

Deve ser imediatamente óbvio que realizar a detecção de frases simplesmente separando o texto nos pontos finais, sem incorporar alguma espécie de noção de contexto ou de informações de alto-nível, é insuficiente. Neste caso, o problema é o uso de "Mr.", uma abreviação válida comumente utilizada no idioma inglês. Ainda que já saibamos desde o capítulo 7 que, se tivéssemos uma quantidade maior de texto para analisar, uma análise de *n*-gramas deste exemplo provavelmente poderia nos dizer que "Mr. Green" representa uma colocação ou um *chunk* (token composto contendo espaço em branco), não é difícil imaginar outros casos que seriam de difícil detecção com base na ocorrência de

colocações. Pensando um pouco à frente, também vale destacar que encontrar os tópicos principais de uma frase não é algo fácil quando estamos munidos de uma lógica apenas trivial. Como um ser humano inteligente, você pode facilmente discernir que os tópicos principais de nosso exemplo devem ser "Mr. Green", "Colonel Mustard", "the study" e "the candlestick", mas treinar uma máquina para dizer-lhe o mesmo é uma tarefa complexa. Algumas alternativas óbvias provavelmente devem estar lhe ocorrendo nesse momento, como realizar a detecção de letras maiúsculas com uma expressão regular, construir uma lista de abreviações usuais para fazer o parsing correto das locuções substantivas, ou aplicar alguma variação dessa lógica ao problema de localização dos limites finais das frases, para impedir que você tenha problemas desse tipo.

OK, está bem. Admitimos que essas práticas poderiam funcionar para alguns exemplos, mas qual será a margem de erro para um texto típico em inglês? Quão tolerante é seu algoritmo a um texto mal-escrito; a informações abreviadas, como mensagens de texto ou tweets; ou a outros idiomas, como espanhol, francês ou italiano? Não há respostas simples nesse sentido, e é exatamente por isso que análise de textos é um tópico tão importante em uma era na qual a quantidade de dados textuais disponíveis cresce literalmente a cada segundo. Não destacamos esses pontos para desencorajá-lo. De fato, mencionamos esses fatos com o objetivo de motivá-lo a não desistir quando encontrar dificuldades, pois este é um universo inerentemente complexo e ainda não conquistado. Mesmo que você se sinta desmotivado, esse sentimento não será duradouro, uma vez que, como veremos na próxima seção, o NLTK tem um desempenho bem razoável em muitas situações que envolvem textos específicos.

Um processo típico de NLP com o NLTK

Este tópico acompanha uma sessão do interpretador que realiza NLP com o NLTK. O processo do NLP que seguiremos é típico e se assemelha ao seguinte fluxo de alto-nível:

Detecção de término de frase →

Tokenização →

Indicação de classe gramatical →

Criação de chunks →

Extração

Utilizaremos o seguinte texto de exemplo: "Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow." Lembre-se de que, mesmo que você já tenha lido o texto, e compreenda que ele é composto por duas frases e por muitos outros elementos, para uma máquina, ele é apenas um simples valor em string. Vamos analisar mais detalhadamente os passos que devemos percorrer:

Detecção de término de frase (End of Sentence, ou EOS Detection)

Este passo quebra um texto em uma coleção de frases significativas. Como frases geralmente representam unidades lógicas de pensamento, costumam ter uma sintaxe previsível adequada a uma análise mais profunda. A maioria dos processos de NLP que você verá tem início com esse passo, pois a tokenização (o próximo passo) opera sobre frases individuais. Quebrar o texto em parágrafos ou seções poderia ser útil para certos tipos de análise, mas é improvável que nos auxilie na tarefa geral da detecção do término das frases. No interpretador, faça o parsing de uma frase com o NLTK da seguinte maneira:

```
>>> import nltk
>>> txt = "Mr. Green killed Colonel Mustard in the study with
the candlestick. \
... Mr. Green is not a very nice fellow."
>>> sentences = nltk.tokenize.sent_tokenize(txt)
>>> sentences
['Mr. Green killed Colonel Mustard in the study with the
candlestick.',
 'Mr. Green is not a very nice fellow.']
```

Falaremos um pouco mais sobre o que está acontecendo internamente com `sent_tokenize` na próxima seção. Por ora, vamos simplesmente aceitar que a detecção de frases foi feita corretamente para um texto específico – uma melhora evidente quando comparada ao desempenho que teríamos com a quebra do texto considerando apenas sinais de pontuação.

Tokenização

Este passo opera sobre frases individuais, dividindo-as em tokens. Seguindo a sessão de exemplo do interpretador, você faria o seguinte:

```
>>> tokens = [nltk.tokenize.word_tokenize(s) for s in
sentences]
>>> tokens
[['Mr.', 'Green', 'killed', 'Colonel', 'Mustard', 'in', 'the',
'study', 'with',
  'the', 'candlestick', '.'],
['Mr.', 'Green', 'is', 'not', 'a', 'very', 'nice', 'fellow',
'.']]
```

Note que, para este simples exemplo, a tokenização parece ter tido o mesmo resultado da separação do texto nos espaços em branco, com a exceção de que removeu corretamente marcadores de término de frase (os pontos finais). Entretanto, como veremos em uma seção futura, se assim quisermos, a tokenização pode realizar ainda mais, e já sabemos que nem sempre é trivial a distinção entre um ponto que marca um término de frase e um que indica uma abreviação.

Indicação de classe gramatical

Este passo atribui informações de classe gramatical a cada token. Na sessão de exemplo do interpretador, os tokens percorrem mais um passo para receberem as tags:

```
>>> pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]
>>> pos_tagged_tokens
[[('Mr.', 'NNP'), ('Green', 'NNP'), ('killed', 'VBD'),
('Colonel', 'NNP'),
  ('Mustard', 'NNP'), ('in', 'IN'), ('the', 'DT'), ('study',
'NN'),
  ('with', 'IN'), ('the', 'DT'), ('candlestick', 'NN'), ('.',
'.')],
 [('Mr.', 'NNP'), ('Green', 'NNP'), ('is', 'VBZ'), ('not',
'RB'), ('a', 'DT'),
  ('very', 'RB'), ('nice', 'JJ'), ('fellow', 'JJ'), ('.',
'.')]]
```

Talvez você não compreenda intuitivamente todas essas tags, mas elas representam informações de classe gramatical. Por exemplo, 'NNP' indica que o token é um substantivo parte de uma locução substantiva. 'VBD' indica um verbo que está no tempo passado simples, e 'JJ' indica um adjetivo. O Penn Treebank Project (*http://www.cis.upenn.edu/~treebank/*) fornece um resumo completo (*http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_po*

s.html) das tags de classe gramatical que poderiam ser retornadas. Terminado esse processo, deve estar se tornando aparente o quão poderosa pode ser nossa análise. Por exemplo, utilizando tags de classes gramaticais, seremos capazes de reunir substantivos como partes de locuções substantivas e, então, tentar decidir que tipo de entidades elas podem representar (por exemplo, pessoas, lugares, organizações etc.).

Criação de chunks

Este passo, também chamado de *chunking*, envolve a análise de cada token marcado dentro de uma frase, seguida pela reunião de tokens compostos que expressam conceitos lógicos – abordagem bem diferente da análise estatística de colocações. É possível definir uma gramática personalizada por meio do `chunk.RegexpParser` do NLTK, mas isso está além do escopo deste capítulo; consulte o capítulo 9 ("Building Feature Based Grammars") do livro *Natural Language Processing with Python* (O'Reilly), disponível on-line em *http://nltk.googlecode.com/svn/trunk/doc/book/ch09.html*, para maiores detalhes. Além disso, o NLTK expõe uma função que combina *chunking* com extração de entidades nomeadas, nosso próximo passo.

Extração

Este passo envolve a análise de cada chunk e sua identificação como entidades nomeadas, sejam elas pessoas, organizações, localizações etc. A continuidade do processo no interpretador demonstra como isso é feito:

```
>>> ne_chunks = nltk.batch_ne_chunk(pos_tagged_tokens)
>>> ne_chunks
[Tree('S', [Tree('PERSON', [('Mr.', 'NNP')]), Tree('PERSON',
[('Green', 'NNP')]),
    ('killed', 'VBD'),
 Tree('ORGANIZATION', [('Colonel', 'NNP'), ('Mustard',
'NNP')]), ('in', 'IN'),
    ('the', 'DT'), ('study', 'NN'), ('with', 'IN'), ('the',
'DT'),
    ('candlestick', 'NN'), ('.', '.')]),
 Tree('S', [Tree('PERSON', [('Mr.', 'NNP')]),
Tree('ORGANIZATION',
```

```
    [('Green', 'NNP')]), ('is', 'VBZ'), ('not', 'RB'),
  ('a', 'DT'), ('very', 'RB'), ('nice', 'JJ'), ('fellow',
'JJ'), ('.', '.')])]
>>> ne_chunks[0].draw() # Você pode desenhar cada chunk na
árvore
```

Não fique tão preocupado em tentar decifrar exatamente qual seria a saída na árvore. Em resumo, ela reúne alguns tokens e tenta classificá-los como certos tipos de entidades (você pode ser capaz de discernir que ela identificou "Mr. Green" como uma pessoa, mas infelizmente categorizou "Colonel Mustard" como uma organização). A figura 8.1 ilustra o efeito obtido se chamarmos `draw()` nos resultados de `nltk.batch_ne_chunk`.

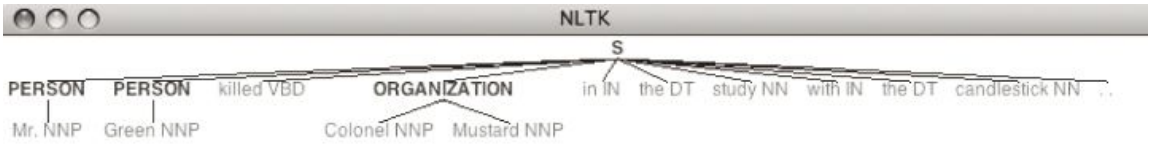


Figura 8.1 – Você pode fazer a interface do NLTK com kits de ferramentas de desenho, para inspecionar a saída agrupada de forma mais intuitiva do que a representação em texto bruto vista no interpretador.

Mesmo que talvez fosse interessante continuar falando sobre as complexidades do NLP, sobre a produção de uma implementação avançada, ou até mesmo sobre uma análise mais profunda do NLTK aplicado ao NLP, não é isso que pretendemos nesta seção. As informações de fundo oferecidas aqui têm como objetivo fazer com que você compreenda a dificuldade dessa tarefa, mas também encorajá-lo a consultar o livro sobre NLTK que mencionamos (*http://www.nltk.org/book*), ou qualquer outro dos muitos recursos disponíveis on-line, caso queira se aprofundar no tópico. Como conselho, caso seu negócio (ou projeto) dependa de uma implementação realmente avançada do NLP, considere adquirir uma licença de um produto de uma instituição comercial ou acadêmica, em vez de tentar criar você mesmo uma solução. Há muito que pode ser feito com software de código aberto, e esse é certamente um ótimo ponto de partida, mas, assim como tudo mais, pode ser necessário um grande investimento se você tiver de realizar diversas melhorias que demandem muita participação de consultores especializados. O NLP ainda é um campo de pesquisa muito ativo, e suas implementações mais avançadas estão longe de se tornarem algo

corriqueiro.

A menos que mencionado o contrário, o restante deste capítulo supõe que você esteja utilizando o NLTK sem alterações (se você tem PhD em linguística computacional, ou algo desse tipo, deve ser mais do que capaz de modificar o NLTK de acordo com suas necessidades e, provavelmente, está lendo textos muitos mais avançados do que este capítulo).

Tendo concluído essa breve introdução ao NLP, vamos minerar alguns dados de blogs.

Detecção de frases em blogs com o NLTK

Dado que a detecção de frases é provavelmente a primeira tarefa pela qual você se interessará quando estiver trabalhando em sua implementação do NLP, faz sentido iniciarmos por ela. Mesmo que você nunca complete as tarefas seguintes do processo que verificamos, uma simples detecção de término de frase (*End of Sentence detection*) permite algumas possibilidades muito poderosas, como o resumo de documentos, que consideraremos em um exercício futuro. Entretanto, primeiro teremos de buscar alguns dados de alta-qualidade. Vamos utilizar o consagrado módulo `feedparser`, que você pode instalar com o `easy_install`, caso ainda não o tenha feito, para buscar alguns posts do blog Radar da O'Reilly (*http://radar.oreilly.com*). O código no exemplo 8.1 busca alguns posts, salvando-os em um arquivo local como JSON simples, uma vez que nada neste capítulo depende das capacidades de uma mídia de armazenamento mais avançada, como o CouchDB. Como sempre, você pode escolher armazenar os posts em outro local, se preferir.

Exemplo 8.1 – Coleta de dados de blogs fazendo o parsing dos feeds (blogs_and_nlp__get_feed.py)

```
# -*- coding: utf-8 -*-

import os
import sys
from datetime import datetime as dt
import json
import feedparser
from BeautifulSoup import BeautifulStoneSoup
from nltk import clean_html

# Feed de exemplo: http://feeds.feedburner.com/oreilly/radar/atom
```

```
FEED_URL = sys.argv[1]
def cleanHtml(html):
    return BeautifulStoneSoup(clean_html(html),
                              convertEntities=BeautifulStoneSoup.
HTML_ENTITIES).contents[0]
fp = feedparser.parse(FEED_URL)
print "Fetched %s entries from '%s'" % (len(fp.entries[0].title),
fp.feed.title)
blog_posts = []
for e in fp.entries:
    blog_posts.append({'title': e.title, 'content'
                       : cleanHtml(e.content[0].value), 'link':
e.links[0].href})
if not os.path.isdir('out'):
    os.mkdir('out')
out_file = '%s__%s.json' % (fp.feed.title, dt.utcnow())
f = open(os.path.join(os.getcwd(), 'out', out_file), 'w')
f.write(json.dumps(blog_posts))
f.close()
print >> sys.stderr, 'Wrote output file to %s' % (f.name, )
```

Obter nosso texto já estruturado e a partir de uma fonte respeitável nos permite presumir o uso de uma gramática correta; esperançosamente, isso também significa que um dos detectores de frases incorporados do NLTK funcionará adequadamente. Não há melhor forma de descobrir qual será o resultado do que criar algum código e ver o que acontece, por isso vá em frente e analise o código do exemplo 8.2. Ele apresenta os métodos `sent_tokenize` e `word_tokenize` do NLTK, que representam respectivamente o detector de sentenças e o tokenizador de palavras recomendados pelo NLTK. Faremos uma breve discussão sobre o código posteriormente.

Exemplo 8.2 – Uso das ferramentas de NLP do NLTK para parsing dos dados do blog (blogs_and_nlp__sentence_detection.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import nltk
# Carrega a saída de blogs_and_nlp__get_feed.py
BLOG_DATA = sys.argv[1]
```

```
blog_data = json.loads(open(BLOG_DATA).read())
# Personalize sua lista de stopwords como necessário. Aqui,
adicionamos
# artefatos comuns de pontuação e contração.
stop_words = nltk.corpus.stopwords.words('english') + [
    '.',
    ',',
    '--',
    '\'s',
    '?',
    ')',
    '(',
    ':',
    '\'',
    '\'re',
    '"',
    '-',
    '}',
    '{',
    ]
for post in blog_data:
    sentences = nltk.tokenize.sent_tokenize(post['content'])
    words = [w.lower() for sentence in sentences for w in
             nltk.tokenize.word_tokenize(sentence)]
    fdist = nltk.FreqDist(words)
    # Estatísticas básicas
    num_words = sum([i[1] for i in fdist.items()])
    num_unique_words = len(fdist.keys())
    # Hapaxes são palavras de uma única ocorrência
    num_hapaxes = len(fdist.hapaxes())
    top_10_words_sans_stop_words = [w for w in fdist.items() if
w[0]
                                    not in stop_words][:10]
    print post['title']
    print '\tNum Sentences:'.ljust(25), len(sentences)
    print '\tNum Words:'.ljust(25), num_words
    print '\tNum Unique Words:'.ljust(25), num_unique_words
    print '\tNum Hapaxes:'.ljust(25), num_hapaxes
    print '\tTop 10 Most Frequent Words (sans stop
words):\n\t\t', \
```

```
        '\n\t\t'.join(['%s (%s)'
        % (w[0], w[1]) for w in top_10_words_sans_stop_words])
print
```

O primeiro ponto que deve ter chamado sua atenção são as chamadas a `sent_tokenize` e `word_tokenize`. O NLTK fornece diversas opções para tokenização, mas apresenta "recomendações" quando às melhores disponíveis por meio desses métodos. Quando da redação deste livro (você pode verificar quando estiver lendo por meio do `pydoc`), o detector de frase recomendado era o `PunktSentenceTokenizer` e o tokenizador de palavras, o `TreebankWordTokenizer`. Vamos analisar ambos brevemente.

Internamente, o `PunktSentenceTokenizer` depende da capacidade que apresenta de detectar abreviações como parte de padrões de colocação, e utiliza algumas expressões regulares para tentar fazer o parsing inteligente de sentenças, considerando padrões usuais do uso de pontuações. Uma explicação completa da lógica do `PunktSentenceTokenizer` está fora do escopo deste livro, mas o texto original de Tibor Kiss e Jan Strunk, "Unsupervised Multilingual Sentence Boundary Detection" (*http://www.linguistics.ruhr-uni-bochum.de/~strunk/ks2005FINAL.pdf*), discute essa abordagem. Sua leitura é prazerosa e aconselhamos que você faça uma pausa para analisá-lo. Como veremos em breve, é possível instanciar o `PunktSentenceTokenizer` com textos de amostra, para que ele treine e melhore sua precisão. O tipo de algoritmo utilizado é de aprendizagem não supervisionada; ele não requer que você faça nenhuma marcação dos dados de treinamento. Em vez disso, o algoritmo inspeciona certas *características* que ocorrem no texto em si, como uso de letras maiúsculas, concorrências de tokens etc., para derivar parâmetros adequados à divisão do texto em sentenças.

Ainda que o `WhitespaceTokenizer` do NLTK, que cria tokens quebrando o texto quando encontra espaços em branco, possa ser o tokenizador mais simples que poderíamos apresentar, você já conhece alguns dos problemas desse tipo de quebra inconsequente, feita considerando apenas espaços em branco. Em vez disso, o NLTK atualmente recomenda o `TreebankWordTokenizer`, um tokenizador de palavras que opera sobre frases e utiliza os mesmos padrões do Penn Treebank Project[4]. A única característica que pode surpreendê-lo é que a tokenização do `TreebankWordTokenizer` (*http://www.cis.upenn.edu/~treebank/tokenization.html*) realiza algumas

operações não tão óbvias, como a indicação separada de componentes em contrações e pronomes na forma possessiva. Por exemplo, fazer o parsing da frase "I'm hungry" (Estou faminto) produziria componentes separados para "I" e "m", mantendo uma distinção entre sujeito e verbo para "I'm". Como você pode imaginar, ter acesso granular a esse tipo de informação gramatical poderá ser muito importante quando chegar o momento de realizar uma análise mais avançada que verifique relacionamentos entre sujeitos e verbos nas frases.

Se você tem dificuldades com tokenizadores avançados como o `TreebankWordTokenizer` ou o `PunktWordTokenizer` do NLTK, pode retornar ao `WhitespaceTokenizer` até que decida se utilizar um tokenizador mais avançado vale o investimento. Por exemplo, utilizar um tokenizador avançado em dados que frequentemente incluem URLs pode ser uma má ideia, pois esses tokenizadores não reconhecem habitualmente URLs e, por isso, costumam dividi-los erroneamente em vários tokens. Não é do escopo deste livro implementar um tokenizador personalizado, mas há muitos recursos on-line que você pode consultar, se pretende fazê-lo.

Tendo agora um tokenizador de frases e outro de palavras, podemos fazer o parsing do texto em frases e depois em tokens individuais. Note que ainda que essa abordagem seja bem intuitiva, tem como calcanhar de Aquiles o fato de que erros produzidos pelo detector de frase poderão se propagar no processo e potencialmente prejudicar o limite de qualidade que o restante da implementação do NLP é capaz de produzir. Por exemplo, caso o tokenizador de frases quebre equivocadamente uma frase no ponto seguinte a "Mr.", em uma seção de texto como "Mr. Green killed Colonel Mustard in the study with the candlestick", pode não ser possível extrair a entidade "Mr. Green" do texto, a menos que haja uma lógica especializada de reparo posicionada. Novamente, tudo depende da sofisticação da implementação completa do NLP e de como ela lida com a propagação de erros. O `PunktSentenceTokenizer` convencional é treinado utilizando o corpus do Penn Treebank e apresenta um desempenho muito adequado. O objetivo final do parsing é instanciar um prático objeto `FreqDist`, que espera uma lista de tokens. O restante do código no exemplo 8.2 representa o uso direto de algumas das APIs mais utilizadas do NLTK.

O objetivo desta seção foi familiarizá-lo com o primeiro passo envolvido na construção de um processo de NLP. Nesse sentido, desenvolvemos algumas métricas que fizeram uma simples tentativa de caracterizar alguns dados de blogs. Nosso processo não envolveu a indicação de classes gramaticais, nem chunking (por enquanto), mas deve ter lhe fornecido um entendimento básico acerca de alguns dos conceitos mais

importantes, além de tê-lo feito pensar sobre algumas das questões mais sutis envolvidas. Mesmo que pudéssemos ter simplesmente feito nossas quebras utilizando espaços em branco, e depois contado os termos, computado os resultados e obtido muitas informações a partir dos dados, não demorará muito para que você perceba como valeu a pena ter dado os passos iniciais, como uma forma de se aproximar de um entendimento mais profundo dos dados. Para ilustrar uma aplicação possível dentro do que você acabou de aprender, na próxima seção veremos um algoritmo simples para resumo de documentos que depende praticamente apenas da segmentação de frases e da análise de frequências.

Resumo de documentos

Ainda que talvez não pareça imediatamente óbvio, ser capaz de realizar uma boa detecção de frases, como parte da abordagem do NLP para mineração de dados não-estruturados, pode permitir algumas capacidades de mineração de texto muito poderosas, como tentativas rudimentares, mas razoáveis, de criar resumos para os textos. Há inúmeras possibilidades e abordagens nesse sentido, mas uma das mais simples que podemos empregar data da edição de abril de 1958 do *IBM Journal*. No influente artigo "The Automatic Creation of Literature Abstracts," H.P. Luhn descreve uma técnica que essencialmente se resume à filtragem de frases contendo palavras que ocorrem com frequência e próximas umas das outras.

O artigo original é de fácil compreensão e muito interessante; nele, Luhn chega mesmo a descrever como preparou cartões perfurados para realizar diversos testes com parâmetros diferentes. É incrível pensar que o que hoje podemos implementar com algumas poucas dúzias de linhas de código Python em um hardware de baixo custo demandou de Luhn horas e horas de programação em um mainframe colossal. O Exemplo 8.3 fornece uma implementação básica do algoritmo de Luhn para resumo de documentos. Uma breve análise do algoritmo segue na próxima seção. Antes de avançar para essa discussão, faça uma pausa e analise o código, vendo se compreende como ele funciona.

Exemplo 8.3 – Algoritmo de resumo de documento (blogs_and_nlp__summarize.py)

```
# -*- coding: utf-8 -*-
import sys
```

```
import json
import nltk
import numpy

N = 100 # Número de palavras a ser considerado
CLUSTER_THRESHOLD = 5 # Distância entre palavras a ser
considerada
TOP_SENTENCES = 5 # Número de frases a retornar para um resumo
das "n principais" frases

# Abordagem obtida em "The Automatic Creation of Literature
Abstracts" por H.P. Luhn

def _score_sentences(sentences, important_words):
    scores = []
    sentence_idx = -1

    for s in [nltk.tokenize.word_tokenize(s) for s in sentences]:

        sentence_idx += 1
        word_idx = []

      # Para cada palavra na lista de palavras...
        for w in important_words:
            try:
                # Computa um índice para o local em que ocorrem
palavras importantes na frase
                word_idx.append(s.index(w))
            except ValueError, e: # w não nessa frase específica
                pass

        word_idx.sort()

        # É possível que algumas frases não contenham nenhuma
palavra importante
        if len(word_idx)== 0: continue

        # Empregando o índice de palavras, computa agrupamentos
utilizando um limite de distância
        # máxima para duas palavras consecutivas

        clusters = []
        cluster = [word_idx[0]]
        i = 1
        while i < len(word_idx):
            if word_idx[i] - word_idx[i - 1] < CLUSTER_THRESHOLD:
                cluster.append(word_idx[i])
            else:
                clusters.append(cluster[:])
                cluster = [word_idx[i]]
            i += 1
```

```
        clusters.append(cluster)
        # Pontua cada agrupamento. A pontuação máxima de qualquer
agrupamento é a pontuação
        # da frase
        max_cluster_score = 0
        for c in clusters:
            significant_words_in_cluster = len(c)
            total_words_in_cluster = c[-1] - c[0] + 1
            score = 1.0 * significant_words_in_cluster \
               * significant_words_in_cluster /
total_words_in_cluster
            if score > max_cluster_score:
                max_cluster_score = score
        scores.append((sentence_idx, score))
    return scores
def summarize(txt):
    sentences = [s for s in nltk.tokenize.sent_tokenize(txt)]
    normalized_sentences = [s.lower() for s in sentences]
    words = [w.lower() for sentence in normalized_sentences for w
in
             nltk.tokenize.word_tokenize(sentence)]
    fdist = nltk.FreqDist(words)
    top_n_words = [w[0] for w in fdist.items()
            if w[0] not in
nltk.corpus.stopwords.words('english')][:N]
    scored_sentences = _score_sentences(normalized_sentences,
top_n_words)
    # 1ª abordagem de resumo:
    # Filtre frases não-significativas utilizando a pontuação
média somada a uma
    # fração do desvio padrão como filtro
    avg = numpy.mean([s[1] for s in scored_sentences])
    std = numpy.std([s[1] for s in scored_sentences])
    mean_scored = [(sent_idx, score) for (sent_idx, score) in
scored_sentences
                   if score > avg + 0.5 * std]
    # 2ª abordagem de resumo:
    # Outra abordagem seria retornar apenas as N principais
frases
    top_n_scored = sorted(scored_sentences, key=lambda s: s[1])[-
```

```
TOP_SENTENCES:]
    top_n_scored = sorted(top_n_scored, key=lambda s: s[0])
    # Acrescenta os resumos ao objeto do post
    return dict(top_n_summary=[sentences[idx] for (idx, score) in
top_n_scored],
                mean_scored_summary=[sentences[idx] for (idx,
score) in mean_scored])
if __name__ == '__main__':
    # Carrega a saída de blogs_and_nlp__get_feed.py
    BLOG_DATA = sys.argv[1]
    blog_data = json.loads(open(BLOG_DATA).read())
    for post in blog_data:
        post.update(summarize(post['content']))
        print post['title']
        print '-' * len(post['title'])
        print
        print '-------------'
        print 'Top N Summary'
        print '-------------'
        print ' '.join(post['top_n_summary'])
        print
        print '--------------------'
        print 'Mean Scored Summary'
        print '--------------------'
        print ' '.join(post['mean_scored_summary'])
        print
```

Como entrada/saída de exemplo, utilizaremos um post de Tim O'Reilly no Radar, intitulado "The Louvre of the Industrial Age" (*http://radar.oreilly.com/2010/07/louvre-industrial-age-henry-ford.html*), com cerca de 460 palavras e reimpresso aqui para que você possa comparar o exemplo da saída das duas tentativas de criação de resumo vistas no código:

> This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it's so much more than that. As I wrote in my first stunned tweet, "it's the Louvre of the Industrial Age."
>
> When we first entered, Marc took us to what he said may be his favorite artifact in the museum, a block of concrete that contains Luther Burbank's shovel, and

Thomas Edison's signature and footprints. Luther Burbank was, of course, the great agricultural inventor who created such treasures as the nectarine and the Santa Rosa plum. Ford was a farm boy who became an industrialist; Thomas Edison was his friend and mentor. The museum, opened in 1929, was Ford's personal homage to the transformation of the world that he was so much a part of. This museum chronicles that transformation.

The machines are astonishing – steam engines and coal-fired electric generators as big as houses, the first lathes capable of making other precision lathes (the makerbot of the 19th century), a ribbon glass machine that is one of five that in the 1970s made virtually all of the incandescent lightbulbs in the world, combine harvesters, railroad locomotives, cars, airplanes, even motels, gas stations, an early McDonalds' restaurant and other epiphenomena of the automobile era.

Under Marc's eye, we also saw the transformation of the machines from purely functional objects to things of beauty. We saw the advances in engineering – the materials, the workmanship, the design, over a hundred years of innovation. Visiting The Henry Ford, as they call it, is a truly humbling experience. I would never in a hundred years have thought of making a visit to Detroit just to visit this museum, but knowing what I know now, I will tell you confidently that it is as worth your while as a visit to Paris just to see the Louvre, to Rome for the Vatican Museum, to Florence for the Uffizi Gallery, to St. Petersburg for the Hermitage, or to Berlin for the Pergamon Museum. This is truly one of the world's great museums, and the world that it chronicles is our own.

I am truly humbled that the Museum has partnered with us to hold Makerfaire Detroit on their grounds. If you are anywhere in reach of Detroit this weekend, I heartily recommend that you plan to spend both days there. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford. P.S. Here are some of my photos from my visit. (More to come soon. Can't upload many as I'm currently on a plane.)

Esta manhã, tive a oportunidade de fazer uma visita ao The Henry Ford Museum em Dearborn, MI, acompanhado de Dale Doughery, criador da revista Make: e da Makerfaire, e de Marc Greuther, curador-chefe do museu. Esperava encontrar um museu dedicado à indústria automobilística, mas ele é muito mais que isso. Como escrevi em meu primeiro e impressionado tweet, o museu "é o Louvre da Era Industrial."

Logo que entramos, Marc nos levou ao que disse ser seu artefato favorito no museu, um bloco de concreto que contém a pá de Luther Burbank, além da assinatura e das pegadas de Thomas Edison. Luther Burbank foi, como sabemos, um grande cientista agrícola que criou alguns tesouros como a nectarina e a ameixa Santa Rosa. Ford era um garoto do campo, que se tornou industrialista; Thomas Edison foi seu amigo e mentor. O museu, inaugurado em 1929, representa a homenagem pessoal de Ford à transformação do mundo da qual ele fazia parte. Este museu registra essa transformação.

As máquinas são impressionantes – temos motores a vapor e geradores movidos a carvão grandes como casas, os primeiros tornos capazes de criar outros tornos de precisão (principal ferramenta de criação do século XIX), uma máquina para trabalho com vidro que é uma das cinco que, nos anos 70 do século XX, criaram praticamente todas as lâmpadas incandescentes do planeta, colheitadeiras, locomotivas, carros, aviões, e até mesmo hotéis de estrada, postos de gasolina e um antigo McDonalds, além de outras criações contemporâneas da era do automóvel.

Sob o olhar de Marc, também vimos a transformação das máquinas, evoluindo de criações puramente funcionais até se tornarem objetos de beleza. Vimos os avanços da engenharia – nos materiais, no trabalho e no design, durante cerca de cem anos de inovação. Visitar o "Henry Ford", como o chamam, é uma verdadeira lição de humildade, eu nunca teria imaginado que recomendaria uma visita a Detroit apenas para conhecer esse museu, mas sabendo do que sei agora, posso dizer-lhe com confiança que uma viagem desse tipo vale a pena, da mesma forma que visitar Paris apenas para conhecer o Louvre, ou Roma para conhecer o Museu do Vaticano, Florença pela Galeria Uffizi, São Petersburgo pelo Hermitage, ou Berlin pelo Museu Pergamon. Este é um dos mais incríveis museus do mundo, e a história que ele registra é a nossa própria história.

Sinto-me verdadeiramente honrado pela parceria que o museu fez conosco para receber a Makerfaire Detroit em suas instalações. Se você estiver próximo a Detroit nesse fim de semana, recomendo sem ressalvas que planeje uma visita de dois dias à exposição. Você pode facilmente dedicar um dia à Makerfaire, e outro ao Henry Ford. PS: Veja algumas fotos de minha visita. (Em breve postarei outras. Não posso fazer o upload de muitas, pois estou em um avião.)

A filtragem de frases utilizando uma pontuação média e desvio padrão resulta em um resumo de cerca de 170 palavras:

This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it's so much more than that. As I wrote in my first stunned tweet, "it's the Louvre of the Industrial Age. This museum chronicles that transformation. The machines are astonishing - steam engines and coal fired electric generators as big as houses, the first lathes capable of making other precision lathes (the makerbot of the 19th century), a ribbon glass machine that is one of five that in the 1970s made virtually all of the incandescent lightbulbs in the world, combine harvesters, railroad locomotives, cars, airplanes, even motels, gas stations, an early McDonalds' restaurant and other epiphenomena of the automobile era. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford.

Esta manhã, tive a oportunidade de fazer uma visita ao The Henry Ford Museum em Dearborn, MI, acompanhado de Dale Doughery, criador da revista Make: e da Makerfaire, e de Marc Greuther, curador-chefe do museu. Esperava

> encontrar um museu dedicado à indústria automobilística, mas ele é muito mais que isso. Como escrevi em meu primeiro e impressionado tweet, o museu "é o Louvre da Era Industrial". Este museu registra essa transformação. As máquinas são impressionantes – temos motores a vapor e geradores movidos a carvão grandes como casas, os primeiros tornos capazes de criar outros tornos de precisão (principal ferramenta de criação do século XIX), uma máquina para trabalho com vidro que é uma das cinco, que nos anos 1970, criaram praticamente todas as lâmpadas incandescentes do planeta, colheitadeiras, locomotivas, carros, aviões, e até mesmo hotéis de estrada, postos de gasolina e um antigo McDonalds, além de outras criações contemporâneas da era do automóvel. Você pode facilmente dedicar um dia à Makerfaire, e outro a Henry Ford.

Uma abordagem alternativa de resumo, considerando as *N* principais frases (em que *N* = 5 neste caso), produz um resultado um pouco mais condensado, com cerca de 90 palavras, ainda mais sucinto, mas que continua muito informativo:

> This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it's so much more than that. As I wrote in my first stunned tweet, "it's the Louvre of the Industrial Age. This museum chronicles that transformation. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford.
>
> Esta manhã, tive a oportunidade de fazer uma visita ao The Henry Ford Museum em Dearborn, MI, acompanhado de Dale Doughery, criador da revista Make: e da Makerfaire, e de Marc Greuther, curador-chefe do museu. Esperava encontrar um museu dedicado à indústria automobilística, mas ele é muito mais que isso. Como escrevi em meu primeiro e impressionado tweet, o museu "é o Louvre da Era Industrial". Este museu registra essa transformação. Você pode facilmente dedicar um dia à Makerfaire, e outro ao Henry Ford.

Assim como qualquer outra situação que envolva processos de análise, muitos dados podem ser obtidos inspecionando resumos e comparando-os ao texto completo. Produzir a saída em um formato simples de marcação, que possa ser aberto em praticamente todos os navegadores, é muito simples: basta ajustar o trecho final do script que realiza a saída para que ele faça algumas substituições de strings. O exemplo 8.4 ilustra uma possibilidade.

Exemplo 8.4 – Alteração da saída do exemplo 8.3 visando produzir marcação HTML adequada para análise dos resultados do algoritmo de resumo (blogs_and_nlp__summarize_markedup_output.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import json
import nltk
import numpy
from blogs_and_nlp__summarize import summarize

HTML_TEMPLATE = """<html>
    <head>
        <title>%s</title>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
    </head>
    <body>%s</body>
</html>"""
if __name__ == '__main__':
    # Carregue a saída de blogs_and_nlp__get_feed.py
    BLOG_DATA = sys.argv[1]
    blog_data = json.loads(open(BLOG_DATA).read())
    # A versão com marcação pode ser escrita no disco
    if not os.path.isdir('out/summarize'):
        os.makedirs('out/summarize')
    for post in blog_data:
        post.update(summarize(post['content']))
    # Você também poderia armazenar uma versão do post completo,
com as principais frases
    # destacadas para análise, utilizando uma simples
substituição de strings...
    for summary_type in ['top_n_summary', 'mean_scored_summary']:
        post[summary_type + '_marked_up'] = '<p>%s</p>' %
(post['content'], )
        for s in post[summary_type]:
            post[summary_type + '_marked_up'] = \
            post[summary_type + '_marked_up'].replace(s, \
              '<strong>%s</strong>' % (s, ))
        filename = post['title'] + '.summary.' + summary_type +
'.html'
        f = open(os.path.join('out', 'summarize', filename), 'w')
        html = HTML_TEMPLATE % (post['title'] + \
          ' Summary', post[summary_type + '_marked_up'],)
        f.write(html.encode('utf-8'))
```

```
    f.close()
    print >> sys.stderr, "Data written to", f.name
```

A saída resultante é o texto completo do documento, com as frases que compõem o resumo destacadas em negrito (Figura 8.2). À medida que você explora técnicas alternativas para criação de resumos, ter a capacidade de verificar rapidamente o resultado nas abas do navegador pode lhe mostrar de modo muito intuitivo a semelhança entre as diferentes técnicas. A principal diferença aqui ilustrada é a presença de uma frase consideravelmente longa (e descritiva), próxima do meio do documento, que inicia com as palavras "The machines are astonishing" (As máquinas são impressionantes).

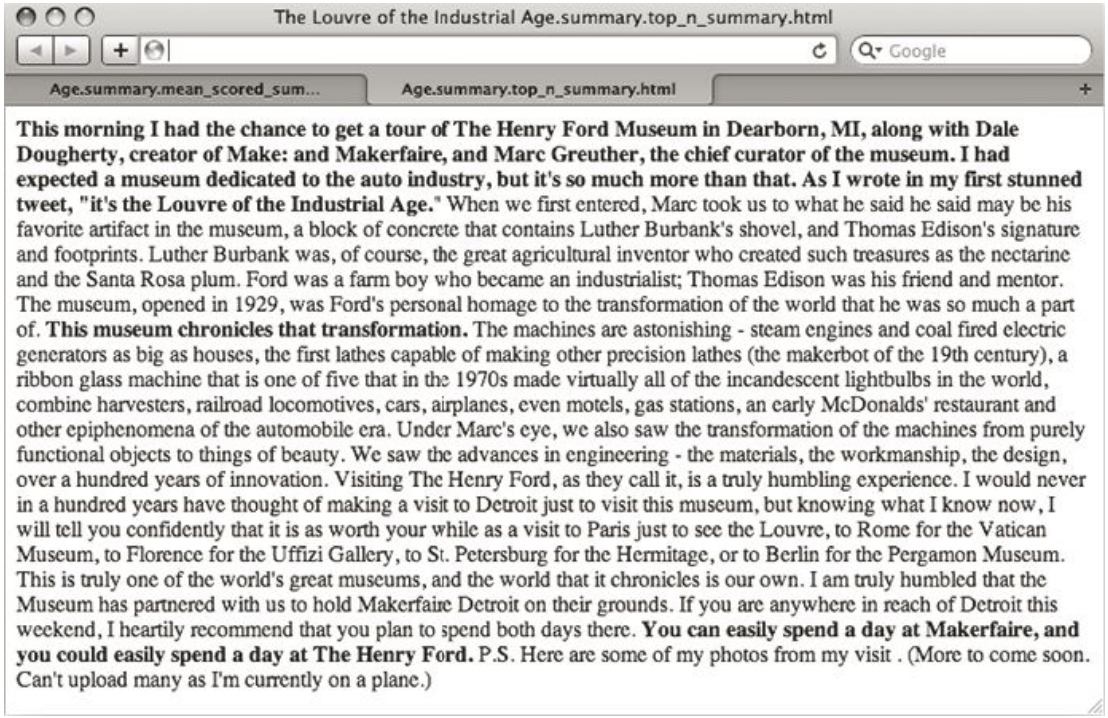


Figura 8.2 – Texto de um post no blog Radar da O'Reilly que marca em negrito as frases mais importantes, determinadas por um algoritmo de criação de resumos.

A próxima seção apresenta uma breve discussão acerca da abordagem de Luhn.

Análise do algoritmo para criação de resumos de Luhn

A premissa básica por trás do algoritmo de Luhn é a de que as frases mais importantes em um documento são as que contêm palavras que ocorrem com frequência. Entretanto, há alguns detalhes que merecem destaque. Primeiro, nem todas as palavras que ocorrem com frequência são

importantes; em termos gerais, stopwords são apenas termos secundários e quase nunca de interesse para análise. Lembre-se de que ainda que tenhamos filtrado as stopwords usuais na implementação de exemplo, também pode ser possível criar uma lista personalizada de stopwords para qualquer blog ou domínio, desde que tenhamos conhecimento adicional a priori sobre seus termos, para que possamos reforçar esse algoritmo ou qualquer outro que suponha a filtragem dessas palavras. Por exemplo, um blog escrito exclusivamente sobre baseball pode utilizar com tamanha frequência a palavra "baseball" que talvez seja interessante adicioná-la a uma lista de stopwords, mesmo ela não sendo uma stopword em termos gerais. Como nota paralela, seria interessante incorporar a TF-IDF à função de pontuação para uma fonte específica de dados, como forma de lidar com palavras usuais no linguajar do domínio.

Supondo que tenha sido feita uma tentativa razoável de eliminar stopwords, o próximo passo no algoritmo deve ser escolher um valor adequado para *N* e selecionar as *N* principais palavras como base da análise. Note que a suposição desse algoritmo é que essas *N* principais palavras são suficientemente descritivas para caracterizar a natureza do documento, e que, para quaisquer duas frases no documento, a frase que contém o maior número dessas palavras será considerada mais descritiva. Tudo que é preciso fazer depois de determinar as "palavras mais importantes" do documento é aplicar uma heurística a cada frase, e filtrar alguns subconjuntos de frases que poderão ser utilizados como resumo ou abstract do documento. A pontuação de cada frase é feita na função `score_sentences`. É nela que ocorre grande parte das operações mais interessantes do código.

Para pontuar cada frase, o algoritmo em `score_sentences` aplica um simples limite de distância para agrupar documentos e pontua cada agrupamento de acordo com a seguinte fórmula:

$$\frac{(\text{palavras significativas no agrupamento})^2}{\text{palavras totais no agrupamento}}$$

A pontuação final de cada frase é igual à pontuação mais alta para qualquer agrupamento que nela ocorre. Vamos considerar os passos abrangentes envolvidos em `score_sentences` para uma frase de exemplo, e ver como essa abordagem funciona na prática:

Entrada: Frase de exemplo

```
['Mr.', 'Green', 'killed', 'Colonel', 'Mustard', 'in', 'the',
'study',
'with', 'the', 'candlestick', '.']
```

Entrada: Lista de palavras importantes

```
['Mr.', 'Green', 'Colonel', 'Mustard', 'candlestick']
```

Entrada/Suposição: Limite de agrupamento (distância)

```
3
```

Computação intermediária: Agrupamentos detectados

```
[ ['Mr.', 'Green', 'killed', 'Colonel', 'Mustard'],
['candlestick'] ]
```

Computação intermediária: Pontuação do agrupamento

```
[ 3.2, 1 ] # Computação: [ (4*4)/5, (1*1)/1]
```

Saída: Pontuação da frase

```
3.2 # max([3.2, 1])
```

O trabalho feito em `score_sentences` é apenas uma contabilização para detectar os agrupamentos da frase. Um agrupamento é definido como uma frase contendo duas ou mais palavras importantes no qual cada palavra importante está dentro de um limite de distância de sua vizinha mais próxima. Enquanto o artigo de Luhn sugere um valor de 4 ou 5 para o limite de distância, utilizamos um valor de 3 para simplificar o exemplo; dessa forma, a distância entre 'Green' e 'Colonel' foi suficiente para nossas condições, e o primeiro agrupamento detectado consistiu nas primeiras cinco palavras da frase. Note que se a palavra "study" também estivesse na lista de palavras importantes, toda a frase (com a exceção da pontuação final) teria sido representada como um agrupamento.

Tendo pontuado cada frase, o que falta fazer é determinar quais frases devem ser retornadas como um resumo. A implementação do exemplo fornece duas abordagens possíveis. A primeira utiliza um limite estatístico para filtrar frases computando a média e um desvio padrão para as pontuações obtidas, enquanto a segunda simplesmente retorna as *N* principais frases. Dependendo da natureza dos dados, os resultados podem variar, mas não deve ser difícil ajustar os parâmetros adequadamente para obter os resultados que deseja. Um ponto positivo no uso das *N* principais frases é que, dessa forma, você tem uma boa noção do comprimento máximo que o resumo terá. Utilizando a média e

o desvio padrão, você pode potencialmente retornar mais frases do que deseja, caso muitas delas tenham pontuações relativamente próximas umas das outras.

O algoritmo de Luhn é de simples implementação, e aproveita o fato de que as palavras que ocorrem com frequência geralmente são descritivas do documento como um todo. Entretanto, não se esqueça de que, assim como muitas das abordagens de RI que exploramos no capítulo anterior, o algoritmo de Luhn não procura compreender os dados em um nível semântico mais profundo. Ele simplesmente computa resumos como uma função de palavras que ocorrem com frequência, e não é particularmente sofisticado na forma como pontua frases. Ainda assim, como no caso da TF-IDF, isso faz com que seja ainda mais incrível que o algoritmo possa trabalhar sobre dados randomicamente selecionados a partir de textos de blogs.

Ao pesar os prós e contras da implementação de uma abordagem mais complicada, vale a pena refletir sobre o esforço necessário para aperfeiçoar um resumo já razoável como o produzido pelo algoritmo de Luhn. Às vezes, heurística básica é tudo de que você necessita para alcançar seu objetivo. Em outros casos, entretanto, pode ser necessário algo mais avançado. A parte delicada é computar o custo-benefício da migração de uma heurística simples para uma solução de ponta. Muitas vezes, tendemos a ser exageradamente otimistas quando consideramos o esforço relativo envolvido.

Análise centrada em entidades: um entendimento mais profundo dos dados

Por todo este capítulo, dissemos que abordagens analíticas que exibem um entendimento mais profundo dos dados podem ser muito mais poderosas do que abordagens que simplesmente tratam cada token com um símbolo simples. Mas o que significa exatamente "esse entendimento mais profundo" dos dados? Uma interpretação possível é a de que ele oferece a capacidade de detectar entidades nos documentos e de utilizá-las como base de análise, em oposição a uma análise centrada em documentos, evolvendo pesquisas apenas de palavras-chave ou a interpretação de um termo de pesquisa como um tipo específico de

entidade e a personalização dos resultados de acordo. Mesmo que talvez você ainda não tenha pensado sobre esse assunto nesses termos, é precisamente isso que tecnologias emergentes como o WolframAlpha fazem à camada de apresentação. Por exemplo, uma pesquisa por "tim o'reilly" no WolframAlpha retorna resultados que indicam um entendimento de que a entidade pesquisada é uma pessoa; você não recebe apenas uma lista de documentos contendo palavras-chave (Figura 8.3). Independentemente da técnica interna utilizada para esse propósito, a experiência de usuário resultante é drasticamente mais poderosa, pois os resultados se ajustam a um formato que satisfaz de maneira mais adequada às expectativas do usuário.

Ainda que esteja além do escopo deste capítulo ponderar sobre as diversas possibilidades da análise centrada em entidades, é apropriado que apresentemos uma forma de extrair entidades de um documento, para que depois possamos utilizá-las para diversos propósitos analíticos. Presumindo o fluxo de exemplo do processo de NLP que mostramos antes neste capítulo, uma abordagem que poderíamos adotar seria simplesmente extrair todos os substantivos e locuções substantivas do texto e indexá-los como entidades que ocorrem nos documentos – a suposição mais importante seria a de que substantivos e locuções substantivas (ou algum subconjunto delas, construído cuidadosamente) podem ser qualificados como entidades de interesse. Esta é de fato uma suposição adequada que, como mostra o código de exemplo a seguir, é um bom ponto de partida para procedimentos de análise centrada em entidades. Note que, para resultados atribuídos de acordo com os padrões do Penn Treebank, qualquer tag que inicie com 'NN' é algum tipo de substantivo ou locução substantiva. Uma lista completa das Tags do Penn Treebank está disponível on-line (*http://bulba.sdsu.edu/jeanette/thesis/PennTags.html*).

Figura 8.3 – Resultados de exemplo para uma consulta por "tim o'reilly" com o WolframAlpha.

O exemplo 8.5 analisa tags de classe gramatical aplicadas aos tokens, e identifica substantivos e locuções substantivas como entidades. No linguajar da mineração de dados, encontrar as entidades de um texto é conhecido como processo de *extração de entidades*.

Exemplo 8.5 – Extração de entidades de um texto com o NLTK (blogs_and_nlp__extract_entities.py)

```
# -*- coding: utf-8 -*-
import sys
import nltk
import json
# Carrega a saída de blogs_and_nlp__get_feed.py
BLOG_DATA = sys.argv[1]
blog_data = json.loads(open(BLOG_DATA).read())
for post in blog_data:
    sentences = nltk.tokenize.sent_tokenize(post['content'])
```

```
    tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
    pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]
    # Nivela a lista, uma vez que não estamos utilizando a
estrutura das frases
    # e que sabemos que as frases estarão separadas por uma tupla
    # especial de classe gramatical como ('.', '.')
    pos_tagged_tokens = [token for sent in pos_tagged_tokens for
token in sent]
    all_entity_chunks = []
    previous_pos = None
    current_entity_chunk = []
    for (token, pos) in pos_tagged_tokens:
        if pos == previous_pos and pos.startswith('NN'):
            current_entity_chunk.append(token)
        elif pos.startswith('NN'):
            if current_entity_chunk != []:
                # Note que current_entity_chunk poderá ser uma
duplicata quando for acrescentado,
                # por isso a análise de frequência volta a ser
algo que devemos considerar
                all_entity_chunks.append((''
'.join(current_entity_chunk), pos))
            current_entity_chunk = [token]
        previous_pos = pos
   # Armazena os chunks como um índice para o documento
   # e aproveite para cuidar das frequências...
    post['entities'] = {}
    for c in all_entity_chunks:
        post['entities'][c] = post['entities'].get(c, 0) + 1
    # Por exemplo, poderíamos exibir apenas as entidades que
iniciam com letras maiúsculas
    print post['title']
    print '-' * len(post['title'])
    proper_nouns = []
    for (entity, pos) in post['entities']:
        if entity.istitle():
            print '\t%s (%s)' % (entity, post['entities']
[(entity, pos)])
    print
```

Talvez você se recorde da descrição de "extração", que vimos neste mesmo capítulo, na seção "Um

processo típico de NLP com o NLTK", em que o NLTK fornece uma função nltk.batch_ne_chunk que tenta extrair entidades nomeadas de tokens com classes gramaticais indicadas. Você pode muito bem utilizar essa capacidade diretamente, mas perceberá que os resultados variam para os modelos convencionais que sustentam a implementação. Uma discussão sobre o aperfeiçoamento da implementação que acompanha o NLTK está fora do escopo deste capítulo.

Um exemplo de saída para o código pode ser visto no exemplo 8.6. Ele fornece resultados muito significativos e que seriam ótimas sugestões para tags em uma plataforma de blogging inteligente. Para um corpus maior do que o utilizado nesse exemplo, uma nuvem de tags também poderia ser uma ótima opção para visualização dos dados. Lembre-se de que a seção "Visualização de tweets com nuvens de tags", do capítulo 5, apresentou uma nuvem de tags fácil de ser utilizada, mas muito poderosa, que você pode facilmente adaptar para este caso.

Exemplo 8.6 – Resultados de exemplo do exemplo 8.5

```
The Louvre of the Industrial Age
--------------------------------
    Paris (1)
    Henry Ford Museum (1)
    Vatican Museum (1)
    Museum (1)
    Thomas Edison (2)
    Hermitage (1)
    Uffizi Gallery (1)
    Ford (2)
    Santa Rosa (1)
    Dearborn (1)
    Makerfaire (1)
    Berlin (1)
    Marc (2)
    Makerfaire (1)
    Rome (1)
    Henry Ford (1)
    Ca (1)
    Louvre (1)
    Detroit (2)
    St. Petersburg (1)
    Florence (1)
    Marc Greuther (1)
    Makerfaire Detroit (1)
    Luther Burbank (2)
    Make (1)
    Dale Dougherty (1)
```

```
Louvre (1)
```

Seria possível descobrir a mesma lista de termos utilizando uma análise cega das características léxicas (como o emprego de letras maiúsculas) das frases? Talvez, mas lembre-se de que esta técnica também pode capturar substantivos e locuções substantivas não indicadas por letras maiúsculas. O uso de letras maiúsculas ou minúsculas é de fato uma *característica* importante do texto que pode, em geral, ser explorada com utilidade, mas há outras entidades importantes no texto que utilizam apenas letras iniciais minúsculas (por exemplo, "chief curator", "locomotives" e "lightbulbs").

Ainda que a lista das entidades certamente não represente o significado geral do texto com a mesma eficiência do resumo que computamos antes, identificá-las pode ser extremamente valioso em nossa análise, pois *elas têm significado semântico e não são apenas palavras que ocorrem com frequência*. De fato, a frequência da maioria dos termos mostrados na saída de exemplo é baixa. Mesmo assim, eles são importantes, pois têm um significado fundamentado no texto – mais especificamente, representam pessoas, locais, coisas ou ideias que, em geral, são as informações substantivas dos dados.

No ponto em que estamos, não é difícil imaginar que outro avanço importante seria considerar os verbos, e computar tripletos na forma de sujeito-verbo-objeto para sabermos quais entidades estão interagindo, e qual a natureza dessas interações. Tais tripletos permitiriam a visualização de grafos de objetos para os documentos, que poderiam ser analisados muito mais rapidamente do que se tivéssemos de ler os documentos em si. Melhor ainda: imagine trabalhar com diversos grafos de objetos derivados de um conjunto de documentos e combiná-los para compreender o corpus como um todo. Essa técnica é atualmente uma área de pesquisa muito ativa, com imensa aplicabilidade em praticamente todas as situações que enfrentam problemas de sobrecarga de informação. Ainda assim, como ilustraremos, na maioria dos casos esse é um problema complicadíssimo e pouco indicado aos menos corajosos.

Supondo que o identificador de classe gramatical tenha identificado as classes de uma frase e emitido uma saída como `[('Mr.', 'NNP'), ('Green', 'NNP'), ('killed', 'VBD'), ('Colonel', 'NNP'), ('Mustard', 'NNP'), ...]`, um índice armazenando tuplas na forma de

sujeito-verbo-objeto `('Mr. Green', 'killed', 'Colonel Mustard')` poderia ser facilmente computado. Entretanto, a realidade é que é muito improvável que você encontre dados de classes gramaticais com tal grau de simplicidade – a não ser que esteja planejando minerar livros infantis (o que não chega a ser uma sugestão tão ruim para iniciantes). Por exemplo, considere a identificação emitida pelo NLTK para o primeiro período que verificamos no post impresso neste capítulo, como um trecho específico e realista de dados que desejamos traduzir em um grafo de objetos:

> This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum.

O tripleto mais simples que você pode encontrar nesse período é `('I', 'get', 'tour')`, mas mesmo analisada nesse nível a informação que podemos obter não indicará que Dale Dougherty também participou do passeio, ou que Mark Greuther estava envolvido. Os dados com as tags de classes gramaticais devem deixar claro que não é tão simples formular qualquer uma dessas interpretações, pois o período verificado tem uma estrutura muito rica:

```
[(u'This', 'DT'), (u'morning', 'NN'), (u'I', 'PRP'), (u'had',
'VBD'), (u'the', 'DT'),
(u'chance', 'NN'), (u'to', 'TO'), (u'get', 'VB'), (u'a', 'DT'),
(u'tour', 'NN'),
(u'of', 'IN'), (u'The', 'DT'), (u'Henry', 'NNP'), (u'Ford',
'NNP'), (u'Museum', 'NNP'),
(u'in', 'IN'), (u'Dearborn', 'NNP'), (u',', ','), (u'MI', 'NNP'),
(u',', ','),
(u'along', 'IN'), (u'with', 'IN'), (u'Dale', 'NNP'),
(u'Dougherty', 'NNP'), (u',',
','), (u'creator', 'NN'), (u'of', 'IN'), (u'Make', 'NNP'), (u':',
':'), (u'and', 'CC'),
(u'Makerfaire', 'NNP'), (u',', ','), (u'and', 'CC'), (u'Marc',
'NNP'), (u'Greuther',
'NNP'), (u',', ','), (u'the', 'DT'), (u'chief', 'NN'),
(u'curator', 'NN'), (u'of',
'IN'), (u'the', 'DT'), (u'museum', 'NN'), (u'.', '.')]
```

É incerto se até mesmo uma solução mais avançada seria capaz de emitir tripletos significativos neste caso, dada a natureza complexa do predicado "had a chance to get a tour", e o fato de que os outros participantes

envolvidos na visita estão listados em uma frase que surge apenas no final do período. Estratégias para construção desses tripletos estão fora do escopo deste livro, mas, em teoria, você deve ser capaz de utilizar informações razoavelmente precisas sobre as classes gramaticais em suas tentativas. A dificuldade dessa tarefa não foi apontada para desencorajá-lo, mas antes para oferecer uma visão realista da complexidade do processo de NLP como um todo, de modo que você saiba com o que está se envolvendo quando decide confrontar o problema da computação de tripletos em situações práticas. Isso pode ser muito trabalhoso, mas os resultados valem a pena.

Assim sendo, a boa notícia é que podemos de fato realizar algumas coisas muito interessantes simplesmente verificando as entidades do texto e utilizando-as como base para análise, como demonstramos antes. Você pode facilmente produzir tripletos a partir do texto de cada frase, em que o "predicado" de cada tripleto represente um relacionamento genérico, significando que o sujeito e o objeto interagiram um com o outro. O exemplo 8.7 é uma refatoração do exemplo 8.5, que coleta entidades com base em cada sentença, e que pode ser muito útil para computar interações entre entidades utilizando uma sentença como janela de contexto.

Exemplo 8.7 – Descobrindo interações entre entidades (blogs_and_nlp__extract_interactions.py)

```
# -*- coding: utf-8 -*-
import sys
import nltk
import json
def extract_interactions(txt):
    sentences = nltk.tokenize.sent_tokenize(txt)
    tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
    pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]
    entity_interactions = []
    for sentence in pos_tagged_tokens:
        all_entity_chunks = []
        previous_pos = None
        current_entity_chunk = []
        for (token, pos) in sentence:
            if pos == previous_pos and pos.startswith('NN'):
                current_entity_chunk.append(token)
```

```
            elif pos.startswith('NN'):
                if current_entity_chunk != []:
                    all_entity_chunks.append((' 
'.join(current_entity_chunk), pos))
                current_entity_chunk = [token]
            previous_pos = pos
        if len(all_entity_chunks) > 1:
            entity_interactions.append(all_entity_chunks)
        else:
            entity_interactions.append([])
    assert len(entity_interactions) == len(sentences)
    return dict(entity_interactions=entity_interactions,
sentences=sentences)
if __name__ == '__main__':
    # Lê a saída de blogs_and_nlp__get_feed.py
    BLOG_DATA = sys.argv[1]
    blog_data = json.loads(open(BLOG_DATA).read())
   # Exiba interações selecionadas com base em cada período
    for post in blog_data:
        post.update(extract_interactions(post['content']))
        print post['title']
        print '-' * len(post['title'])
        for interactions in post['entity_interactions']:
            print '; '.join([i[0] for i in interactions])
        print
```

Os resultados deste código, apresentados no exemplo 8.8, destacam algo muito importante sobre a natureza da análise de dados não-estruturados: ela é bem bagunçada!

Exemplo 8.8 – Saída de exemplo do exemplo 8.7

```
The Louvre of the Industrial Age
--------------------------------
morning; chance; tour; Henry Ford Museum; Dearborn; MI; Dale
Dougherty; creator;
Make; Makerfaire; Marc Greuther; chief curator
tweet; Louvre
"; Marc; artifact; museum; block; contains; Luther Burbank;
shovel; Thomas Edison…
Luther Burbank; course; inventor; treasures; nectarine; Santa
```

```
Rosa
Ford; farm boy; industrialist; Thomas Edison; friend
museum; Ford; homage; transformation; world
machines; steam; engines; coal; generators; houses; lathes;
precision; lathes;
makerbot; century; ribbon glass machine; incandescent;
lightbulbs; world; combine;
harvesters; railroad; locomotives; cars; airplanes; gas;
stations; McDonalds;
restaurant; epiphenomena
Marc; eye; transformation; machines; objects; things
advances; engineering; materials; workmanship; design; years
years; visit; Detroit; museum; visit; Paris; Louvre; Rome;
Vatican Museum; Florence;
Uffizi Gallery; St. Petersburg; Hermitage; Berlin
world; museums
Museum; Makerfaire Detroit
reach; Detroit; weekend
day; Makerfaire; day
```

Certa quantidade de ruído nos resultados é praticamente inevitável, mas esforçar-se para obter dados altamente inteligíveis e úteis – mesmo que contenham uma quantidade considerável de ruído – é um objetivo digno. A quantidade de esforço necessário para atingir resultados imaculados, livres de qualquer tipo de ruído, pode ser imensa. De fato, na maioria dos casos, isso é praticamente impossível, em razão da complexidade inerente envolvida em processos de linguagem natural e das limitações da maioria dos kits de ferramentas atualmente disponíveis, como o NLTK. Se você for capaz de formular certas suposições sobre o domínio dos dados, ou tiver conhecimento especializado acerca da natureza do ruído, pode ser capaz de projetar heurísticas que sejam efetivas sem arriscar um volume inaceitável de perda potencial de informações. Mas isso não é nada fácil.

Ainda assim, as interações verificadas no texto fornecem uma representação muito valiosa. Por exemplo, o quão próxima sua interpretação de "morning; chance; tour; Henry Ford Museum; Dearborn; MI; Dale Dougherty; creator; Make; Makerfaire; Marc Greuther; chief curator" estaria do significado da frase original?

Assim como em nossa aventura anterior pela criação de resumos, a

produção de uma marcação que possa ser inspecionada visualmente é algo muito prático. Uma simples modificação à saída do script é o suficiente para produzir o resultado da figura 8.4 (Exemplo 8.9).

Exemplo 8.9 – Modificação do script do exemplo 8.7 (blogs_and_nlp__extract_interactions_markedup_output.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import nltk
import json
from blogs_and_nlp__extract_interactions import
extract_interactions
HTML_TEMPLATE = """<html>
    <head>
        <title>%s</title>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
    </head>
    <body>%s</body>
</html>"""
if __name__ == '__main__':
    # Leia saída de blogs_and_nlp__get_feed.py
    BLOG_DATA = sys.argv[1]
    blog_data = json.loads(open(BLOG_DATA).read())
    # A versão com marcação pode ser escrita no disco
    if not os.path.isdir('out/interactions'):
        os.makedirs('out/interactions')
    for post in blog_data:
        post.update(extract_interactions(post['content']))
        # Exibe a saída como marcação com entidades representadas
em negrito
        post['markup'] = []
        for sentence_idx in range(len(post['sentences'])):
        s = post['sentences'][sentence_idx]
        for (term, _) in post['entity_interactions']
[sentence_idx]:
            s = s.replace(term, '<strong>%s</strong>' % (term, ))
        post['markup'] += [s]
```

```
    filename = post['title'] + '.entity_interactions.html'
    f = open(os.path.join('out', 'interactions', filename), 'w')
    html = HTML_TEMPLATE % (post['title'] + ' Interactions', '
'.join(post['markup']),)
    f.write(html.encode('utf-8'))
    f.close()
    print >> sys.stderr, "Data written to", f.name
```

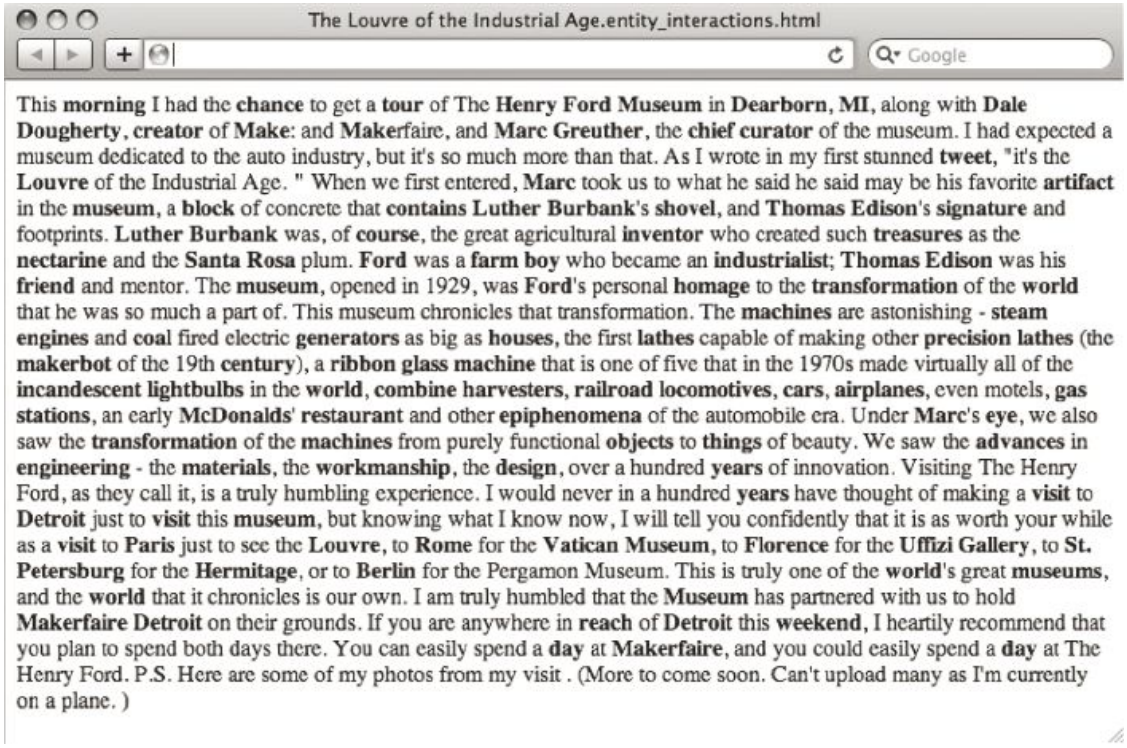


Figura 8.4 – Saída HTML de exemplo que exibe em negrito as entidades identificadas no texto para facilitar a visualização dos conceitos mais importantes do conteúdo.

Em consideração ao "leitor interessado", também é interessante realizar análises adicionais identificando os conjuntos de interações para um corpo maior de texto, para encontrar coocorrências nessas interações. Você também poderia muito facilmente adaptar algumas das saídas do Graphviz ou do Protovis, do capítulo 1, para visualizar um grafo das interações em que as arestas não necessariamente teriam rótulos. O arquivo de exemplo em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/introduction__retweet_visualization.py* seria um ótimo ponto de partida.

Mesmo desconhecendo a natureza específica da interação, ainda é de grande valor simplesmente saber qual o sujeito e o objeto. Da mesma forma, caso você esteja se sentindo ambicioso e não se importe com certo

grau de desorganização ou falta de precisão, nada impede que tente preencher as lacunas deixadas pelos verbos faltantes.

Qualidade das análises

Depois de ter feito muita mineração de dados, você eventualmente desejará quantificar a qualidade de suas análises. Especialmente quando falamos em mineração de textos. Por exemplo, caso você inicie a personalização de um algoritmo básico para extração das entidades de textos não-estruturados, como poderá saber se o desempenho desse algoritmo melhora ou piora, no que diz respeito à qualidade dos resultados? Ainda que seja possível inspecionar manualmente os resultados de um pequeno corpus e ajustar o algoritmo até que você esteja satisfeito, isso seria praticamente impossível em um corpus maior ou em um documento de classe diferente – daí a necessidade de um processo mais automatizado.

Um ponto de partida óbvio é efetuar uma amostragem randômica de alguns documentos, criar um "conjunto principal" de entidades, cuja extração por um algoritmo adequado você acredita ser absolutamente crucial, e utilizar essa lista como base de avaliação. Dependendo do rigor pretendido, você pode até ser capaz de computar o *erro amostral* e utilizar um dispositivo estatístico conhecido como intervalo de confiança[5] para prever um *erro verdadeiro* com um grau suficiente de confiança para suas necessidades.

Entretanto, qual é exatamente o cálculo que você deve computar, com base nos resultados de seu extrator e do conjunto principal, para avaliar a precisão? Um cálculo muito usual para medição de precisão é conhecido como *medida F1*, definida em termos de dois conceitos, *precisão* e *recall*[6], como:

$$F = 2 \cdot \frac{\text{precisão} \cdot \text{recall}}{\text{precisão} + \text{recall}}$$

em que:

$$\text{precisão} = \frac{TP}{TP + FP}$$

e:

$$\text{recall} = \frac{TP}{TP + FN}$$

No atual contexto, precisão é a medida da exatidão que reflete "falsos positivos", e recall é a medida da completude que reflete "verdadeiros positivos". A lista a seguir esclarece o significado dos termos em relação à presente discussão, caso você ainda não esteja familiarizado com eles:

Verdadeiros positivos (True Positives, TP)

Termos que foram corretamente identificados como entidades.

Falsos positivos (False Positives, FP)

Termos que foram identificados como entidades, mas que não deveriam ter sido.

Verdadeiros negativos (True Negatives, TN)

Termos que não foram identificados como entidades e que não deveriam ter sido.

Falsos negativos (False Negatives, FN)

Termos que não foram identificados como entidades, mas que deveriam ter sido.

Como a precisão é uma medida de exatidão que quantifica falsos positivos, ela é definida como *precisão = TP / (TP + FP)*. Intuitivamente, se o número de falsos positivos for zero, a exatidão do algoritmo será perfeita e a precisão produzirá um valor de 1,0. Da mesma forma, se o número de falsos positivos for alto e se aproximar, ou até mesmo ultrapassar, o valor dos verdadeiros positivos, a precisão será pobre e a razão se aproximará de zero. Como medida de completude, o recall é definido como *TP / (TP + FN)* e produz um valor de 1,0, indicando um recall perfeito, caso o número de falsos negativos seja zero. Conforme eleva o número de falsos negativos, o recall se aproxima de zero. Note que, por definição, F1 produzirá um valor de zero quando tanto a precisão quanto o recall forem pobres. Evidentemente, o que você descobrirá, na prática, é que geralmente há um dilema entre aperfeiçoar a precisão ou o recall, pois é muito difícil atender a ambos. Se pensar por um instante, você perceberá que isso faz sentido, considerando as alternativas envolvidas com falsos positivos e falsos negativos (Figura 8.5).

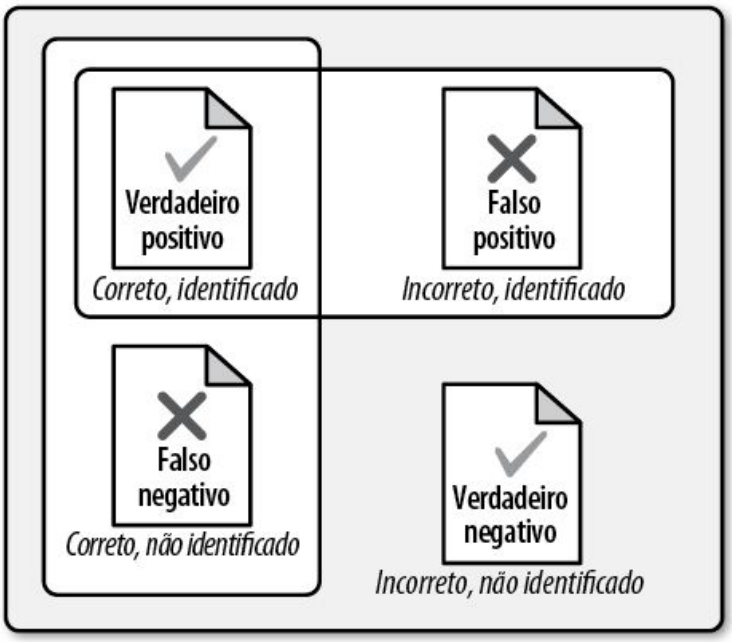


Figura 8.5 – Intuição por trás de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos.

Para colocar tudo que vimos em perspectiva, vamos analisar a já clássica frase "Mr. Green killed Colonel Mustard in the study with the candlestick", supondo que um especialista tenha determinado como suas entidades principais as seguintes: "Mr. Green", "Colonel Mustard", "study" e "candlestick". Supondo também que seu algoritmo tenha identificado esses quatro termos, e apenas eles, você terá 4 verdadeiros positivos, 0 falsos positivos, 5 verdadeiros negativos ("killed", "with", "the", "in", "the") e 0 falsos negativos. Isso representa precisão e recall perfeitos, o que nos dá uma medida F1 de 1,0. A substituição de vários valores nas fórmulas de precisão e recall é um exercício simples, que merece ser feito, caso este seja seu primeiro contato com esses termos. Por exemplo, quais seriam os valores de precisão, recall e F1 se o seu algoritmo tivesse identificado "Mr. Green", "Colonel", "Mustard" e "candlestick"?

Como nota paralela, pode ser interessante saber que muitas das implementações tecnológicas mais atraentes utilizadas empresarialmente no setor de NLP utilizam modelos estatísticos avançados para processar linguagem natural de acordo com algoritmos de aprendizagem supervisionada. Um *algoritmo de aprendizagem supervisionada* é essencialmente uma abordagem na qual você fornece exemplos de treinamento na forma de `[(entrada1, saída1), (entrada2, saída2), ..., (entradaN, saídaN)]` a um modelo, de modo que ele seja capaz de

prever as tuplas com razoável precisão. O complicado é garantir que o modelo treinado faça generalizações adequadas para entradas ainda não encontradas. Caso o modelo tenha bom desempenho com dados de treinamento, mas um resultado inadequado para exemplos desconhecidos, podemos dizer que ele sofre de um problema de *overfitting*[7] dos dados de treinamento. Uma abordagem usual para avaliar a eficácia de um modelo é conhecida como *validação cruzada*. Com ela, uma parte dos dados de treinamento (digamos, um terço) é reservada exclusivamente para testes do modelo, e apenas o restante é utilizado para seu treinamento.

Comentários finais

Este capítulo apresentou o mínimo essencial em análise avançada de dados não-estruturados, e demonstrou como utilizar o NLTK para ir além do parsing de frases que apresentamos no capítulo 7, formulando o restante do processo do NLP e da extração de entidades do texto. O campo da linguística computacional ainda é nascente, e solucionar o problema do NLP para a maioria dos idiomas mais falados é, talvez, o grande desafio do século. Leve o NLTK ao limite, e quando for necessário mais desempenho ou qualidade, não tenha medo de arregaçar as mangas e pesquisar a literatura acadêmica. Esta pode ser uma tarefa desafiadora, mas é também um problema digno, caso você esteja interessado em enfrentá-lo.

Se você está interessado em mais informações sobre o conteúdo deste capítulo, utilize as ferramentas de lematização do NLTK na tentativa de computar tuplas (entidade, predicado derivado, entidade), aprimorando o código do exemplo 8.7. Também pode ser interessante verificar o WordNet (*http://wordnet.princeton.edu/*), ferramenta que você inevitavelmente encontrará mais cedo ou mais tarde, para descobrir significados adicionais referentes aos itens das tuplas. Caso você tenha muito tempo livre, pode consultar algumas das muitas e populares APIs de comentários, como a DISQUS (*http://groups.google.com/group/disqus-dev/web/api-1-1*), e tentar incorporar as técnicas de NLP que abordamos aos fluxos de comentários para posts de blogs. Formular um plug-in WordPress que sugira inteligentemente tags com base nas entidades extraídas de um post de exemplo também pode ser uma ótima atividade para ocupar uma noite ou um fim de semana vazios.

1 Um homônimo é um caso especial de um homógrafo. Duas palavras são homógrafas se têm a mesma grafia. Duas palavras são homônimas se têm a mesma grafia e a mesma pronúncia. Por alguma razão, é mais costumeiro ouvirmos falar de "homônimos" do que de "homógrafos", mesmo que o termo seja utilizado equivocadamente.

2 Consulte também a definição do Urban Dictionary para a expressão "bomb" (*http://www.urbandictionary.com/define.php?term=bomb*).

3 A resolução de pronomes é conhecida como resolução de anáforas (*http://en.wikipedia.org/wiki/Anaphora_(linguistics)*), tópico que não faz parte do escopo deste livro.

4 "Treebank" é um termo muito específico que faz referência a um corpus especialmente identificado com informações linguísticas avançadas. De fato, o motivo de tal corpus ser chamado "treebank" é para enfatizar que se trata de um "banco" de frases (um bank em inglês, semelhante a uma coleção), ao qual foi feito o parsing para transformá-lo em árvores (trees) que aderem a uma gramática específica.

5 Há muitas informações sobre intervalos de confiança (*http://en.wikipedia.org/wiki/Confidence_interval*) disponíveis on-line.

6 Mais precisamente, diz-se que F1 é a média harmônica entre precisão e recall, em que a média harmônica de dois números quaisquer, x e y, é definida como:

$$H = 2 * \frac{x * y}{x + y}$$

Você pode ler mais sobre por que chamamos essa média de "harmônica" consultando a definição de um número harmônico (*http://en.wikipedia.org/wiki/Harmonic_number*).

7 N.T.: em estatística, overfitting ocorre quando um modelo estatístico descreve um erro randômico ou ruído, em vez de um relacionamento inerente. Overfitting geralmente ocorre quando um modelo é excessivamente complexo, com parâmetros demais em relação ao número de observações. Nesse caso, o modelo geralmente terá pobre desempenho preditivo, visto que pode exagerar pequenas flutuações nos dados (fonte: Wikipédia).

CAPÍTULO 9

Facebook: a rede social capaz de tudo

Do ponto de vista das redes sociais, o Facebook é verdadeiramente uma maravilhosa criação capaz de tudo. Considerando que seus mais de 500 milhões de usuários podem atualizar seus status públicos e informar aos seus amigos o que estão fazendo/pensando/etc., trocar mensagens mais longas, de modo semelhante a uma comunicação por e-mail, participar de chats em tempo real, organizar e compartilhar fotos, fazer "check in" em locais físicos, e mais uma dúzia de outras ações por meio do site, não é nenhuma surpresa que o Facebook tenha superado o Google (*http://techcrunch.com/2010/12/29/hitwise-facebook-overtakes-google-to-become-most-visited-website-in-2010/*) como o site mais visitado até o fim de 2010. A figura 9.1 mostra um gráfico que justapõe os números de visitas do Google e do Facebook, caso você ainda não esteja convencido. Esses números são particularmente empolgantes, pois sabemos que onde há um grande número de usuários regulares, sempre há muitos dados interessantes. Neste capítulo, você aproveitará a API incrivelmente poderosa do Facebook para minerar esses dados e descobrir quais são seus amigos mais conectados, agrupá-los de acordo com interesses comuns e obter um rápido indicador que mostre sobre o que as pessoas em sua rede social estão conversando.

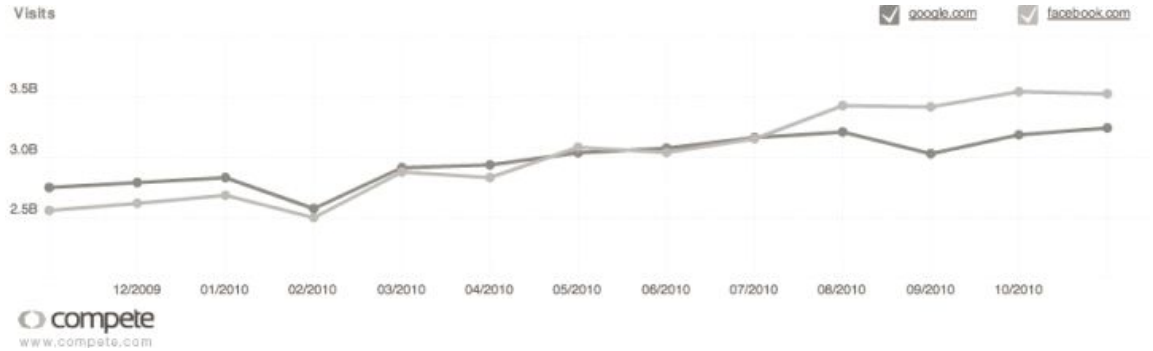


Figura 9.1 – Total de visitas mensais do Facebook e do Google em agosto de 2010.

Iniciaremos com uma breve visão geral das APIs mais comuns do Facebook, e depois faremos uma rápida transição para criar alguns scripts

que aproveitem essas APIs, para que possamos analisar e visualizar alguns dados sociais. Virtualmente todas as técnicas que aplicamos em capítulos anteriores poderão ser aplicadas aos dados do Facebook, pois sua plataforma é muito rica e diversa. Assim como na maioria dos outros capítulos, não seremos capazes de abordar todos os pontos possíveis: um livro inteiro poderia ser escrito se quiséssemos abordar apenas as muitas aplicações interessantes e dispositivos de mineração de dados que você pode construir em uma plataforma rica como essa para acessar todos os detalhes das pessoas mais próximas a você. Assim como ocorre com qualquer outra plataforma rica de desenvolvedor, certifique-se de que as técnicas que você utiliza neste capítulo em aplicações de produção consideram a privacidade dos usuários e lembre-se de revisar regularmente os princípios e políticas de uso do Facebook (*http://developers.facebook.com/policy/*) para evitar surpresas inesperadas quando estiver apresentando sua aplicação (ou aplicativo, como o Facebook as chama no Brasil) ao mundo.

Explorando os dados de sua rede social

Esta seção fornece uma breve visão geral sobre como completar o fluxo OAuth 2.0 do Facebook para uma aplicação desktop, e obter um token de acesso. Depois, mostraremos ainda alguns exercícios de coleta de dados. Para manter o capítulo o mais simples possível, não discutiremos a criação de uma aplicação, ou aplicativo, para o Facebook: há muitos tutoriais on-line que podem ajudá-lo nesse sentido, e apresentar o desenvolvimento de aplicações para o Facebook exigiria uma visão geral sobre uma plataforma de servidor, como a Google App Engine (GAE), uma vez que as aplicações e apps[1] do Facebook devem ser hospedadas externamente em seu próprio ambiente de servidor. Entretanto, uma versão GAE dos scripts apresentados neste capítulo está disponível para download em *http://github.com/ptwobrussell/Mining-the-Social-Web/tree/master/web_code/facebook_gae_demo_app*. É muito fácil implantá-la, e ela representa uma autêntica aplicação Facebook que você poderá usar como ponto de partida assim que decidir converter seus scripts em aplicações funcionais.

De zero ao token de acesso em menos de 10 minutos

Assim como ocorre com qualquer outra aplicação que utilize o OAuth, você terá de obter um ID e um segredo de aplicação que utilizará para autorização. Também terá de aceitar o acesso a uma comunidade de desenvolvedores, e depois criar uma "aplicação". A lista a seguir resume os principais passos, e você também pode estudar algumas dicas visuais na figura 9.2:

- Primeiro, caso você ainda não a tenha, terá de criar uma conta no Facebook. Basta acessar *http://facebook.com* e completar o registro para se juntar à festa.
- Depois, você terá de instalar a aplicação Developer, visitando *http://www.facebook.com/developers* e selecionando as opções da solicitação para instalar a aplicação.
- Assim que a aplicação Developer tiver sido instalada, você poderá clicar no botão "Set Up New Application" para criar sua aplicação.

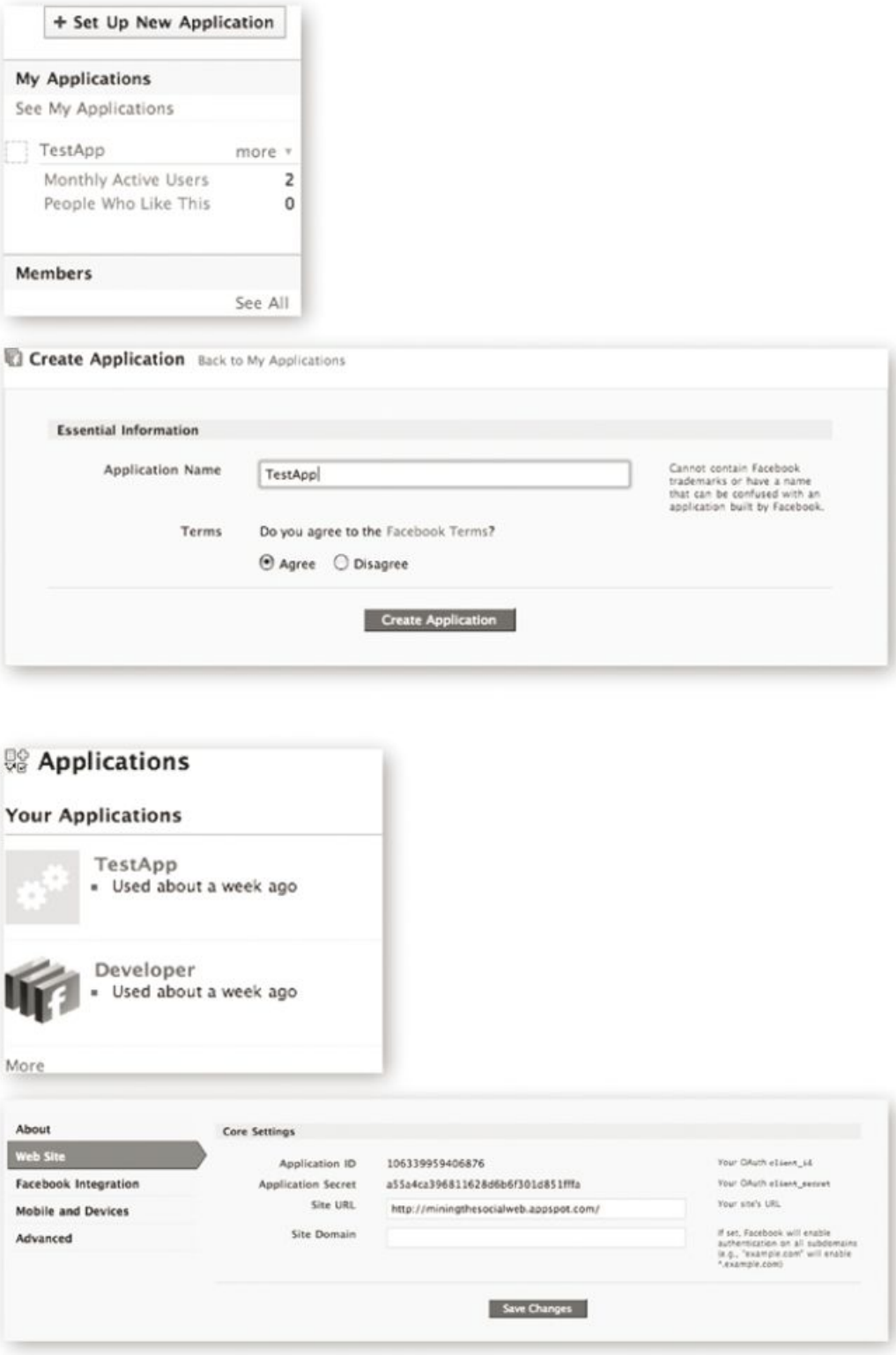


Figura 9.2 – De cima para baixo: a) o botão no qual você clicará em http://www.facebook.com/developers para criar uma nova aplicação, b) a caixa de diálogo que você completará para dar um nome à sua aplicação e aceitar os termos de serviço, c) sua

aplicação agora fará parte da lista de aplicações, e d) as configurações de sua aplicação, incluindo seu ID e segredo OAuth 2.0.

- Assim que você tiver concluído a verificação de segurança, sua aplicação terá um ID e um segredo que poderão ser utilizados para completar os passos envolvidos na implementação OAuth 2.0 do Facebook, e você será apresentado a um formulário que deverá preencher para especificar as configurações do Web Site de sua aplicação. Apenas digite o URL que você planeja utilizar para hospedagem da aplicação no campo Site URL, e inclua o mesmo domínio no Site Domain. O Facebook utiliza suas configurações de Web Site como parte do fluxo do OAuth, e você receberá uma mensagem de erro durante esse processo caso elas não sejam preenchidas corretamente.

Pode não parecer óbvio, mas talvez a forma mais simples de retornar à sua aplicação de desenvolvimento, depois de ter saído dela, é simplesmente retornar a *http://facebook.com/developers* (é necessário um log-in).

Esclarecidos os detalhes básicos de registro, o próximo passo é escrever um script que cuide da autenticação e que lhe consiga um token de acesso que poderá ser utilizado para acessar as APIs. O fluxo geral para o processo é de fato um pouco mais simples do que o visto nos capítulos anteriores envolvendo o Twitter e o LinkedIn. Nosso script abrirá um navegador web, você se conectará à sua conta no Facebook e, depois, ele apresentará um código especial (seu token de acesso) que você pode copiar/colar em um prompt para salvá-lo no disco e utilizá-lo em solicitações futuras. O exemplo 9.1 ilustra o processo e não é nada mais que uma implementação simples do fluxo descrito em "Desktop Application Authentication" (*http://developers.facebook.com/docs/authentication/desktop*). Uma breve revisão da documentação de autenticação do Facebook (*http://developers.facebook.com/docs/authentication/*) pode ser útil; consulte novamente a seção "Não, você não pode saber minha senha", no capítulo 4, caso ainda não o tenha feito. Entretanto, note que o fluxo implementado no exemplo 9.1 para uma aplicação desktop é um pouco mais simples do que o envolvido na autenticação de uma aplicação web.

Exemplo 9.1 – Obtenção de um token de acesso OAuth 2.0 para uma aplicação desktop (facebook__login.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import webbrowser
import urllib
def login():
    # Obtenha este valor nas configurações de sua aplicação do
Facebook
    CLIENT_ID = ''
    REDIRECT_URI = \
      'http://miningthesocialweb.appspot.com/static/facebook_oaut
h_helper.html'
    # Você poderia personalizar quais permissões estendidas estão
sendo solicitadas na página
    # de log-in ou editar a lista a seguir. Por padrão, incluímos
todas as que devem ter acesso
    # de escrita como descritas em
http://developers.facebook.com/docs/authentication/.
    # (Provavelmente seria um absurdo solicitar tantas permissões
    # de acesso se você quisesse lançar uma aplicação de produção
de êxito)
    EXTENDED_PERMS = [
        'user_about_me',
        'friends_about_me',
        'user_activities',
        'friends_activities',
        'user_birthday',
        'friends_birthday',
        'user_education_history',
        'friends_education_history',
        'user_events',
        'friends_events',
        'user_groups',
        'friends_groups',
        'user_hometown',
        'friends_hometown',
        'user_interests',
        'friends_interests',
        'user_likes',
        'friends_likes',
        'user_location',
        'friends_location',
```

```
        'user_notes',
        'friends_notes',
        'user_online_presence',
        'friends_online_presence',
        'user_photo_video_tags',
        'friends_photo_video_tags',
        'user_photos',
        'friends_photos',
        'user_relationships',
        'friends_relationships',
        'user_religion_politics',
        'friends_religion_politics',
        'user_status',
        'friends_status',
        'user_videos',
        'friends_videos',
        'user_website',
        'friends_website',
        'user_work_history',
        'friends_work_history',
        'email',
        'read_friendlists',
        'read_requests',
        'read_stream',
        'user_checkins',
        'friends_checkins',
        ]
    args = dict(client_id=CLIENT_ID, redirect_uri=REDIRECT_URI,
                scope=','.join(EXTENDED_PERMS),
type='user_agent', display='popup')
    webbrowser.open('https://graph.facebook.com/oauth/authorize?'
                    + urllib.urlencode(args))
    # Opcionalmente, armazene seu token de acesso localmente para
uso, quando necessário,
    # em vez de passá-lo como um parâmetro de linha de comando
nos scripts...
    access_token = raw_input('Enter your access_token: ')
    if not os.path.isdir('out'):
        os.mkdir('out')
    filename = os.path.join('out', 'facebook.access_token')
    f = open(filename, 'w')
```

```
    f.write(access_token)
    f.close()
    print >> sys.stderr, \
            "Access token stored to local file:
'out/facebook.access_token'"
    return access_token
if __name__ == '__main__':
    login()
```

Um detalhe importante sobre o qual você pode estar em dúvida é a definição de `EXTENDED_PERMS`, por isso cabe uma breve explicação nesse sentido. Na primeira vez em que você tentar fazer o log-in na aplicação, será notificado de que ela está solicitando muitas *permissões estendidas*, para que você possa ter o máximo de flexibilidade ao acessar os dados disponíveis (Figura 9.3). Os detalhes das permissões estendidas estão descritos na documentação de autenticação do Facebook (*http://developers.facebook.com/docs/authentication/*), mas o resumo é que, por padrão, aplicações podem acessar apenas alguns dados básicos dos perfis dos usuários – como nome, sexo e foto do perfil – e permissões explícitas têm de ser oferecidas para acesso a dados adicionais. *O detalhe que merece atenção aqui é que ainda que você seja capaz de visualizar certos detalhes de seus amigos, como as coisas de que mais "gostam" ou as atividades em seus murais, por meio de sua conta normal do Facebook, sua aplicação não poderá acessar esses mesmos detalhes, a menos que receba permissão explícita para fazê-lo*. Em outras palavras, há diferença entre o que você vê nos perfis de seus amigos quando faz seu log-in no *facebook.com* e os dados que recebe quando solicita informações por meio da API. Isso ocorre, pois é o Facebook (e não você) que emprega a plataforma para obter os dados quando você está logado em *facebook.com*, mas é você (como desenvolvedor) que solicita os dados quando cria sua aplicação.

Caso sua aplicação não solicite permissões estendidas para acessar os dados, mas tente acessá-los da mesma forma, você poderá receber de volta objetos vazios de dados em vez de uma mensagem de erro explícita.

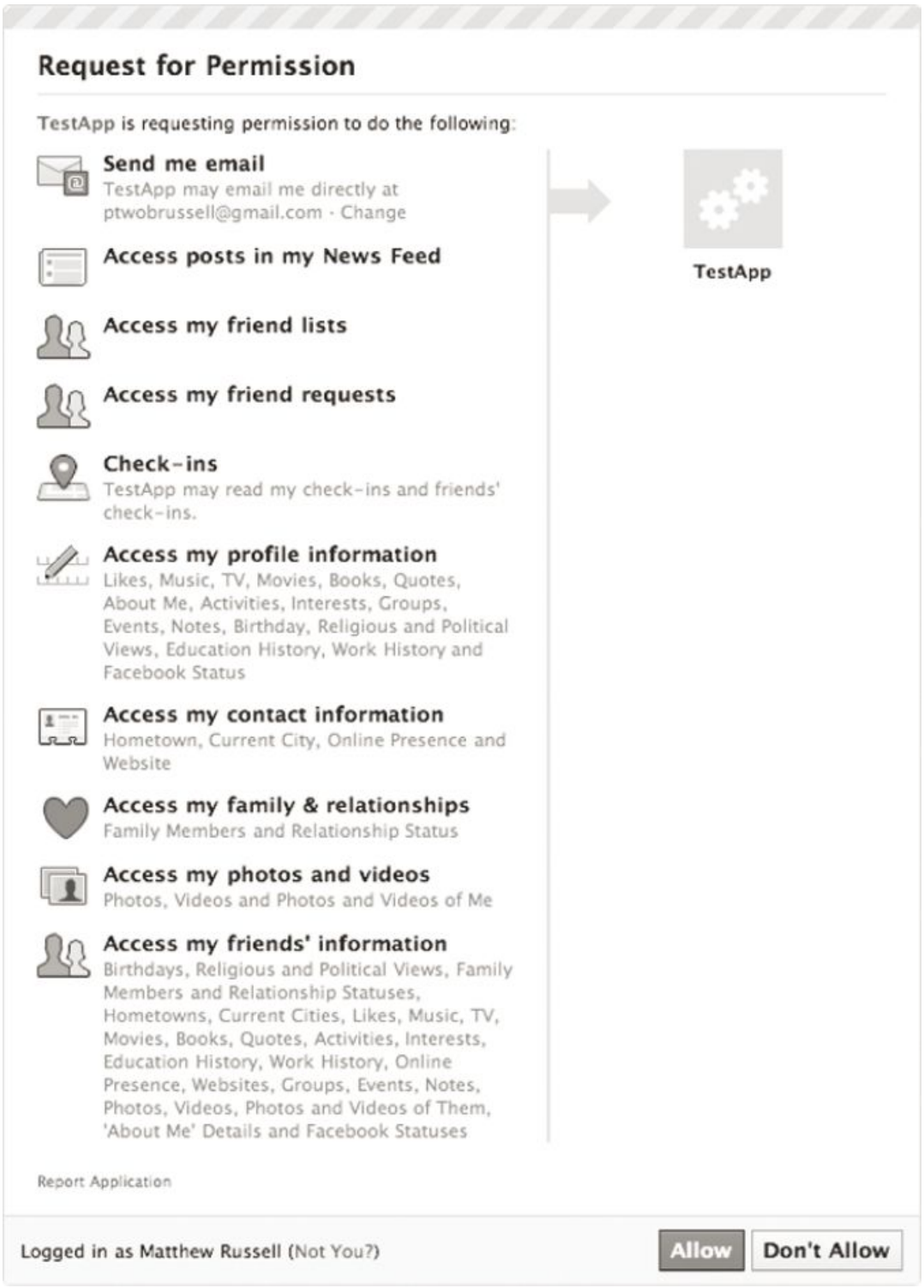


Figura 9.3 – Diálogo de exemplo que você veria caso uma aplicação solicitasse permissão para acessar tudo que estivesse disponível a ela.

A documentação da plataforma do Facebook está em constante evolução, e pode não lhe dizer tudo o que você deseja saber sobre permissões estendidas. Por exemplo, parece que para obter acesso a informações religiosas ou políticas de seus amigos (sua permissão estendida `friends_religion_politics`), essas pessoas devem ter instalado sua aplicação e ainda autorizado

explicitamente o acesso a esses dados, via a permissão estendida `user_religion_politics`.

Agora, munido de um belo novo token de acesso com permissões para acessar todos os seus dados, é o momento de avançar para alguns tópicos mais interessantes.

APIs de consulta do Facebook

Como você já deve saber, o ecossistema de desenvolvedores do Facebook é complexo, está em constante evolução[2], e é recheado de particularidades, envolvendo os controles de privacidade (*http://www.facebook.com/privacy/explanation.php*) mais complexos que podemos encontrar na web. Além de oferecer um conjunto-padrão de APIs REST (*http://developers.facebook.com/docs/reference/rest/*) e uma linguagem de tipo SQL mais avançada para consultas de dados, a Facebook Query Language (*http://developers.facebook.com/docs/reference/fql/*), ou FQL, o Facebook apresentou também a API Graph e o Open Graph Protocol, OGP (*http://opengraphprotocol.org*), em abril de 2010 na conferência F8. Em resumo, o OGP é um mecanismo que permite que você transforme qualquer página web em um objeto de gráfico social, injetando nela metadados RDFa (*http://www.w3.org/TR/xhtml-rdfa-primer/*, falaremos mais sobre isso em breve), enquanto a API Graph é um mecanismo simples e intuitivo que permite consultar esse gráfico. Cada objeto tem um tipo específico. Quando da redação deste livro, a API Graph aceitava os seguintes tipos de objetos, como descritos em sua referência (*http://developers.facebook.com/docs/reference/api/*):

Album
: Um álbum de fotos.

Application
: Uma aplicação individual registrada no Facebook Platform.

Checkin
: Um check-in feito utilizando o Facebook Places.

Event
: Um evento do Facebook.

Group

Um grupo do Facebook.

Link

Um link compartilhado.

Note

Uma nota do Facebook.

Page

Uma página do Facebook.

Photo

Uma foto individual.

Post

Uma entrada individual em um feed de perfil.

Status Message

Uma mensagem de usuário no mural de um usuário.

Subscription

Uma assinatura individual de uma aplicação para obter atualizações em tempo real para um tipo de objeto.

User

Um perfil de usuário.

Video

Um vídeo individual.

A referência da API Graph (*http://developers.facebook.com/docs/reference/api/*) contém documentação detalhada para cada tipo de objeto, descrevendo os tipos de propriedades e conexões que você pode esperar de cada um deles.

O exemplo 9.2 é o caso canônico da documentação que demonstra como transformar a página do IMDB referente ao filme *The Rock* (no Brasil, A Rocha) em um objeto no protocolo Open Graph como parte de um documento XHTML que utiliza namespaces. Esses trechos de metadados têm imenso potencial quando aplicados em grande escala, pois permitem que um URI como *http://www.imdb.com/title/tt0117500* represente de forma inequívoca qualquer página web – seja para uma pessoa, empresa,

produto etc. – de uma forma legível por máquinas, promovendo a visão de uma web semântica.

Exemplo 9.2 – RDFa de exemplo para o Open Graph Protocol

```
<html xmlns:og="http://ogp.me/ns#">
<head>
<title>The Rock (1996)</title>
<meta property="og:title" content="The Rock" />
<meta property="og:type" content="movie" />
<meta property="og:url"
content="http://www.imdb.com/title/tt0117500/" />
<meta property="og:image" content="http://ia.media-
imdb.com/images/rock.jpg" />
...
</head>
...
</html>
```

Ao considerar as possibilidades permitidas pelo OGP, seja previdente e criativo, mas não se esqueça de que este é um recurso novo, ainda em evolução. Considerando a forma como o OGP se relaciona com a web semântica, e com os web standards[3] em geral, há confusão quanto ao uso da expressão "open" em seu nome (*http://techcrunch.com/2010/04/23/facebook-open-graph/*), e muitos pequenos problemas na especificação ainda estão sendo solucionados (*http://groups.google.com/group/open-graph-protocol/browse_thread/thread/cc03368ef0d12c1a*). Lembre-se, todavia, de que estamos falando essencialmente da iniciativa de uma única empresa, e que, no momento, sua implementação mal chega a acompanhar as capacidades dos meta-elementos (*http://en.wikipedia.org/wiki/Meta_element*) de dias mais antigos da Web. De fato, o OGP ainda está longe de ser um padrão, mas o potencial existe para que isso mude, e muitas transformações empolgantes podem acontecer no futuro com a expansão dessa inovação. Retornaremos ao tópico da web semântica no capítulo 10, no qual discutiremos brevemente seu vasto potencial. Agora, vamos nos concentrar em como podemos colocar para funcionar a API Graph, construindo uma simples aplicação para Facebook que permita minerar dados sociais.

Em razão da semelhança de título, é fácil confundir a API Social Graph do Google (*http://socialgraph-resources.googlecode.com/svn/trunk/samples/findyours.html*) com a API Graph do Facebook, mesmo

que elas sejam bem diferentes.

Em seu cerne, a API Graph é incrivelmente simples: basta substituir um ID de um objeto no URI *http(s)://graph.facebook.com/ID* para buscar detalhes sobre esse objeto. Por exemplo, buscar o URL *http://graph.facebook.com/http://www.imdb.com/title/tt0117500* em seu navegador retornaria a resposta do exemplo 9.3.

Exemplo 9.3 – Resposta de exemplo para uma consulta do Open Graph a http://graph.facebook.com/http://www.imdb.com/title/tt0117500

```
{
    "id": "114324145263104",
    "name": "The Rock (1996)",
    "picture": "http://profile.ak.fbcdn.net/hprofile-ak-
snc4/hs344.snc4/41581...jpg",
    "link": "http://www.imdb.com/title/tt0117500/",
    "category": "Movie",
    "description": "Directed by Michael Bay. With Sean Connery,
Nicolas Cage, ...",
    "likes" : 3
}
```

Se você inspecionar o código-fonte para o URL *http://www.imdb.com/title/tt0117500*, verá que os campos na resposta correspondem aos dados nas meta tags da página, e que isso não é nenhuma coincidência. A entrega de metadados ricos em resposta a uma simples consulta é o princípio por trás da forma como o OGP foi projetado para funcionar. Tudo fica mais interessante quando você solicita explicitamente metadados adicionais para um objeto da página, acrescentando o parâmetro de string de consulta `metadata=1` à solicitação. Uma resposta de exemplo para a consulta *https://graph.facebook.com/114324145263104?metadata=1* pode ser vista no exemplo 9.4.

Exemplo 9.4 – Reposta de exemplo para uma consulta Open Graph a http://graph.facebook.com/http://www.imdb.com/title/tt0117500?metadata=1 com os metadados opcionais incluídos

```
{
    "id": "118133258218514",
    "name": "The Rock (1996)",
    "picture": "http://profile.ak.fbcdn.net/hprofile-ak-
snc4/..._s.jpg",
```

```
    "link": "http://www.imdb.com/title/tt0117500",
    "category": "Movie",
    "website": "http://www.imdb.com/title/tt0117500",
    "description": "Directed by Michael Bay. With Sean Connery,
Nicolas Cage, ...",
    "likes": 3,
    "metadata": {
       "connections": {
          "feed":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/fe
ed",
          "posts":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/po
sts",
          "tagged":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/ta
gged",
          "statuses":
"http://graph.facebook.com/http://www.imdb.com/title/...",
          "links":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/li
nks",
          "notes":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/no
tes",
          "photos":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/ph
otos",
          "albums":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/al
bums",
          "events":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/ev
ents",
          "videos":
"http://graph.facebook.com/http://www.imdb.com/title/tt0117500/vi
deos"
        },
        "fields": [
          {
             "name": "id",
             "description": "The Page's ID. Publicly available. A
JSON string."
          },
```

```
            {
               "name": "name",
               "description": "The Page's name. Publicly available.
A JSON string."
            },
            {
               "name": "category",
               "description": "The Page's category. Publicly
available. A JSON string."
            },
            {
               "name": "likes",
               "description": "\\* The number of users who like the
Page..."
            }
         ]
      },
      "type": "page"
}
```

Os itens em `metadata.connections` apontam para outros nós no grafo. Você pode rastreá-los para obter mais informações interessantes. Por exemplo, você poderia seguir o link "photos" para acessar as fotos associadas ao filme, e potencialmente acompanhar links associados às fotos para descobrir quem as postou, ou para visualizar comentários que tenham sido feito sobre elas. Caso ainda não lhe tenha ocorrido, você também é um objeto no grafo. Tente visitar o mesmo prefixo de URL, mas substitua seu próprio ID ou nome de usuário no Facebook como contexto, para ver o resultado. Como você é o centro lógico de sua própria rede social, revisitaremos essa possibilidade no restante deste capítulo. A próxima seção mergulha mais fundo nas consultas da API Graph, e a seção que lhe segue analisa mais atentamente as consultas FQL.

Explorando a API Graph, uma conexão de cada vez

O Facebook oferece um SDK Python oficial para a API Graph (*http://github.com/facebook/python-sdk/*); basta executar `easy_install facebook-python-sdk`, e você estará pronto para iniciar seu trabalho. Há muitos bons exemplos (*http://github.com/facebook/python-sdk/tree/master/examples/*) sobre como você pode utilizar esse módulo para rapidamente completar a "dança" do OAuth e construir uma

aplicação Facebook completa, hospedada em uma plataforma como a Google App Engine. Analisaremos apenas algumas seções mais específicas da classe `GraphAPI` (definidas em *facebook.py*), que serão utilizadas em scripts individuais. Alguns desses métodos podem ser vistos a seguir:

- `get_object(self, id, **args)`

  Exemplo: `get_object("me", metadata=1)`
- `get_objects(self, id, **args)`

  Exemplo: `get_objects(["me", "some_other_id"], metadata=1)`
- `get_connections(self, id, connection_name, **args)`

  Exemplo: `get_connections("me", "friends", metadata=1)`
- `request(self, path, args=None, post_args=None)`

  Exemplo: `request("search", {"q" : "programming", "type" : "group"})`

Diferentemente de outras redes sociais, não parece haver diretrizes claras acerca dos rate limits das APIs do Facebook. Mesmo que a disponibilidade dessas APIs pareça ser bem generosa, você ainda deve projetar cuidadosamente sua aplicação para que utilize a API o mínimo possível e manipule todas as condições de erro, apenas por precaução. No final de 2010, o mais próximo que tínhamos de uma determinação de diretrizes eram as discussões de desenvolvedores nos fóruns (*http://forum.developers.facebook.net/viewtopic.php?pid=221976*).

O argumento de palavra-chave mais usual (e, muitas vezes, o único) que você provavelmente utilizará é `metadata=1`, para receber as conexões associadas a um objeto, além dos detalhes do objeto em si. Veja o exemplo 9.5, que apresenta a classe `GraphAPI` e utiliza seu método `get_objects` para consultar grupos com a expressão "programming". O exemplo transmite uma característica importante sobre o tamanho dos conjuntos de resultados que você pode obter para muitos tipos de solicitações.

Exemplo 9.5 – Consulta no Open Graph por grupos de "programming"
(facebook__graph_query.py)

```
# -*- coding: utf-8 -*-
import sys
import json
import facebook
import urllib2
from facebook__login import login
try:
    ACCESS_TOKEN = open('out/facebook.access_token').read()
```

```
    Q = sys.argv[1]
except IOError, e:
    try:
        # Se você passar o token de acesso da aplicação do
Facebook como um parâmetro
        # de linha de comando, não se esqueça de encapsulá-lo em
aspas simples para que o shell
        # não interprete nenhum de seus caracteres
        ACCESS_TOKEN = sys.argv[1]
        Q = sys.argv[2]
    except:
        print >> sys.stderr, \
            "Could not either find access token in
'facebook.access_token' or parse args."
        ACCESS_TOKEN = login()
        Q = sys.argv[1]
LIMIT = 100
gapi = facebook.GraphAPI(ACCESS_TOKEN)
# Encontre grupos com o termo de consulta em seus nomes
group_ids = []
i = 0
while True:
    results = gapi.request('search', {
        'q': Q,
        'type': 'group',
        'limit': LIMIT,
        'offset': LIMIT * i,
        })
    if not results['data']:
        break
    ids = [group['id'] for group in results['data'] if
group['name'
           ].lower().find('programming') > -1]
    # assim que os grupos não contiverem mais em seus nomes o
termo que você procura, vá embora
    if len(ids) == 0:
        break
    group_ids += ids
    i += 1
if not group_ids:
```

```
    print 'No results'
    sys.exit()
# Obtém detalhes para os grupos
groups = gapi.get_objects(group_ids, metadata=1)
# Conte o número de membros em cada grupo. A documentação da API
da FQL em
# http://developers.facebook.com/docs/reference/fql/group_member
indica que, para
# grupos com mais de 500 membros, você receberá apenas um
subconjunto randômico de até
# 500 membros.
for g in groups:
    group = groups[g]
    conn = urllib2.urlopen(group['metadata']['connections']
['members'])
    try:
        members = json.loads(conn.read())['data']
    finally:
        conn.close()
    print group['name'], len(members)
```

Resultados de exemplo para a consulta de "programming" são apresentados no exemplo 9.6, e não é nenhuma coincidência que o limite dos conjuntos de resultado se aproxime de 500. Como mostra o comentário no código, a documentação da FQL afirma que, quando você consultar a tabela `group_member`, seus resultados serão limitados a 500 itens totais. Infelizmente, a documentação da API Graph ainda está em evolução e, quando da redação deste livro, alertas semelhantes não estavam documentados (com sorte isso será corrigido em breve). Na contagem de membros de grupos, você muitas vezes trabalhará com uma amostra randômica de tamanho razoável. A seção "Visualização de sua rede social inteira", mais adiante neste capítulo, descreve um cenário diferente, em que ocorre um truncamento inesperado dos resultados, e mostra como contornar esse problema emitindo consultas múltiplas.

Exemplo 9.6 – Resultados do exemplo 9.5

```
Graffiti Art Programming 492
C++ Programming 495
Basic Programming 495
Programming 215
C Programming 493
```

```
C programming language 492
Programming 490
ACM Programming Competitors 496
programming 494
COMPUTER PROGRAMMING 494
Programming with Python 494
Game Programming 494
ASLMU Programming 494
Programming 352
Programming 450
Programmation - Programming 480
```

Um exemplo de aplicação web (*http://miningthesocialweb.appspot.com*) que encapsula a maioria do código de exemplo deste capítulo, e utiliza o mesmo padrão básico, está hospedado no GAE, caso você queira experimentá-lo antes de criar código de sua autoria. A figura 9.4 ilustra os resultados de nossa consulta de exemplo para grupos com a expressão "programming". Lembre-se de que você pode instalar e personalizar completamente a aplicação do Facebook que utiliza o GAE, se esta for uma opção melhor do que executar scripts a partir de um console local. Em termos de produtividade, é provavelmente melhor trabalhar com scripts locais e depois acrescentar funcionalidades ao código do GAE quando tudo estiver pronto, para manter um ciclo de desenvolvimento acelerado.

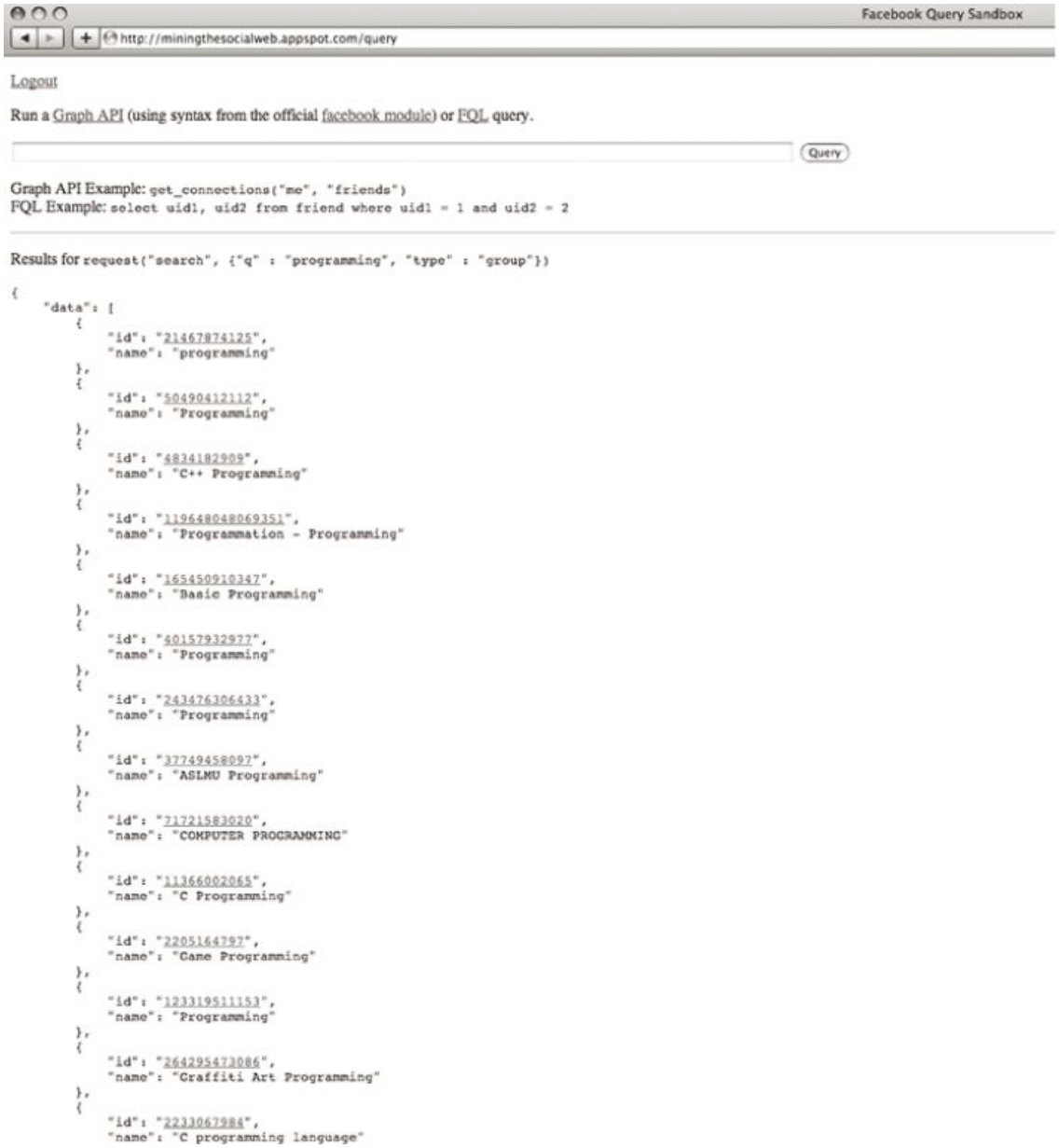


Figura 9.4 – Exemplo de consultas e resultados para grupos de "programming "– você pode clicar nos links para rastrear até o próximo nível do grafo.

Decompondo e analisando dados com a FQL

Como você aprendeu na seção anterior, não há muito overhead envolvido na criação de rotinas simples para interagir com a API Graph, pois objetos no gráfico são simples e você recebe URLs que pode utilizar para acompanhar as conexões desses objetos. Todavia, para tipos mais avançados de consultas, ou certos fluxos de trabalho específicos, você pode descobrir que a FQL é mais adequada. Extensa documentação sobre esse tópico pode ser encontrada on-line

(*http://developers.facebook.com/docs/reference/fql/*). Não entrarei aqui em muitos detalhes, uma vez que a documentação on-line é completa e evolui constantemente, acompanhando a plataforma, mas os pontos essenciais são previsíveis se você está familiarizado com sintaxe SQL básica. Consultas FQL têm a forma `select [fields] from [table] where [conditions]`, mas várias restrições são aplicadas e impedem que a FQL seja mais do que apenas um pequeno subgrupo cuidadosamente selecionado da SQL. Por exemplo, apenas um nome de tabela pode ser utilizado na cláusula `from`, e as condições que podem ser utilizadas na cláusula `where` são limitadas (mas geralmente adequadas) e devem ser marcadas como campos indexados na documentação FQL. Em termos gerais, para executar consultas FQL na plataforma do Facebook, basta enviar suas consultas para um dos dois endpoints da API: *https://api.facebook.com/method/fql.query* ou *https://api.facebook.com/method/fql.multiquery*. A diferença entre eles será discutida a seguir.

O exemplo 9.7 ilustra uma simples consulta FQL que busca nome, sexo e status de relacionamento dos amigos do usuário atualmente conectado.

Exemplo 9.7 – Consulta FQL aninhada que relaciona usuário e dados de conexão

```
select name, sex, relationship_status from user where uid in
    (select target_id from connection where source_id = me() and
target_type = 'user')
```

Essa *consulta aninhada* funciona executando primeiro a subconsulta:

```
select target_id from connection where source_id = me() and
target_type = 'user'
```

que produz uma lista de valores de ID de usuário. A diretriz especial `me()` é um atalho conveniente que corresponde ao ID do usuário atualmente conectado, e a tabela de conexão é projetada para permitir consultas em que você procura os amigos desse usuário. Note que, enquanto a maioria das conexões armazenadas na tabela de conexão (*http://developers.facebook.com/docs/reference/fql/connection*) ocorre entre usuários, outros tipos de conexão podem existir entre usuários e outros tipos de objetos, como `'page'`, por isso a presença de um filtro `target_type` é importante. A consulta externa é depois avaliada, o que resulta na consulta:

```
select name, sex, relationship_status from user where uid in (
```

```
... )
```

da tabela user (*http://developers.facebook.com/docs/reference/fql/user*). A tabela FQL user (*http://developers.facebook.com/docs/reference/fql/user*) oferece muitas informações que podem permitir diversos tipos interessantes de análises referentes a seus amigos. Não deixe de conferi-la.

A forma geral do conjunto de resultados desta consulta FQL pode ser vista no exemplo 9.8.

Exemplo 9.8 – Exemplos de resultados da consulta FQL

```
[
    {
        "name": "Matthew Russell",
        "relationship_status": "Married",
        "sex": "male"
      },
      ...
]
```

Uma *consulta múltipla* FQL funciona essencialmente da mesma forma, exceto que nela você pode executar diversas consultas e referenciar os resultados como nomes de tabela utilizando o símbolo de cerquilha (#). O exemplo 9.9 é uma consulta múltipla FQL equivalente à consulta aninhada mostrada antes.

Exemplo 9.9 – Consulta múltipla FQL que relaciona usuário e dados de conexões

```
{
    "name_sex_relationships" : "select name, sex,
relationship_status from user \
        where uid in (select target_id from #ids)",
    "ids" : "select target_id from connection where source_id =
me() \
        and target_type = 'user'"
}
```

Note que, enquanto a lista individual de objetos é retornada da consulta aninhada, os resultados de ambos os componentes da consulta múltipla FQL são retornados, como demonstra o exemplo 9.10.

Exemplo 9.10 – Exemplo de resultados da consulta múltipla FQL.

```
[
    {
        "fql_result_set": [
```

```
            {
                "target_id": -1
            },
            ...
        ],
        "name": "ids"
    },
    {
        "fql_result_set": [
            {
                "name": "Matthew Russell",
                "relationship_status": "Married",
                "sex": "male"
            },
            ...
        ],
        "name" : "name_sex_relationships"
    }
    ]
```

Programaticamente, a lógica da consulta é bem simples e pode ser encapsulada em uma pequena classe. O exemplo 9.11 demonstra uma classe FQL que pode aceitar uma consulta da linha de comando e executá-la. Veja alguns exemplos de consultas que você poderia executar:

```
$ python facebook__fql_query.py 'select name, sex,
relationship_status
from user where uid in (select target_id from connection
where source_id = me())'
```

```
$ python facebook__fql_query.py '{"name_sex_relationships" :
"select name,
sex, relationship_status from user where uid in (select target_id
from #ids)",
"ids" : "select target_id from connection where source_id =
me()"}'
```

Exemplo 9.11 – Encapsulando consultas FQL com uma pequena abstração de classe Python (facebook__fql_query.py)

```
# -*- coding: utf-8 -*-
```

```
import sys
from urllib import urlencode
import json
import urllib2
```

```
from facebook__login import login
class FQL(object):
    ENDPOINT = 'https://api.facebook.com/method/'
    def __init__(self, access_token=None):
        self.access_token = access_token
    def _fetch(cls, url, params=None):
        conn = urllib2.urlopen(url, data=urlencode(params))
        try:
            return json.loads(conn.read())
        finally:
            conn.close()
    def query(self, q):
        if q.strip().startswith('{'):
            return self.multiquery(q)
        else:
            params = dict(query=q,
access_token=self.access_token, format='json')
            url = self.ENDPOINT + 'fql.query'
            return self._fetch(url, params=params)
        def multiquery(self, q):
            params = dict(queries=q,
access_token=self.access_token, format='json')
            url = self.ENDPOINT + 'fql.multiquery'
            return self._fetch(url, params=params)
# Utilização de exemplo...
if __name__ == '__main__':
    try:
        ACCESS_TOKEN = open('out/facebook.access_token').read()
        Q = sys.argv[1]
    except IOError, e:
        try:
            # Se você passar o token de acesso da aplicação
Facebook como um parâmetro de linha
            # de comando, não se esqueça de encapsulá-lo em aspas
simples para que o shell não
            # interprete nenhum de seus caracteres. Você também
pode ter de escapar o caractere #
            ACCESS_TOKEN = sys.argv[1]
            Q = sys.argv[2]
        except IndexError, e:
            print >> sys.stderr, \
```

```
                "Could not either find access token in
'facebook.access_token'
                    or parse args."
            ACCESS_TOKEN = login()
            Q = sys.argv[1]
    fql = FQL(access_token=ACCESS_TOKEN)
    result = fql.query(Q)
    print json.dumps(result, indent=4)
```

A aplicação GAE de exemplo (*http://miningthesocialweb.appspot.com/*), fornecida como parte do código-fonte deste capítulo, detecta e executa consultas FQL, assim como consultas da API Graph, e pode ser utilizada como um ambiente de testes para que você experimente a FQL. Tendo fornecido uma infraestrutura básica para a execução de consultas, a próxima seção analisará alguns casos de uso para a construção de visualizações e widgets da UI (*user interface*, ou interface do usuário) centrados em dados.

Visualização de dados do Facebook

Esta seção estabelece alguns templates e apresenta alguns bons pontos de partida para análise e visualização de dados do Facebook. Caso você esteja acompanhando o texto, agora tem as ferramentas básicas de que necessita para acessar o que a plataforma tem a oferecer. Os tipos de exercícios que exploraremos nesta seção incluem:

- visualização das amizades mútuas em sua rede social;
- visualização das amizades mútuas dentro de grupos específicos, e de acordo com determinados critérios, como sexo dos usuários;
- construção de um simples jogo com base em dados que desafie você a identificar amigos de acordo com local de nascimento e local em que vivem agora.

A lista de possibilidades pode se estender ainda mais e, mantendo intacta a estrutura que criamos, você não terá dificuldades para personalizar os scripts que escreveremos para solucionar muitos outros problemas.

Visualização de sua rede social inteira

Esta seção demonstra o processo em que buscamos informações de amizades de sua conta no Facebook e visualizamos esses dados de formas

interessantes e úteis, dando preferência ao uso do JavaScript InfoVis Toolkit (*http://thejit.org/*), ou JIT. O JIT oferece alguns ótimos templates de exemplo que podem ser facilmente personalizados para dar origem a uma visualização interativa.

Visualização com gráficos radiais

Um RGraph (*http://en.wikipedia.org/wiki/Radial_tree*)[4] é uma visualização de rede que organiza a exibição, dispondo os nós em círculos concêntricos. RGraphs estão disponíveis em muitas bibliotecas de visualização, incluindo o JIT. Vale a pena fazer uma pausa e explorar os exemplos de RGraphs do JIT para se familiarizar com as vastas possibilidades oferecidas.

A árvore Node-Link do Protovis (*http://vis.stanford.edu/protovis/ex/tree.html*), que apresentamos na seção "Agrupamentos hierárquicos e das k-médias", no capítulo 6, é essencialmente a mesma visualização, mas vale a pena ter também o JIT disponível, pois ele oferece exemplos convencionais mais interativos que podem ser mais fáceis de utilizar. Ter muitas opções nunca é um mau negócio.

Como você já deve saber, a maior parte do esforço envolvido em buscar dados brutos do Facebook e visualizá-los está na transformação necessária para deixar esses dados em um formato que possa ser consumido pela visualização. Um dos formatos de dados aceitos pelo RGraph e por outras visualizações de gráficos do JIT é uma estrutura previsível JSON que consiste em uma lista de objetos, cada um representando um nó e suas adjacências. O exemplo 9.12 transmite o conceito geral para que você tenha uma boa noção de nosso objetivo final. Esta estrutura mostra que "Matthew" está conectado a três outros nós identificados por valores de ID, como definidos por `adjacencies`. O campo `data` fornece informações adicionais sobre "Matthew" e, neste caso, transmite um rótulo humanamente legível referente às suas três conexões, acompanhado do que parece ser uma pontuação normalizada de popularidade.

Exemplo 9.12 – Exemplo de entrada que pode ser consumida pela visualização RGraph do JIT

```
[
    {
        "adjacencies": [
            "2",
            "3",
            "4",
            ...
```

```
        ],
    "data": {
        "connections": "Mark<br>Luke<br>John",
        "normalized_popularity": 0.0079575596817,
        "sex" : "male"
    },
    "id": "1",
    "name": "Matthew"
},
...
]
```

Vamos computar as *amizades mútuas* que existem dentro de sua rede de amigos e visualizar esses dados como um gráfico radial. Em outras palavras, computaremos todas as amizades que existem dentro de sua rede de amigos, o que deve nos fornecer uma boa noção de quem são as pessoas mais populares de sua rede, assim como as que não estão bem conectadas. Há muito valor potencial nesses dados. Em um nível abstrato, temos de executar apenas poucas consultas FQL para coletar os dados: uma consulta para calcular os IDs de seus amigos, outra para conectar seus amigos no gráfico e uma consulta final para capturar detalhes pertinentes que desejamos adicionar ao gráfico, como nomes, datas de nascimento etc. As consultas a seguir transmitem uma noção do que queremos:

Pegue os IDs dos amigos

```
q = "select target_id from connection where source_id = me() and
target_type = 'user'"
my_friends = [str(t['target_id']) for t in fql.query(q)]
```

Calcule amizades mútuas

```
q = "select uid1, uid2 from friend where uid1 in (%s) and uid2 in
(%s)" %
        (",".join(my_friends), ",".join(my_friends),)
mutual_friendships = fql(q)
```

Capture detalhes adicionais para adicionar ao gráfico

```
q = "select uid, first_name, last_name, sex from user where uid
in (%s)" %
        (",".join(my_friends),)
names = dict([(unicode(u["uid"]), u["first_name"] + " " +
              u["last_name"][0] + ".") for u in fql(q)])
```

Esta é a essência, mas há outro detalhe não tão bem documentado que você pode descobrir: se você passar um número mais razoável de valores de ID no segundo passo, quando estiver computando as amizades mútuas, verá que recebe de volta resultados truncados. Isso não é de todo injustificável, pois você está solicitando à plataforma do Facebook que execute, em termos lógicos, uma operação $O(n^2)$ em seu nome (loop aninhado que compara seus amigos uns aos outros), mas o fato de nem sequer receber dados como resposta, em vez de uma mensagem de erro sugerindo que passe menos dados, pode surpreendê-lo.

Assim como tudo no mundo, a plataforma do Facebook não é perfeita. Dê crédito ao que merece elogios e registre bugs quando necessário.

Felizmente, há uma solução para essa situação: basta dividir em lotes os dados utilizando diversas consultas e depois agregar todas elas. Em código, isso teria a seguinte apresentação:

```
mutual_friendships = []
N = 50
for i in range(len(my_friends)/N +1):
    q = "select uid1, uid2 from friend where uid1 in (%s) and
uid2 in (%s)" %
        (",".join(my_friends), ",".join(my_friends[i*N:
(i+1)*N]),)
    mutual_friendships += fql(query=q)
```

Fora essa dificuldade potencial, o restante da lógica envolvida na produção da saída esperada para visualizar o gráfico radial é bem simples e segue o padrão que discutimos. O exemplo 9.13 demonstra um exemplo funcional, que reúne tudo que vimos, e produz saída JavaScript adicional que pode ser utilizada para outras análises.

Exemplo 9.13 – Coleta e transformação dos dados de amigos para a visualização em gráfico radial do JIT (facebook__get_friends_rgraph.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import json
import webbrowser
import shutil
from facebook__fql_query import FQL
from facebook__login import login
HTML_TEMPLATE = '../web_code/jit/rgraph/rgraph.html'
```

```
OUT = os.path.basename(HTML_TEMPLATE)
try:
    ACCESS_TOKEN = open('out/facebook.access_token').read()
except IOError, e:
    try:
        # Se você passar o token de acesso da aplicação Facebook
como um parâmetro de linha
        # de comando, não se esqueça de encapsulá-lo em aspas
simples para que o shell não
        # interprete nenhum de seus caracteres. Você também pode
ter de escapar o caractere #
        ACCESS_TOKEN = sys.argv[1]
    except IndexError, e:
        print >> sys.stderr, \
            "Could not either find access token in
'facebook.access_token' or parse args."
        ACCESS_TOKEN = login()
fql = FQL(ACCESS_TOKEN)
# pega os ids dos amigos
q = \
    'select target_id from connection where source_id = me() and
target_type =\'user\''
my_friends = [str(t['target_id']) for t in fql.query(q)]
# agora, encontra as amizades entre seus amigos. Note que esta
api parece retornar
# resultados arbitrariamente truncados, caso você passe mais do
que algumas centenas de amigos
# em cada parte da consulta. Por isso, realizamos (número de
amigos)/N consultas e agregamos
# os resultados para tentar obter resultados completos
# Aviso: isso pode resultar em várias chamadas à API e em muitos
dados retornados que
# você terá de processar
mutual_friendships = []
N = 50
for i in range(len(my_friends) / N + 1):
    q = 'select uid1, uid2 from friend where uid1 in (%s) and
uid2 in (%s)' \
        % (','.join(my_friends), ','.join(my_friends[i * N:(i +
1) * N]))
    mutual_friendships += fql.query(q)
```

```
# acessa detalhes para seus amigos, como primeiro e último nomes,
e cria um mapa acessível
# note que nem todos os ids necessariamente fornecerão
informações, por isso esteja preparado
# para manipular esses casos futuramente
q = 'select uid, first_name, last_name, sex from user where uid
in (%s)' \
    % (','.join(my_friends), )
results = fql.query(q)
names = dict([(unicode(u['uid']), u['first_name'] + ' ' +
u['last_name'][0] + '.'
             ) for u in results])
sexes = dict([(unicode(u['uid']), u['sex']) for u in results])
# consolida um mapa de informações de conexão sobre seus amigos.
friendships = {}
for f in mutual_friendships:
    (uid1, uid2) = (unicode(f['uid1']), unicode(f['uid2']))
    try:
        name1 = names[uid1]
    except KeyError, e:
        name1 = 'Unknown'
    try:
        name2 = names[uid2]
    except KeyError, e:
        name2 = 'Unknown'
    if friendships.has_key(uid1):
        if uid2 not in friendships[uid1]['friends']:
            friendships[uid1]['friends'].append(uid2)
    else:
        friendships[uid1] = {'name': name1, 'sex':
sexes.get(uid1, ''),
                             'friends': [uid2]}
    if friendships.has_key(uid2):
        if uid1 not in friendships[uid2]['friends']:
            friendships[uid2]['friends'].append(uid1)
    else:
        friendships[uid2] = {'name': name2, 'sex':
sexes.get(uid2, ''),
                             'friends': [uid1]}
# Emite saída JIT para consumo pela visualização
jit_output = []
```

```
for fid in friendships:
    friendship = friendships[fid]
    adjacencies = friendship['friends']
    connections = '<br>'.join([names.get(a, 'Unknown') for a in
adjacencies])
    normalized_popularity = 1.0 * len(adjacencies) /
len(friendships)
    sex = friendship['sex']
    jit_output.append({
      'id': fid,
      'name': friendship['name'],
      'data': {'connections': connections,
'normalized_popularity'
               : normalized_popularity, 'sex': sex},
      'adjacencies': adjacencies,
      })
# Encapsule a saída em uma declaração de variável e armazene em
# um arquivo de nome facebook.rgraph.js, para consumo do
rgraph.html
if not os.path.isdir('out'):
    os.mkdir('out')
# HTML_TEMPLATE faz referência a algumas dependências que temos
de copiar em out/
shutil.rmtree('out/jit', ignore_errors=True)
shutil.copytree('../web_code/jit', 'out/jit')
html = open(HTML_TEMPLATE).read() % (json.dumps(jit_output),)
f = open(os.path.join(os.getcwd(), 'out', 'jit', 'rgraph', OUT),
'w')
f.write(html)
f.close()
print >> sys.stderr, 'Data file written to: %s' % f.name
# Escreve outro arquivo JSON-padrão para análise adicional
# e uso potencial futuro (por facebook_sunburst.py, por exemplo)
json_f = open(os.path.join('out', 'facebook.friends.json'), 'w')
json_f.write(json.dumps(jit_output, indent=4))
json_f.close()
print 'Data file written to: %s' % json_f.name
# Abre a página web em seu navegador
webbrowser.open('file://' + f.name)
```

Se você tem uma rede de amigos muito grande, pode ser necessário filtrar os amigos que visualiza de

acordo com um critério útil que facilite o manuseio dos resultados. Veja o exemplo 9.16 para um exemplo de filtragem por grupo que consulta interativamente o usuário para obter um input.

Se tudo ocorrer como previsto, você terá em mãos uma visualização interativa de toda sua rede de social. A figura 9.5 ilustra o visual desse resultado para uma grande rede social. A próxima seção apresenta algumas técnicas para simplificar a saída em algo mais manejável.

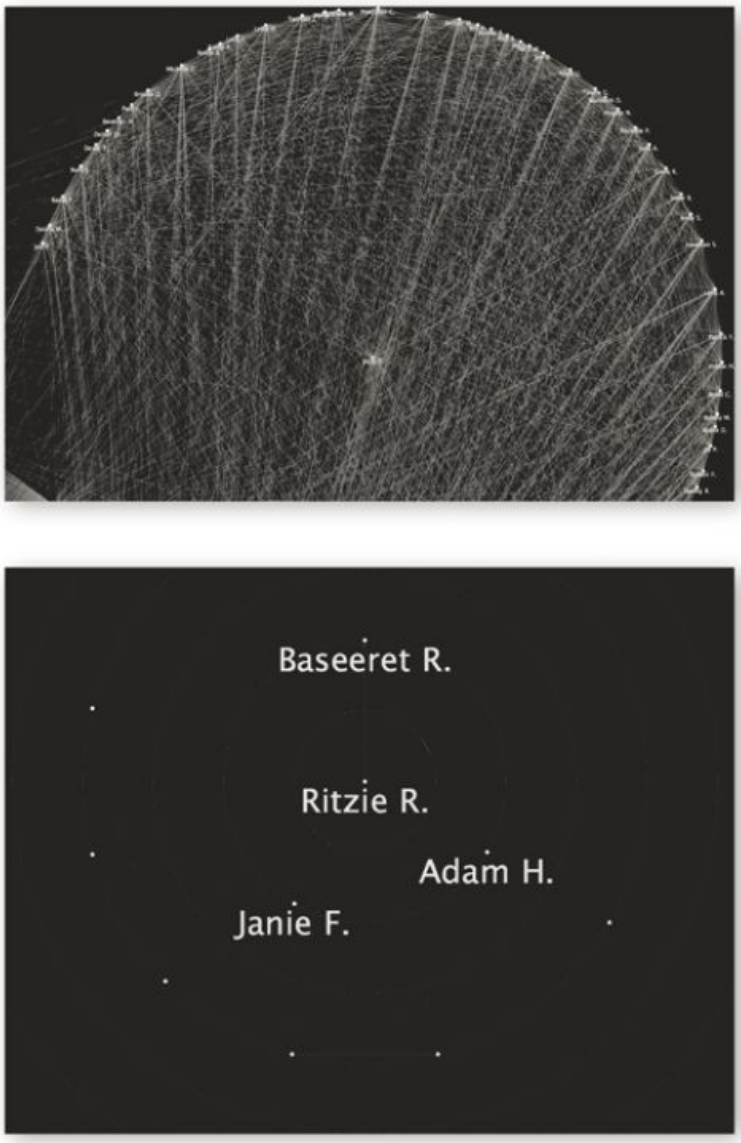


Figura 9.5 – Exemplo de gráfico radial computado com dados do Facebook para uma rede de tamanho considerável, com mais de 500 pessoas, de acordo com o exemplo 9.13 (imagem superior), e um gráfico radial muito menor de amigos mútuos, membros de um grupo específico, como computados pelo exemplo 9.16, mais adiante neste capítulo (imagem inferior).

Visualização com um Sunburst

Uma visualização em Sunburst (*http://www.cc.gatech.edu/gvu/ii/sunburst/*) é um tipo de visualização de preenchimento de espaço para renderização de estruturas hierárquicas como árvores. Ela recebe esse nome em razão de sua semelhança com imagens do Sol e de seus raios, como a que temos na bandeira do Exército Imperial Japonês (Figura 9.6). A implementação Sunburst do JIT (*http://thejit.org/static/v20/Jit/Examples/Sunburst/example2.html*) é tão poderosa quanto bela. Assim como um gráfico radial, ela consome uma simples estrutura JSON de tipo-grafo e expõe manipuladores de eventos muito práticos. Mas enquanto uma visualização em Sunburst consome essencialmente a mesma estrutura de dados de um gráfico radial, seu layout oferece insight adicional de forma praticamente imediata. Você pode facilmente visualizar os graus relativos de nós imediatos da árvore com base na área ocupada por cada camada no Sunburst. Por exemplo, não é difícil adaptar a implementação do exemplo do JIT para renderizar uma visualização interativa e útil de uma rede social, mesmo de tamanho considerável, acompanhada da fatia relativa da população que atende a determinado critério, como o sexo dos usuários. De fato, isso é exatamente o que vemos na figura 9.6. Ela mostra que cerca de dois terços dos membros dessa rede social são mulheres, representando cada um desses membros como adjacentes a essa porção do setor.

O exemplo 9.14 demonstra como tomar a saída de exemplo do exemplo 9.13 e transformá-la de modo que possa ser consumida por um Sunburst JIT, que segmente sua rede social por sexo. Note que o parâmetro `$angularWidth` é uma medida relativa do ângulo utilizado por cada nível da árvore, e é utilizado pelo script para escalonar o setor (*http://en.wikipedia.org/wiki/Circular_sector*). A área que cada amigo consome é escalonada de acordo com sua popularidade e fornece um indicador visual intuitivo. Amigos que têm configurações de privacidade posicionadas para impedir que a API acesse suas informações de sexo terão valores para esse dado como uma string vazia e serão simplesmente ignorados[5]. A versão interativa da visualização também exibe uma tool tip que mostra os amigos mútuos de uma pessoa sempre que você passa o ponteiro do mouse sobre uma fatia (Figura 9.6), para que você possa identificar as pessoas e ser mais específico.

Exemplo 9.14 – Coleta e transformação de dados para a visualização Sunburst do JIT (facebook__sunburst.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import json
import webbrowser
import shutil
from copy import deepcopy
HTML_TEMPLATE = '../web_code/jit/sunburst/sunburst.html'
OUT = os.path.basename(HTML_TEMPLATE)
# Reutiliza out/facebook.friends.json escrito por
facebook__get_friends_rgraph.py
DATA = sys.argv[1]
data = json.loads(open(DATA).read())
# Define as cores para utilizar na visualização por estética
colors = ['#FF0000', '#00FF00', '#0000FF']
# O output primário para coletar input
jit_output = {
    'id': 'friends',
    'name': 'friends',
    'data': {'$type': 'none'},
    'children': [],
    }
# Um template de conveniência
template = {
    'id': 'friends',
    'name': 'friends',
    'data': {'connections': '', '$angularWidth': 1, '$color':
''},
    'children': [],
    }
i = 0
for g in ['male', 'female']:
    # Cria um objeto para o sexo dos amigos
    go = deepcopy(template)
    go['id'] += '/' + g
    go['name'] += '/' + g
    go['data']['$color'] = colors[i]
```

```
    # Encontra amigos de acordo com sexo
    friends_by_gender = [f for f in data if f['data']['sex'] ==
g]
    for f in friends_by_gender:
        # Carrega amigos no objeto criado
        fo = deepcopy(template)
        fo['id'] = f['id']
        fo['name'] = f['name']
        fo['data']['$color'] = colors[i % 3]
        fo['data']['$angularWidth'] = len(f['adjacencies']) #
Classifique por popularidade global
        fo['data']['connections'] = f['data']['connections'] #
Para a tooltip
        go['children'].append(fo)
    jit_output['children'].append(go)
    i += 1
# Emite a saída esperada pelo Sunburst do JIT
if not os.path.isdir('out'):
    os.mkdir('out')
# HTML_TEMPLATE faz referência a algumas dependências que temos
de copiar em out/
shutil.rmtree('out/jit', ignore_errors=True)
shutil.copytree('../web_code/jit', 'out/jit')
html = open(HTML_TEMPLATE).read() % (json.dumps(jit_output),)
f = open(os.path.join(os.getcwd(), 'out', 'jit', 'sunburst',
OUT), 'w')
f.write(html)
f.close()
print 'Data file written to: %s' % f.name
# Abra a página web em seu navegador
webbrowser.open('file://' + f.name)
```

Ainda que a implementação de exemplo apenas agrupe amigos por sexo e popularidade, você poderia produzir um objeto JSON consumível que fosse representativo de uma árvore com níveis múltiplos, em que cada nível correspondesse a um critério diferente. Por exemplo, você poderia visualizar primeiro por sexo, e depois por status de relacionamento, para obter uma noção intuitiva sobre como essas duas variáveis se correlacionam em sua rede social.

Visualização com planilhas (como nos velhos tempos)

Ainda que você possa estar imaginando criar muitas visualizações avançadas, com ótimo apelo visual quando exibidas no navegador, é bem mais provável que você queira visualizar seus dados como nos velhos tempos: em uma planilha. Presumindo que você tenha salvado a saída JSON do gráfico radial da seção anterior em um arquivo local, pode muito facilmente produzir um simples formato CSV, que poderá ser carregado em uma planilha de modo que você perceba rapidamente a essência da apresentação de suas amizades – por exemplo, como um histograma mostrando a popularidade de cada um dos amigos em sua rede social. O exemplo 9.15 demonstra como transformar rapidamente os dados em um formato CSV que possa ser prontamente consumido.

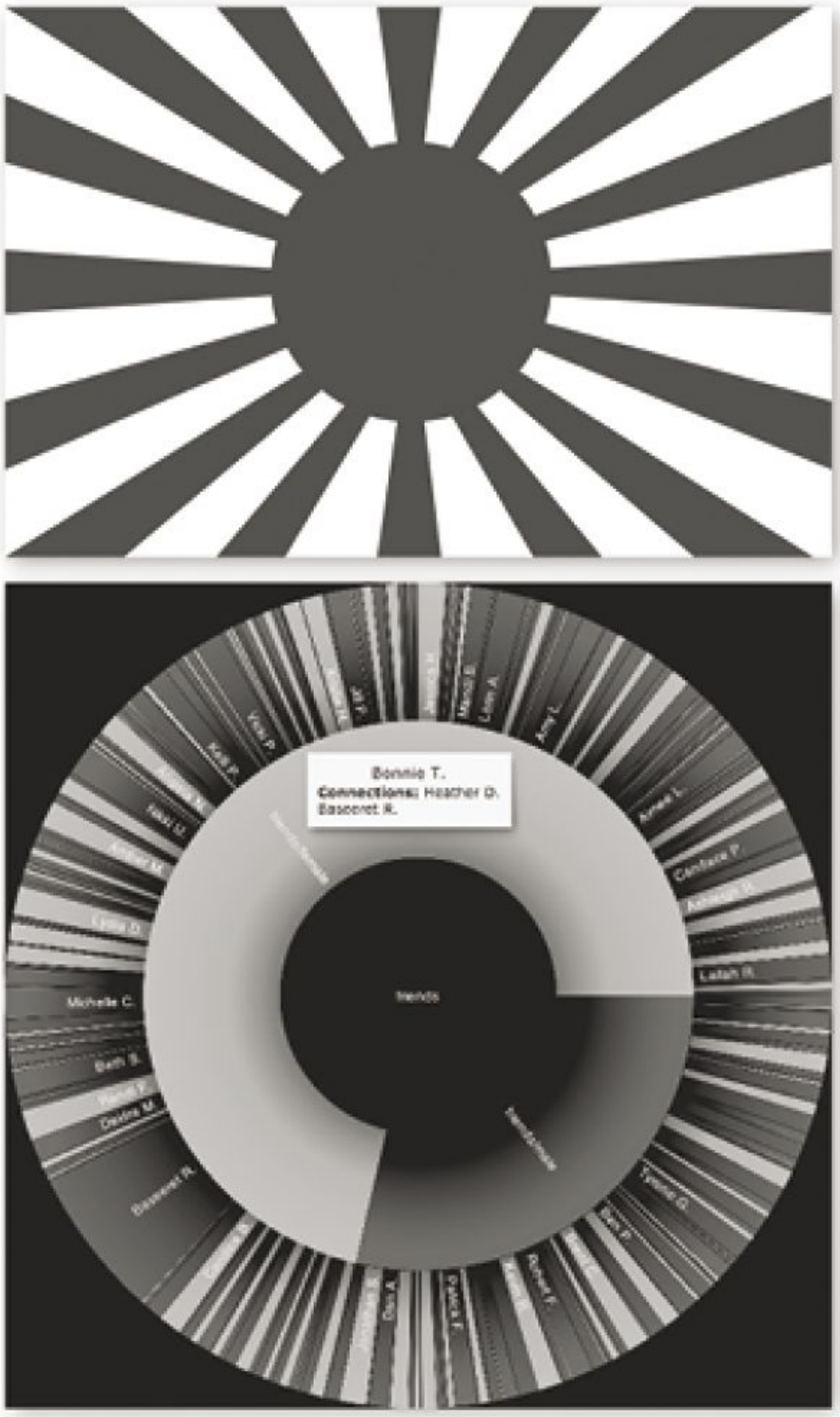


Figura 9.6 – Visualizações Sunburst recebem esse nome em razão da semelhança óbvia com símbolos do Sol, como o que temos na bandeira do Exército Imperial Japonês (imagem

superior). Esta visualização específica demonstra que cerca de 2/3 dos amigos na rede são mulheres e cerca de 1/3, homens (imagem inferior).

Exemplo 9.15 – Exportação de dados para que possam ser facilmente carregados em uma planilha para análise (facebook__popularity_spreadsheet.py)

```
# -*- coding: utf-8 -*-

import os
import sys
import json
import operator

# Reutiliza out/facebook.friends.json escrito por
facebook__get_friends_rgraph.py
DATA = open(sys.argv[1]).read()
data = json.loads(DATA)

popularity_data = [(f['name'], len(f['adjacencies'])) for f in
data]
popularity_data = sorted(popularity_data,
key=operator.itemgetter(1))

csv_data = []
for d in popularity_data:
    csv_data.append('%s\t%s' % (d[0], d[1]))

if not os.path.isdir('out'):
    os.mkdir('out')

filename = os.path.join('out', 'facebook.spreadsheet.csv')
f = open(filename, 'w')
f.write('\n'.join(csv_data))
f.close()

print 'Data file written to: %s' % filename
```

A visualização dos dados como um histograma produz resultados interessantes, pois fornece uma imagem rápida de quão bem conectadas estão as pessoas em sua rede. Por exemplo, se a rede estivesse perfeitamente conectada, a distribuição seria plana. Na figura 9.7 você pode ver que a segunda pessoa mais popular na rede tem cerca de metade das conexões da mais popular, e que o restante dos relacionamentos acompanha aproximadamente uma distribuição de tipo-Zipf[6], com uma longa cauda que se aproxima muito de uma linha de tendência logarítmica.

Uma distribuição logarítmica não é de todo inesperada para uma rede social de tamanho considerável e diversificada. É inevitável que haja

alguns indivíduos muito conectados, enquanto a maioria das pessoas tem relativamente poucas conexões, em comparação com indivíduos mais populares. Na linha do princípio de Pareto, ou do princípio 80-20 (*http://en.wikipedia.org/wiki/Pareto_principle*), podemos dizer neste caso que "20% das pessoas têm 80% dos amigos".

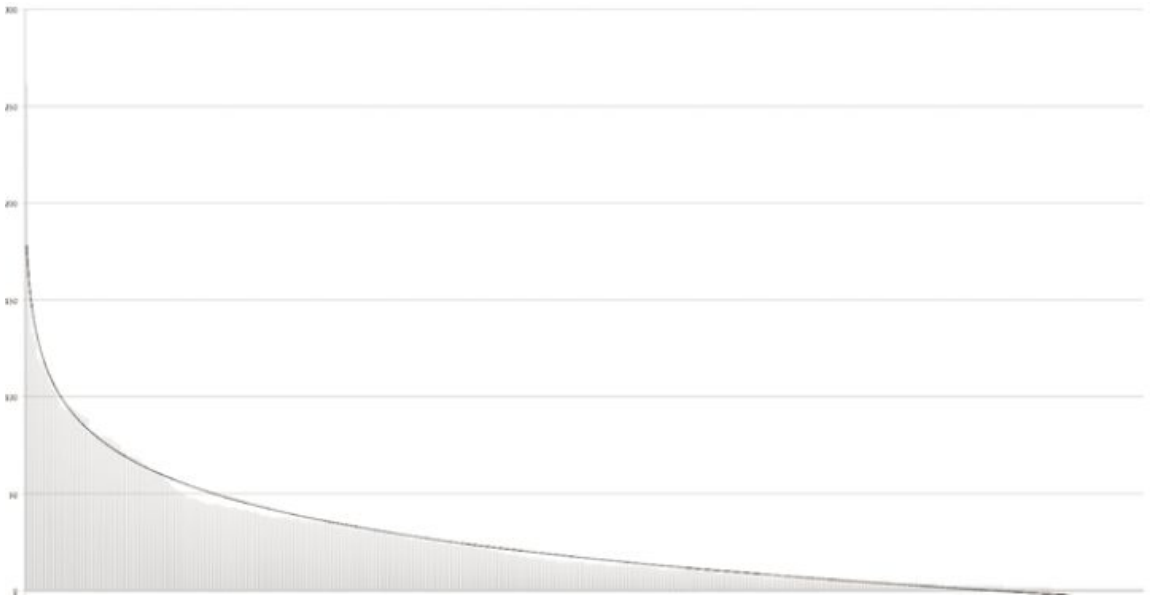


Figura 9.7 – Distribuição de exemplo da popularidade em uma rede de amigos do Facebook, com uma linha de tendência logarítmica (nomes foram omitidos por privacidade).

Visualização de amizades mútuas dentro de grupos

A implementação fornecida no exemplo 9.13 procura visualizar todas as amizades mútuas em sua rede social. Caso você esteja executando essa implementação em um navegador moderno, com suporte adequado ao elemento `canvas`, o JIT apresentará um desempenho surpreendentemente adequado, até para gráficos imensos. Um manipulador de cliques está incluído na implementação de exemplo: ele exibe as conexões do nó selecionado e, sob o gráfico, uma pontuação normalizada de popularidade (calculada como o número de conexões dividido pelo número total de conexões em sua rede). Visualizar sua rede inteira de amigos é realmente interessante, mas, mais cedo ou mais tarde, você desejará filtrar seus amigos de acordo com alguns critérios significativos ou analisar gráficos menores.

Há muitas ótimas opções de filtragem disponíveis, caso você pretenda construir uma boa interface de usuário ao redor de uma pequena aplicação web, mas adotaremos a abordagem mais simples e filtraremos a

saída JSON produzida no exemplo 9.13 de acordo com um critério de grupo especificável. Grupos são o ponto de partida mais lógico, e o exemplo 9.16 demonstra como implementar essa funcionalidade adicional. Uma vantagem de filtrar a saída de sua rede global de amizades é o fato de que você conservará os dados completos de "`connections`" para o restante da rede, por isso continuará capaz de monitorar a quais outros amigos uma pessoa está conectada, mesmo que esses amigos não façam parte do grupo. Lembre-se de que as possibilidades para análise adicional de grupos são vastas: você poderia analisar se seus amigos homens estão mais conectados do que suas amigas mulheres, efetuar análise de cliques, como as descritas em "Detecção e análise de cliques", no capítulo 4, ou ainda muitas outras possibilidades diferentes.

Exemplo 9.16 – Coleta e transformação de dados para visualizar amigos mútuos com um grupo específico (facebook__filter_rgraph_output_by_group.py)

```
# -*- coding: utf-8 -*-

import os
import sys
import json
import facebook
import webbrowser
import shutil
from facebook__fql_query import FQL
from facebook__login import login

HTML_TEMPLATE = '../web_code/jit/rgraph/rgraph.html'
OUT = os.path.basename(HTML_TEMPLATE)

# Reutiliza out/facebook.friends.json escrito por
facebook__get_friends_rgraph.py

DATA = sys.argv[1]
rgraph = json.loads(open(DATA).read())

try:
    ACCESS_TOKEN = open('out/facebook.access_token').read()
except IOError, e:
    try:

        # Se você passar o token de acesso da aplicação Facebook
como um parâmetro de linha de
        # comando, não se esqueça de encapsulá-lo em aspas
simples para que o shell não
        # interprete nenhum de seus caracteres. Você também pode
ter de escapar o caractere #
```

```
        ACCESS_TOKEN = sys.argv[2]
    except IndexError, e:
        print >> sys.stderr, \
            "Could not either find access token in
'facebook.access_token' or parse args."
        ACCESS_TOKEN = login()
gapi = facebook.GraphAPI(ACCESS_TOKEN)
groups = gapi.get_connections('me', 'groups')
# Exibe grupos e consulta o usuário
for i in range(len(groups['data'])):
    print '%s) %s' % (i, groups['data'][i]['name'])
choice = int(raw_input('Pick a group, any group: '))
gid = groups['data'][choice]['id']
# Encontre os amigos no grupo
fql = FQL(ACCESS_TOKEN)
q = \
    """select uid from group_member where gid = %s and uid in
(select target_id from connection where source_id = me() and
target_type = 'user')
""" \
    % (gid, )
uids = [u['uid'] for u in fql.query(q)]
# Filtra a saída gerada previamente para esses ids
filtered_rgraph = [n for n in rgraph if n['id'] in uids]
# Reduz listas de adjacência para qualquer pessoa que não esteja
presente no gráfico.
# Note que os dados completos de conexão exibidos como marcação
HTML em
# #connectionsc ainda estão preservados para o gráfico global.
for n in filtered_rgraph:
    n['adjacencies'] = [a for a in n['adjacencies'] if a in uids]
if not os.path.isdir('out'):
    os.mkdir('out')
# HTML_TEMPLATE faz referência a algumas dependências que temos
de copiar em out/
shutil.rmtree('out/jit', ignore_errors=True)
shutil.copytree('../web_code/jit', 'out/jit')
html = open(HTML_TEMPLATE).read() %
(json.dumps(filtered_rgraph),)
```

```
f = open(os.path.join(os.getcwd(), 'out', 'jit', 'rgraph', OUT),
'w')
f.write(html)
f.close()
print 'Data file written to: %s' % f.name
# Abra a página web em seu navegador
webbrowser.open('file://' + f.name)
```

Um gráfico de exemplo para um grupo que frequentou a mesma escolha primária pode ser visto na figura 9.5.

Certamente, seria interessante comparar a conectividade de tipos diferentes de grupos, identificar quais amigos participam do maior número de grupos aos quais você também pertence etc. Apenas não dedique todo seu tempo a uma única tarefa; as possibilidades são vastas.

Caso esteja utilizando um navegador moderno que ofereça suporte ao mais recente elemento canvas, você pode adaptar o código JavaScript a seguir para salvar seu gráfico como uma imagem:

```
//pegue o element canvas
var canvas = document.getElementById("infovis-canvas");
//agora solicite um download de arquivo
window.location = canvas.toDataURL("image/png");
```

Para onde foram todos meus amigos? (um jogo orientado por dados)

Há muitas variáveis interessantes que você pode correlacionar para análise, mas todos gostamos de um pouco de diversão de vez em quando. Esta seção constrói a base para um simples jogo que você pode utilizar para verificar se conhece bem seus amigos, agrupando-os de modo que suas cidades natais e seus atuais endereços fiquem justapostos. Reutilizaremos o widget de árvore da seção "Agrupamentos inteligentes permitem experiências de usuário atrativas", do capítulo 6, como parte deste exercício, e a maioria do esforço será direcionada para a transformação dos dados obtidos de uma consulta FQL em um formato adequado. Como esse trabalho braçal não é muito interessante, economizaremos espaço exibindo a consulta FQL já em seu formato final. Como sempre, o código-fonte completo está disponível on-line em *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/python_code/linkedin__get_friends_current_locations_and_hometowns.py*.

A consulta FQL que executaremos para acessar nomes, localizações atuais e cidades natais é simples e deve ter visual semelhante ao das consultas FQL feitas anteriormente:

```
q = """select name, current_location, hometown_location from user
where uid in
       (select target_id from connection where source_id =
me())"""
results = fql(query=q)
```

O exemplo 9.17 mostra o formato final que alimenta o widget de árvore, depois que você tiver dedicado o tempo necessário para preparar os dados no formato adequado, e o exemplo 9.18 mostra o código Python para gerá-lo.

Exemplo 9.17 – JSON pretendido que deve ser produzido para consumo pelo widget de árvore Dojo

```
{
    "items": [
        {
            "name": " Alabama (2)",
            "children": [
                {
                    "state": " Alabama",
                    "children": [
                        {
                            "state": " Tennessee",
                            "name": "Nashville, Tennessee (1)",
                            "children": [
                                {
                                    "name": "Joe B."
                                }
                            ]
                        }
                    ],
                    "name": "Prattville, Alabama (1)",
                    "num_from_hometown": 1
                }
            ]
        },
        {
            "name": " Alberta (1)",
        "children": [
            {
```

```
                "state": " Alberta",
                "children": [
                    {
                        "state": " Alberta",
                        "name": "Edmonton, Alberta (1)",
                        "children": [
                            {
                                "name": "Gina F."
                            }
                        ]
                    }
                ],
                "name": "Edmonton, Alberta (1)",
                "num_from_hometown": 1
            }
        ]
    },
    ...
],
"label": "name"
}
```

O widget final acaba tendo o visual da figura 9.8, um display hierárquico que agrupa seus amigos primeiro de acordo com sua localização atual, e depois por suas cidades natais. Na figura 9.8, Jess C. atualmente vive em Tuscaloosa, AL, mas cresceu em Princeton, WV. Ainda que neste caso estejamos correlacionando duas variáveis simples, este exercício ajuda a determinar rapidamente onde se encontra a maioria de seus amigos e permite que você compreenda quem migrou de sua cidade natal e quem não mudou de endereço. Não é difícil imaginar variações mais interessantes, ou displays facetados que apresentem variáveis adicionais, como a universidade frequentada, a afiliação profissional, ou o estado civil dessas pessoas.

Basta uma simples consulta FQL para buscar os dados essenciais, mas há um pouco de trabalho envolvido em reunir itens de dados para preencher o widget de árvore hierárquica. O código-fonte para o widget de árvore em si é idêntico ao da seção "Agrupamentos inteligentes permitem experiências de usuário atrativas", do capítulo 6, e tudo de que necessitamos para visualizar os dados é capturá-los em um arquivo e apontar o widget de árvore para eles. Uma melhoria divertida à

experiência do usuário poderia ser integrar o Google Maps com o widget, de modo que você pudesse rapidamente consultar endereços desconhecidos em um mapa. Adicionar informações quanto à idade e ao sexo dos usuários nesse display também poderia ser interessante se você quisesse se aprofundar nos dados, ou adotar outra abordagem para os agrupamentos. Emitir dados como KML de forma semelhante ao que vimos em "Agrupamento geográfico de sua rede", no capítulo 6, e visualizá-los no Google Earth, poderia ser outra possibilidade válida, dependendo de seu objetivo.

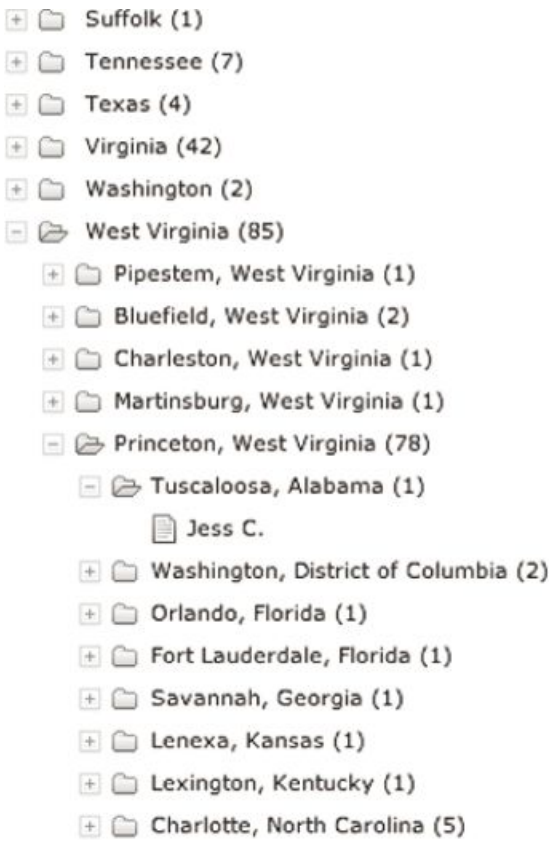


Figura 9.8 – Será que você conhece bem seus amigos? Descubra se divertindo com o grande sucesso "Onde eles estão agora?", um jogo tão informativo quanto divertido!

Exemplo 9.18 – Coletando dados e computando o JSON pretendido como exibido no exemplo 9.17 (facebook__get_friends_current_locations_and_hometowns.py)

```
import sys
import json
import facebook
from facebook__fql_query import FQL
from facebook__login import login

try:
```

```
    ACCESS_TOKEN = open("facebook.access_token").read()
except IOError, e:
    try:
        # Se você passar o token de acesso da aplicação Facebook
como um parâmetro de linha de
        # comando, não se esqueça de encapsulá-lo em aspas
simples para que o shell não
        # interprete nenhum de seus caracteres. Você também pode
ter de escapar o caractere #
        ACCESS_TOKEN = sys.argv[1]
    except IndexError, e:
        print >> sys.stderr, "Could not either find access token"
+ \
            in 'facebook.access_token' or parse args. Logging
in..."
        ACCESS_TOKEN = login()
# Processa os resultados da seguinte consulta FQL, criando saída
JSON adequada para
# consumo por um simples widget de árvore hierárquica:
fql = FQL(ACCESS_TOKEN)
q = \
    """select name, current_location, hometown_location from user
where uid in
        (select target_id from connection where source_id = me()
and target_type =
            'user')"""
results = fql.query(q)
# Primeiro, lê a consulta FQL e cria dois mapas hierárquicos que
agrupam
# as pessoas de acordo com o local em que vivem agora e com suas
cidades natais. Em nosso caso,
# simplesmente tabularemos frequências, mas você poderia
facilmente pegar dados adicionais
# na consulta FQL e utilizá-los para muitas situações criativas.
current_by_hometown = {}
for r in results:
    if r['current_location'] != None:
        current_location = r['current_location']['city'] + ', ' \
            + r['current_location']['state']
    else:
        current_location = 'Unknown'
    if r['hometown_location'] != None:
```

```
        hometown_location = r['hometown_location']['city'] + ', '
\
            + r['hometown_location']['state']
    else:
        hometown_location = 'Unknown'
    if current_by_hometown.has_key(hometown_location):
        if
current_by_hometown[hometown_location].has_key(current_location):
            current_by_hometown[hometown_location]
[current_location] += \
                [r['name']]
        else:
            current_by_hometown[hometown_location]
[current_location] = \
                [r['name']]
    else:
        current_by_hometown[hometown_location] = {}
        current_by_hometown[hometown_location][current_location]
= \
            [r['name']]
# Há muitas formas diferentes de analisar os dados que estão
agora
# em uma estrutura de dados razoável. Vamos criar uma estrutura
# hierárquica adequada à exibição como uma árvore.
items = []
for hometown in current_by_hometown:
    num_from_hometown = sum([len(current_by_hometown[hometown]
[current])
                             for current in
current_by_hometown[hometown]])
    name = '%s (%s)' % (hometown, num_from_hometown)
    try:
        hometown_state = hometown.split(',')[1]
    except IndexError:
        hometown_state = hometown
    item = {'name': name, 'state': hometown_state,
            'num_from_hometown': num_from_hometown}
    item['children'] = []
    for current in current_by_hometown[hometown]:
        try:
            current_state = current.split(',')[1]
        except IndexError:
```

```
            current_state = current
        item['children'].append({'name': '%s (%s)' % (current,
                                len(current_by_hometown[hometown]
[current])),
                                'state': current_state,
'children'
                                : [{'name': f[:f.find(' ') + 2] +
'.'}
                                for f in
                                current_by_hometown[hometown]
[current]]})
        # Classifica os itens alfabeticamente por estado. Uma
ordenação adicional por estado
        # poderia ser feita aqui, se desejado.
    item['children'] = sorted(item['children'], key=lambda i:
i['state'])
    items.append(item)
# Opcionalmente, arrola os itens de níveis externos por estado
para criar uma melhor
# experiência do usuário na exibição. Como alternativa, você
poderia simplesmente passar o valor
# atual dos itens na instrução final que cria a saída JSON para
conjuntos menores de dados.
items = sorted(items, key=lambda i: i['state'])
all_items_by_state = []
grouped_items = []
current_state = items[0]['state']
num_from_state = items[0]['num_from_hometown']
for item in items:
    if item['state'] == current_state:
        num_from_state += item['num_from_hometown']
        grouped_items.append(item)
    else:
        all_items_by_state.append({'name': '%s (%s)' %
(current_state,
                num_from_state), 'children': grouped_items})
        current_state = item['state']
        num_from_state = item['num_from_hometown']
        grouped_items = [item]
all_items_by_state.append({'name': '%s (%s)' % (current_state,
                           num_from_state), 'children':
grouped_items})
```

```
# Finalmente, emite uma saída adequada para consumo por um widget
de árvore hierárquica
print json.dumps({'items': all_items_by_state, 'label': 'name'},
indent=4)
```

Visualização de dados do mural como uma nuvem (giratória) de tags

Assim como ocorre com qualquer outra fonte de dados não-estruturados, analisar a linguagem utilizada em seu mural ou em seu feed de notícias pode ser uma proposta interessante. Há diversos widgets de nuvens de tags que você pode encontrar na web, e todos aceitam a mesma entrada – essencialmente, uma distribuição de frequência. Mas por que visualizar dados com uma nuvem de tags regular quando você poderia utilizar uma nuvem personalizada e interativa? Lembre-se de que existe um popular recurso de código aberto que oferece uma nuvem de tags giratória, o WP-Cumulus (*http://code.google.com/p/word-cumulus-goog-vis/wiki/UserGuide*)[7], capaz de produzir um belo resultado. Para colocá-lo em prática, basta produzir o simples formato de entrada esperado. Por brevidade, não repetiremos os passos necessários aqui. Consulte *http://github.com/ptwobrussell/Mining-the-Social-Web/blob/master/web_code/dojo/facebook.current_locations_and_hometowns.html*.

O exemplo 9.19 apresenta uma lógica mínima para capturar diversas páginas de dados de notícias e computar uma simples estrutura JSON que é uma lista de tuplas `[term, URL, frequency]` que pode ser alimentada em um template HTML. Passaremos strings vazias para o trecho do URL dessas tuplas, mas, com um pouco de esforço extra, você poderia introduzir lógica adicional que mantivesse um mapa de quais termos ocorreram em cada post, para que os termos na nuvem de tags apontassem de volta para os dados de origem. Note que, como neste exemplo estamos buscando várias páginas de dados da API Graph, simplesmente optamos por fazer a interface diretamente com o endpoint da API para seu mural no Facebook (*http://www.facebook.com/help/?page=820*).

Exemplo 9.19 – Coleta e transformação dos dados para visualização como nuvem de tags do WP-Cumulus (facebook__tag_cloud.py)

```
# -*- coding: utf-8 -*-
import os
import sys
import urllib2
import json
import webbrowser
import nltk
from cgi import escape
from facebook__login import login
try:
    ACCESS_TOKEN = open('out/facebook.access_token').read()
except IOError, e:
    try:
        # Se você passar o token de acesso da aplicação Facebook
como um parâmetro de linha
        # de comando, não se esqueça de encapsulá-lo em aspas
simples para que o shell não
        # interprete nenhum de seus caracteres. Você também pode
ter de escapar o caractere #
        ACCESS_TOKEN = sys.argv[1]
    except IndexError, e:
        print >> sys.stderr, \
            "Could not find local access token or parse args.
Logging in..."
        ACCESS_TOKEN = login()
BASE_URL = 'https://graph.facebook.com/me/home?access_token='
HTML_TEMPLATE = '../web_code/wp_cumulus/tagcloud_template.html'
OUT_FILE = 'out/facebook.tag_cloud.html'
NUM_PAGES = 5
MIN_FREQUENCY = 3
MIN_FONT_SIZE = 3
MAX_FONT_SIZE = 20
# Faz um loop nas páginas de dados de conexão e constrói
mensagens
url = BASE_URL + ACCESS_TOKEN
messages = []
current_page = 0
while current_page < NUM_PAGES:
    data = json.loads(urllib2.urlopen(url).read())
    messages += [d['message'] for d in data['data'] if
d.get('message')]
    current_page += 1
```

```
    url = data['paging']['next']
# Computa a distribuição de frequência para os termos
fdist = nltk.FreqDist([term for m in messages for term in
m.split()])
# Personaliza uma lista de stop words como necessário
stop_words = nltk.corpus.stopwords.words('english')
stop_words += ['&', '.', '?', '!']
# Crie a saída para a nuvem de tags do WP-Cumulus e ordene os
termos por frequência
raw_output = sorted([[escape(term), '', freq] for (term, freq) in
fdist.items()
                     if freq > MIN_FREQUENCY and term not in
stop_words],
                     key=lambda x: x[2])
# Implementação adaptada de http://help.com/post/383276-anyone-
knows-the-formula-for-font-s
min_freq = raw_output[0][2]
max_freq = raw_output[-1][2]
def weightTermByFreq(f):
    return (f - min_freq) * (MAX_FONT_SIZE - MIN_FONT_SIZE) /
(max_freq
            - min_freq) + MIN_FONT_SIZE
weighted_output = [[i[0], i[1], weightTermByFreq(i[2])] for i in
raw_output]
# Substitua a estrutura de dados JSON no template
html_page = open(HTML_TEMPLATE).read() %
(json.dumps(weighted_output), )
f = open(OUT_FILE, 'w')
f.write(html_page)
f.close()
print 'Date file written to: %s' % f.name
# Abre a página web em seu navegador
webbrowser.open('file://' + os.path.join(os.getcwd(), OUT_FILE))
```

A figura 9.9 mostra alguns resultados de exemplo. Os dados para esta nuvem de tags foram coletados durante uma partida de futebol americano entre as universidades norte-americanas Boise State e Virginia Tech. Não deve ser nenhuma surpresa que a palavra mais usual nos feeds de notícia seja "I" (eu, em inglês). Em última análise, não foi necessário muito

esforço para produzir uma rápida visualização, mostrando sobre o que as pessoas estão conversando em seu mural, e você poderia facilmente estender esse resultado para analisar qualquer outra fonte de dados textuais. Entretanto, caso você esteja interessado na construção automática de uma nuvem de tags com conteúdo interessante, pode considerar a incorporação de algumas das técnicas de filtragem e NLP mais avançadas que apresentamos antes neste livro. Dessa forma, você estaria se dirigindo especificamente a termos que provavelmente são entidades, em vez de realizar uma análise de frequência simples. Lembre-se de que o capítulo 8 ofereceu uma visão geral bem sucinta sobre como você pode realizar tal procedimento.

Comentários finais

Você pode literalmente fazer praticamente tudo que puder imaginar no Facebook, e as vastas quantidades de dados que ele expõe por meio de suas poderosas APIs fornecem enormes oportunidades para criação de aplicações orientadas por dados inteligentes e incríveis. O tratamento superficial que demos à plataforma do Facebook neste capítulo não chega sequer a arranhar a superfície de tudo que é possível, mas esperamos que ele tenha fornecido muitas ferramentas que você poderá utilizar para se aprofundar e, talvez, se tornar o criador da mais nova incrível aplicação (orientada por dados) do Facebook.

Como exercício adicional, pode ser interessante visualizar o texto não-estruturado nos murais de seus amigos ou agrupar os murais com uma nuvem de tags, aplicando os tipos de análises centradas em entidades que apresentamos no capítulo 8, em vez de utilizar o esquema trivial de tokenização que vimos neste capítulo. Também poderia ser interessante trabalhar com os dados não-estruturados disponíveis por meio da API Graph para descobrir quem é a pessoa mais comunicativa de sua rede, com base no número geral de atualizações de estado e nos comentários que seus amigos postam. Outra ótima sugestão seria tentar agrupar seus amigos de acordo com o que gostam, e depois analisar essas preferências em uma tabela dinâmica (*http://en.wikipedia.org/wiki/Pivot_table*) ou dispô-las em um gráfico.

Figura 9.9 – Nuvem de tags giratória altamente personalizável e que requer muito pouco esforço para ser criada e utilizada.

---

1 N.T.: apps são programas que executam funções específicas em computadores ou dispositivos móveis, como smartphones, sendo voltadas ao usuário final (fonte: Wikipédia).

2 Vale a pena marcar como favorito e verificar regularmente o Developer Roadmap (*http://developers.facebook.com/roadmap*), para que você esteja sempre atualizado.

3 N.T.: Web Standards é um conjunto de normas, diretrizes, recomendações, notas, artigos, tutoriais e afins de caráter técnico, produzidos pela W3C. É destinado a orientar fabricantes, desenvolvedores e projetistas quanto ao uso de práticas que possibilitem a criação de uma web acessível a todos (fonte: Wikipédia).

4 Também conhecidos como gráficos radiais, árvores radiais, mapas radiais e muitos outros nomes que não necessariamente incluem o termo "radial" ou "raio".

5 Também comumente chamados de "pie piece", ou de "fatia". em gráficos de setores ou de "pizza".

6 Esse conceito foi apresentado na seção "Exploração de dados com o NLTK", no capítulo 7.

7 Ele foi apresentado na seção "Visualização de tweets com nuvens de tags", no capítulo 5.

CAPÍTULO 10

A web semântica: uma discussão descontraída

Enquanto os capítulos anteriores procuraram fornecer uma visão geral da web social e motivá-lo a explorar os dados que ela oferece, parece apropriado finalizar nosso texto com uma breve nota acerca da web semântica. Esta breve discussão não pretende reproduzir inúmeros trechos de listas de discussão, posts de blogs e outras fontes de informação que documentam a origem da web, agora que ela revolucionou praticamente tudo em nossas vidas em menos de duas décadas, nem mencionar como a web semântica sempre foi parte dessa visão. Nosso objetivo, por outro lado, será apresentar-lhe uma discussão descontraída que, mesmo evitando a extensão e profundidade dessas questões, esperamos que sirva para motivá-lo quanto às possibilidades existentes.

Uma revolução evolucionária?

Vamos iniciar analisando o termo "web semântica". Uma vez que a web diz respeito principalmente ao compartilhamento de informações, e que uma definição funcional de *semântica* seria "significado suficiente para resultar em uma ação"[1], não é absurdo deduzir que a web semântica trata principalmente da representação do conhecimento de modo mais significativo. Entretanto, levaremos essa definição um passo adiante, supondo que nem sempre será um ser humano que consumirá a informação representada. Nesse sentido, vamos considerar as possibilidades que poderiam se tornar realidade se as informações fossem compartilhadas de uma forma completamente *compreensível por máquina* – um modo suficientemente inequívoco, que permitisse a um *user agent* razoavelmente sofisticado, como um *web robot*, extrair, interpretar e utilizar a informação na tomada de decisões importantes. Alguns passos já foram dados nesse sentido: por exemplo, discutimos como os microformatos tornam isso possível em certos domínios no capítulo 2, e, no capítulo 9, vimos como o Facebook está agressivamente inserindo uma construção explícita de grafos na web, com seu Open Graph Protocol.

Todavia, antes de sonharmos tão alto, vamos retroceder por um momento e refletir sobre como chegamos à posição que ocupamos hoje.

A Internet é apenas uma rede de redes[2], e o que há de mais fascinante nisso, do ponto de vista técnico, é a forma como camadas de protocolos de níveis cada vez mais altos estão posicionadas sobre protocolos de níveis mais baixos para produzir, em última instância, uma infraestrutura computacional global tolerante a erros. Em nossa atividade on-line, dependemos de dúzias de protocolos diariamente, mesmo que isso não seja sempre evidente. Entretanto, há um protocolo na prática onipresente que dificilmente deixamos de encontrar com regularidade: HTTP, o prefixo de praticamente todos os URLs que você digita em seu navegador, e protocolo habilitador para o extenso universo de documentos de hipertexto (páginas HTML) e para os links que unem esses documentos no que conhecemos como a web. Porém, como você já deve saber, a web não trata apenas de hipertextos; ela também inclui diversas tecnologias integradas, como JavaScript, Flash e recursos HTML5 emergentes, como streams de áudio e vídeo. Todas essas interações ricas fazem com que o termo "hipertexto" pareça um pouco antiquado, não é mesmo?

A noção de um mundo virtual de documentos, plataformas e aplicações com os quais podemos interagir por meio de navegadores modernos (incluindo aqueles em dispositivos móveis ou *tablets*) utilizando HTTP é reconhecidamente genérica, mas está provavelmente muito próxima do que a maioria das pessoas imagina quando ouve falar na "Web". Até certo grau, a motivação por trás do raciocínio da Web 2.0, que emergiu em 2004, era definir de modo mais preciso a noção cada vez mais confusa do que exatamente era a web e no que ela estava se tornando. Nessas linhas, algumas pessoas pensam naquilo que chamamos de Web 1.0 como a web na forma em que ela existiu, de sua origem até a presente era (com aplicações web altamente interativas e colaboração do usuário); ao mesmo tempo, essas pessoas pensam na era atual como a era da Web 2.x, caracterizada por aplicações ricas de Internet (*Rich Internet Applications*, ou RIAs) e colaboração, e na era do karma semântico que está por vir, como a Web 3.0 (Tabela 10.1).

Tabela 10.1 – Diversas manifestações/eras da Web e suas virtudes

Manifestação/era	Virtudes
Internet	Protocolos de aplicação como SMTP, FTP, BitTorrent, HTTP etc.

Web 1.0	Páginas HTML e hiperlinks, em grande parte estáticos.
Web 2.0	Plataformas, colaborações e experiências ricas de usuário.
Web social (Web 2.x ???)	Pessoas e suas conexões e atividades sociais, virtuais e do mundo real.
Web 3.0 (a web semântica)	Quantidades prolíficas de conteúdo compreensível por máquina.

No presente, não há consenso real sobre o que significa a Web 3.0, mas a maioria das discussões sobre o assunto geralmente inclui a frase "web semântica", assim como a noção de informações que serão consumidas e utilizadas por máquinas de formas ainda impossíveis na escala atual da web. Por exemplo, ainda é muito difícil conseguir que máquinas extraiam e formulem inferências sobre fatos representados nos documentos disponíveis on-line. Buscas e heurísticas de palavras-chave podem certamente oferecer listagens de resultados de busca muito relevantes, mas uma inteligência humana continua sendo necessária para interpretar e sintetizar as informações dos documentos em si. Se a Web 3.0 e a web semântica têm o mesmo significado, isso ainda está em debate; entretanto, é geralmente aceito que o termo *web semântica* se refere a uma web muito semelhante à que conhecemos e amamos, mas que evoluiu ao ponto em que as máquinas são capazes não só de extrair, mas também de *atuar sobre* informações contidas nos documentos em nível granular.

Um homem não pode viver apenas de fatos

A construção fundamental que a web semântica utiliza para representar conhecimentos é conhecida como *tripla*, uma forma muito intuitiva e natural de expressar um fato. Como exemplo, a frase que consideramos em muitas situações prévias – "Mr. Green killed Colonel Mustard in the study with the candlestick" – poderia ser expressa em uma tripla como (*Mr. Green, killed, Colonel Mustard*), na qual seus elementos fazem referência ao sujeito, predicado e objeto da frase. O Resource Description Framework (RDF) é o modelo da web semântica para definição e habilitação da troca de triplas. O RDF é altamente extensível, no sentido em que fornece uma base sólida para expressão de conhecimentos, e também pode ser utilizado para definir vocabulários especializados conhecidos como *ontologias*, que oferecem semântica precisa para modelagem de domínios específicos. Fazer mais do que uma rápida menção das tecnologias específicas da web semântica, como RDF (*http://www.w3.org/RDF/*), RDFa (*http://www.w3.org/TR/xhtml-rdfa-primer/*), RDF Schema (*http://www.w3.org/TR/rdf-schema/*), e OWL

(*http://www.w3.org/TR/owl2-overview/*), estaria fora do escopo deste livro, ainda mais no final do texto, mas trabalharemos com um exemplo de alto-nível para tentar explicar a razão de todo o agito em torno da web semântica.

Suposições de mundo aberto versus de mundo fechado

Uma diferença interessante entre a forma como inferências funcionam em linguagens de programação lógica, como a Prolog[3], e o modo como isso ocorre em outras tecnologias, como a implementação RDF, está no uso de suposições de *mundo aberto* e de *mundo fechado* sobre o universo. Linguagens de programação lógica como a Prolog, e a maioria dos sistemas de bancos de dados, supõem um mundo fechado, enquanto a tecnologia RDF geralmente supõe um mundo aberto. Em um mundo fechado, tudo o que não foi dito explicitamente sobre o universo deve ser considerado falso, enquanto em um mundo aberto, tudo o que você não sabe é manipulado como indefinido (outra forma de dizermos "desconhecido"). A distinção significa que *reasoners*[4] que supõem um mundo aberto *não* descartarão interpretações que incluam fatos que não foram explicitamente declarados em uma base de conhecimento, enquanto *reasoners* que supõem um mundo fechado, como o da linguagem de programação Prolog, ou o da maioria dos sistemas de bancos de dados, *descartarão* fatos que não foram explicitamente declarados. Além disso, em um sistema que supõe um mundo fechado, a mescla de conhecimentos contraditórios geralmente disparará um erro, enquanto um sistema que supõe um mundo aberto pode tentar realizar novas inferências que, de alguma forma, conciliem as informações contraditórias. Como você pode imaginar, sistemas de mundo aberto são muito flexíveis e podem resultar no surgimento de questões muito interessantes; seu potencial pode se tornar especialmente evidente quando mesclamos conhecimentos distintos.

Intuitivamente, você pode pensar nisso da seguinte maneira: sistemas construídos sobre um raciocínio de mundo fechado supõem que os dados que recebem estão completos, e são tipicamente não-monotônicos[5], no sentido de que não é sempre que todos os fatos prévios (explícitos ou inferidos) ainda serão válidos quando novos forem adicionados. Em contraste, sistemas de mundo aberto não fazem tal suposição acerca da completude de seus dados e são monotônicos. Como você pode imaginar,

há muito debate sobre os méritos de cada uma dessas suposições. Como alguém interessado na web semântica, você deve ao menos estar ciente dessa discussão. Na forma como esse assunto se relaciona especificamente com o RDF, a orientação oficial da documentação do W3C afirma[6]:

> Para facilitar operações na escala da Internet, o RDF é um framework de mundo aberto que permite a qualquer pessoa formular declarações sobre qualquer recurso. Em geral, não se presume que estejam disponíveis informações completas sobre qualquer recurso. O RDF não impede ninguém de fazer asserções que sejam sem sentido e inconsistentes com outras declarações, ou com o mundo como as pessoas o veem. Designers de aplicações que utilizam RDF devem estar cientes disso e podem projetar suas aplicações para tolerar fontes incompletas ou inconsistentes de informação.

Também pode ser interessante consultar o texto "Position Paper: A Comparison of Two Modelling Paradigms in the Semantic Web" (*http://www2006.org/programme/files/xhtml/4015/4015-patel-schneider/4015-patel-schneider-xhtml.html*), de Peter Patel-Schneider e Ian Horrocks, caso você esteja interessado em se aprofundar neste tópico. Independentemente de decidir fazê-lo agora, lembre-se de que os dados disponíveis na web estão incompletos, e que fazer uma suposição de mundo fechado (por exemplo, considerar todas as informações desconhecidas enfaticamente falsas) resultará, mais cedo ou mais tarde, em consequências severas.

Inferências sobre um mundo aberto com FuXi

Linguagens fundacionais, como RDF Schema e OWL, são projetadas de modo que vocabulários precisos possam ser utilizados para expressar fatos, como a tripla (*Mr. Green, killed, Colonel Mustard*), de forma legível por máquinas; e esta é uma condição necessária, mas não suficiente, para que a web semântica se torne realidade. Em termos gerais, assim que você tiver um conjunto de fatos, o próximo passo será realizar *inferências* sobre eles e formular conclusões que sejam consequências. O conceito de inferência formal data ao menos da Grécia Antiga, com os silogismos de Aristóteles, e a conexão óbvia quanto à forma como as máquinas podem aproveitá-lo não passou despercebida aos pesquisadores de inteligência artificial nos últimos 50 anos. O panorama das tecnologias com base em Java, repleto de opções comerciais como Jena (*http://jena.sourceforge.net/*) e Sesame (*http://www.openrdf.org/*), certamente parece ser o local onde

acontecem os principais avanços, mas felizmente também temos algumas sólidas opções para trabalhar com Python.

Dentre o que você poderá encontrar, uma das melhores opções em Python capazes de inferência é o FuXi (*http://code.google.com/p/fuxi/*), poderoso sistema de raciocínio lógico para a web semântica que utiliza uma técnica conhecida como encadeamento progressivo (*forward chaining*, *http://en.wikipedia.org/wiki/Forward_chaining*) para deduzir novas informações a partir de informações existentes. Para tanto, partimos de um conjunto de fatos, derivamos novos fatos dos fatos conhecidos, aplicando um conjunto de regras lógicas, e repetimos esse processo até que uma conclusão específica possa ser provada ou refutada, ou até que não haja novos fatos para derivar. O tipo de encadeamento progressivo que o FuXi oferece é tido tanto como *sólido*, uma vez que todos os novos fatos produzidos são verdadeiros, quanto como *completo*, uma vez que todos os fatos verdadeiros podem ser eventualmente provados. Uma discussão extensa sobre lógica proposicional e de primeira ordem poderia facilmente preencher um livro inteiro; caso você esteja interessado em se aprofundar, o texto clássico *Artificial Intelligence: A Modern Approach*, de Stuart Russell e Peter Norvig (Prentice Hall), talvez seja a fonte mais completa.

Para demonstrar os tipos de capacidades de inferência que um sistema como o FuXi pode oferecer, vamos considerar o famoso exemplo do silogismo[7] de Aristóteles, no qual você recebe uma base de conhecimento que contém os fatos "Sócrates é homem" e "Todos os homens são mortais", e que nos permite deduzir que "Sócrates é mortal". Ainda que esse problema possa parecer excessivamente trivial, lembre-se de que os mesmos algoritmos deterministas que produzem o novo fato "Sócrates é mortal" trabalham da mesma forma quando há muito mais fatos disponíveis – e esses novos fatos podem produzir ainda outros novos fatos, que produzirão mais novos fatos e assim por diante. Por exemplo, considere uma base de conhecimento um pouco mais complexa, contendo alguns fatos adicionais:

- Sócrates é homem.
- Todos os homens são mortais.
- Apenas deuses vivem no Monte Olimpo.
- Todos os mortais bebem uísque.

- Chuck Norris vive no Monte Olimpo.

Se você fosse apresentado a essa base de conhecimento e lhe perguntassem "Por acaso Sócrates bebe uísque?", você teria primeiro de deduzir o fato de que "Sócrates é mortal", antes que pudesse deduzir, como consequência, que "Sócrates bebe uísque". Para ilustrar como isso seria feito em código, considere a mesma base de conhecimento, agora expressa em Notation3 (*http://www.w3.org/DesignIssues/Notation3*, sigla N3), como mostra o exemplo 10.1. A N3 é uma sintaxe simples, mas poderosa, que expressa fatos e regras em RDF.

Ainda que haja muitos formatos diferentes para expressar RDF, muitas ferramentas de web semântica escolhem o formato N3 em razão de sua legibilidade e expressividade.

Exemplo 10.1 – Pequena base de conhecimento expressa com Notation3

```
#Atribui um namespace para predicados lógicos
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

#Atribui um namespace para o vocabulário definido neste documento
@prefix : <MiningTheSocialWeb#> .

#Sócrates é homem
:Socrates a :Man.

@forAll :x .

#Todos os homens são mortais: Man(x) => Mortal(x)
{ :x a :Man } log:implies { :x a :Mortal } .

#Apenas deuses vivem no Monte Olimpo: Lives(x, MtOlympus) <=>
God(x)
{ :x :lives :MtOlympus } log:implies { :x a :god } .
{ :x a :god } log:implies { :x :lives :MtOlympus } .

#Todos os mortais bebem uísque: Mortal(x) => Drinks(x, whisky)
{ :x a :Man } log:implies { :x :drinks :whisky } .

#Chuck Norris vive no Monte Olimpo: Lives(ChuckNorris, MtOlympus)
:ChuckNorris :lives :MtOlympus .
```

Executar o FuXi com a opção `--ruleFacts` diz a ele que é preciso fazer o parsing dos fatos da fonte de entrada que você pode especificar com a opção `--rules`, e também que é necessário acumular fatos adicionais da fonte. Caso execute o FuXi a partir da linha de comando, uma saída semelhante à que temos no exemplo 10.2 deve ser o resultado. Note que o FuXi estará presente em seu path depois que você executar `easy_install`

`fuxi`.

Exemplo 10.2 – Resultados da execução do FuXi a partir da linha de comando sobre a base de conhecimento do exemplo 10.1

```
$ FuXi --rules=foo.n3 --ruleFacts
@prefix _7: <file:///Users/matthew/MiningTheSocialWeb#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
_7:ChuckNorris a _7:god.
_7:Socrates a _7:Mortal;
    _7:drinks _7:whisky.
```

A saída do programa nos informa alguns dados não declarados explicitamente na base de conhecimento inicial: Chuck Norris é um deus, Sócrates é mortal e Sócrates bebe uísque. Ainda que a derivação desses fatos possa parecer óbvia para a maioria dos seres humanos, isso não é tão simples para uma máquina – e é justamente por isso que esse resultado é tão empolgante.

Cabe-nos lembrá-lo de que a declaração descuidada de fatos sobre Chuck Norris (mesmo no contexto de uma base de conhecimento de exemplo) pode ser prejudicial à sua saúde, ou à vida de seu computador. Você foi avisado.

Caso este simples exemplo tenha conseguido empolgá-lo, não deixe de se aprofundar no FuXi e no potencial da web semântica. Pode-se dizer que a web semântica é comprovadamente mais complexa que a web social, e investigá-la é um esforço verdadeiramente digno – especialmente se você estiver interessado nas possibilidades que as inferências trazem aos dados sociais.

Esperança

Ainda que a omissão intencional de uma discussão referente à web semântica durante este pequeno livro possa ter dado a impressão de que há uma distinção arbitrária e rígida entre a web social e a web semântica, na prática a divisão entre esses dois conceitos é bem nebulosa e está em constante estado de fluxo. É muito provável que a inegável proliferação de dados sociais na web – combinada com outras iniciativas, como os microformatos publicados por empresas como o Food Network e o LinkedIn, o Open Graph Protocol do Facebook e os esforços criativos de hackers de dados como você – esteja acelerando rapidamente o

surgimento de uma web semântica que poderá não ser muito diferente daquela tão anunciada com exagero nos últimos 15 anos. A proliferação dos dados sociais tem o potencial de ser um grande catalisador para o desenvolvimento da web semântica, que permitirá que agentes efetuem ações mais elaboradas em seu nome. Ainda não estamos nesse ponto, mas tenha esperança! Ou, para dizermos de outra forma, ao menos considere as sábias palavras de Epicuro, que disse: "Não estrague o que você tem desejando o que não tem; mas lembre-se de que o que você agora possui um dia já esteve entre as coisas com relação às quais você só tinha esperança."

---

1 Como definido em Programming the Semantic Web (*http://oreilly.com/catalog/9780596153823/*), de Toby Segaran, Jamie Taylor e Colin Evans (O'Reilly).

2 Inter-net implica literalmente "redes mútuas ou cooperantes".

3 Recomendamos que você consulte uma autêntica linguagem de programação lógica, como a Prolog, escrita em um paradigma projetado especificamente para que você possa representar conhecimento e deduzir informações a partir de fatos existentes. O GNU Prolog (*http://www.gprolog.org/*) é um ótimo ponto de partida.

4 N.T.: um reasoner semântico, engine de regras, ou simplesmente reasoner, é um software capaz de inferir consequências lógicas de um conjunto de fatos declarados ou axiomas. A noção de um reasoner semântico generaliza a de um engine de inferência, fornecendo um conjunto mais rico de mecanismos com os quais se pode trabalhar (fonte: Wikipédia).

5 N.T.: nesse mesmo sentido, uma lógica não-monotônica é uma lógica formal cuja relação de consequência não é monotônica. A maioria das lógicas formais tem uma relação de consequência monotônica, o que significa que adicionar uma fórmula a uma teoria nunca produz uma redução de seu conjunto de consequências. Intuitivamente, a monotonicidade indica que aprender um novo conhecimento não pode reduzir o conjunto do que já é sabido (fonte: Wikipédia).

6 Consulte *http://www.w3.org/TR/rdf-concepts/*.

7 No linguajar moderno, um silogismo é também conhecido como uma "implicação".

Sobre o autor

Matthew E. Russel, vice-presidente de engenharia na Digital Reasoning Systems e Diretor na Zaffra, é um cientista da computação apaixonado por mineração de dados, código aberto e tecnologias de aplicações web. Também é autor de *Dojo: The Definitive Guide* (*http://oreilly.com/catalog/9780596516482/*), da O'Reilly. Conecte-se a ele no LinkedIn, ou siga *@ptwobrussell* no Twitter, para acompanhar todas as suas raras atualizações profissionais.

Colofão

O animal na capa do livro *Mineração de dados da web social* é uma marmota (*Marmota monax*), também conhecida em inglês como *woodchuck* (nome derivado do algonquino *wuchak*). Marmotas são popularmente associadas ao feriado norte-americano e canadense chamado Groundhog Day (Dia da Marmota), celebrado sempre no dia dois de fevereiro. O folclore diz que, se a marmota sair de sua toca nesse dia e enxergar sua própria sombra, o inverno continuará por mais seis semanas. Adeptos dizem que o roedor prevê as condições climáticas corretamente, numa escala que vai de 75 a 90% dos casos. Muitas cidades hospedam marmotas famosas por suas habilidades de previsão, incluindo a marmota Punxsutawney Phil (de Punxsutawney, PA, mostrada no filme Feitiço do Tempo, de 1993, com Bill Murray).

Essa lenda provavelmente tem origem no fato de que a marmota é uma das poucas espécies que entram em verdadeira hibernação durante o inverno. Basicamente herbívoras, marmotas engordam durante o verão, alimentando-se de vegetação, frutos, nozes, insetos e plantações de jardineiros humanos, fazendo com que muitos as considerem pestes. Depois, cavam uma toca de inverno, e permanecem dentro dela de outubro a março do ano seguinte (ainda que saiam mais cedo em áreas de clima temperado, ou, supostamente, quando se tornam o centro das atenções no feriado que lhes é dedicado).

A marmota é o maior membro da família dos esquilos, medindo cerca de

40 a 65 centímetros de comprimento e pesando de 2 a 4 quilos. Apresenta garras longas e curvas, ideais para cavar, e duas camadas de pele: uma inferior, densa e acinzentada, e uma superior, de cor mais clara e de pelos mais compridos, que serve como proteção contra os elementos.

Marmotas ocupam grande parte do Canadá e das regiões do norte dos Estados Unidos, em locais onde os espaços abertos encontram as florestas. Ainda que sejam capazes de escalar árvores e de nadar, geralmente são encontradas no solo, próximas às tocas que cavam para dormir, cuidar de seus filhotes e que utilizam como proteção contra os predadores. Suas tocas geralmente têm de duas a cinco entradas e até 14 metros de túneis.

A imagem da capa é do livro Animate Creatures, de J.G. Wood.


O'REILLY
Analítica de dados com Hadoop
UMA INTRODUÇÃO PARA CIENTISTAS DE DADOS
novatec
Benjamin Bengfort e Jenny Kim

Analítica de dados com Hadoop

Bengfort, Benjamin
9788575227626
352 páginas

Compre agora e leia

Pronto para usar técnicas estatísticas e de aprendizado de máquina (machine learning) em grandes conjuntos de dados? Este guia prático mostra por que o ecossistema do Hadoop é perfeito para essa tarefa. Em vez de ter como foco a implantação, as operações ou o desenvolvimento de softwares geralmente associados à computação distribuída, você se concentrará nas análises particulares que poderá fazer, nas técnicas de armazém de dados (data warehousing) oferecidas pelo Hadoop e em fluxos de trabalho de alta ordem que esse framework é capaz de gerar. Os cientistas e os analistas de dados aprenderão a usar diversas técnicas que variam da escrita de aplicações

MapReduce e Spark com Python ao uso de modelagem avançada e gerenciamento de dados com Spark MLlib, Hive e HBase. Você também conhecerá os processos analíticos e os sistemas de dados disponíveis para desenvolver e conferir eficácia aos produtos de dados capazes de lidar com – e que, na verdade, exigem – quantidades enormes de dados. • Entenda os conceitos principais do Hadoop e do processamento em cluster. • Utilize padrões de projeto e algoritmos analíticos paralelos para criar jobs de análise de dados distribuídos. • Adquira conhecimentos sobre gerenciamento de dados, mineração e armazém de dados em um contexto distribuído usando Apache Hive e HBase. • Utilize Sqoop e Apache Flume para entrada de dados a partir de bancos de dados relacionais. • Programe aplicações Hadoop e Spark complexas com Apache Pig e Spark DataFrames. • Utilize técnicas de aprendizado de máquina, como classificação, clustering e filtragem colaborativa, com a MLib do Spark.

Compre agora e leia

O'REILLY

Padrões para Kubernetes

Elementos reutilizáveis no design de aplicações nativas de nuvem

Bilgin Ibryam
Roland Huß

novatec

Padrões para Kubernetes

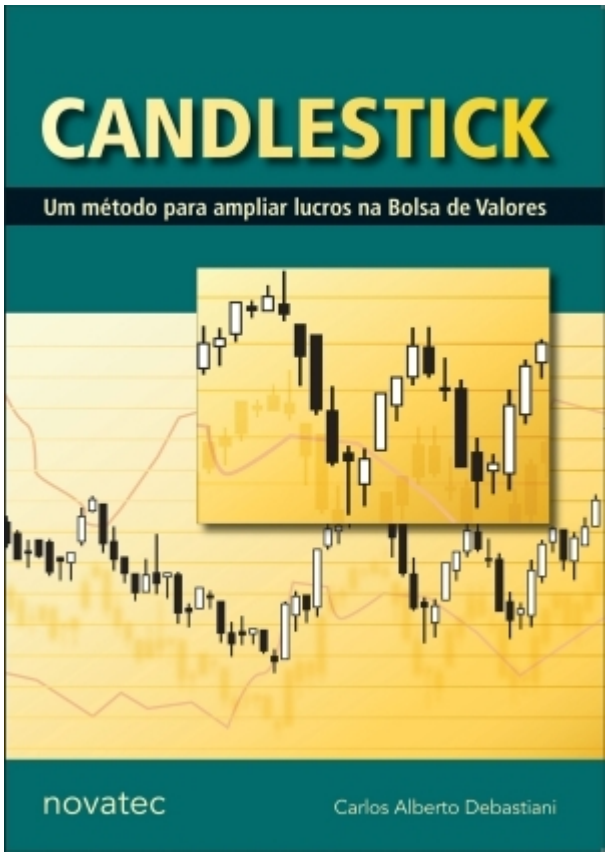
Ibryam, Bilgin
9788575228159
272 páginas

Compre agora e leia

O modo como os desenvolvedores projetam, desenvolvem e executam software mudou significativamente com a evolução dos microsserviços e dos contêineres. Essas arquiteturas modernas oferecem novas primitivas distribuídas que exigem um conjunto diferente de práticas, distinto daquele com o qual muitos desenvolvedores, líderes técnicos e arquitetos estão acostumados. Este guia apresenta padrões comuns e reutilizáveis, além de princípios para o design e a implementação de aplicações nativas de nuvem no Kubernetes. Cada padrão inclui uma descrição do problema e uma solução específica no Kubernetes. Todos os padrões acompanham e são demonstrados por

exemplos concretos de código. Este livro é ideal para desenvolvedores e arquitetos que já tenham familiaridade com os conceitos básicos do Kubernetes, e que queiram aprender a solucionar desafios comuns no ambiente nativo de nuvem, usando padrões de projeto de uso comprovado. Você conhecerá as seguintes classes de padrões: • Padrões básicos, que incluem princípios e práticas essenciais para desenvolver aplicações nativas de nuvem com base em contêineres. • Padrões comportamentais, que exploram conceitos mais específicos para administrar contêineres e interações com a plataforma. • Padrões estruturais, que ajudam você a organizar contêineres em um Pod para tratar casos de uso específicos. • Padrões de configuração, que oferecem insights sobre como tratar as configurações das aplicações no Kubernetes. • Padrões avançados, que incluem assuntos mais complexos, como operadores e escalabilidade automática (autoscaling).

Compre agora e leia


CANDLESTICK
Um método para ampliar lucros na Bolsa de Valores
novatec
Carlos Alberto Debastiani

Candlestick

Debastiani, Carlos Alberto

9788575225943

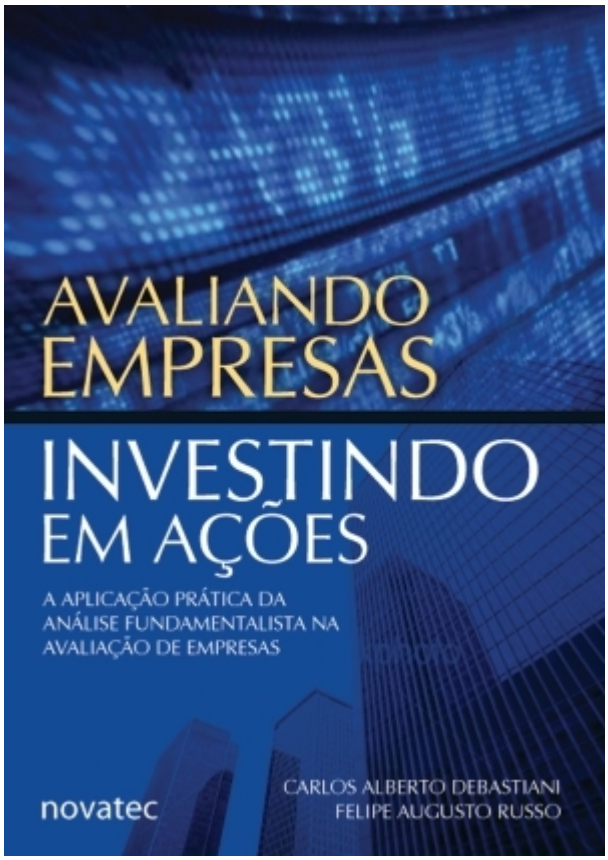
200 páginas

Compre agora e leia

A análise dos gráficos de Candlestick é uma técnica amplamente utilizada pelos operadores de bolsas de valores no mundo inteiro. De origem japonesa, este refinado método avalia o comportamento do mercado, sendo muito eficaz na previsão de mudanças em tendências, o que permite desvendar fatores psicológicos por trás dos gráficos, incrementando a lucratividade dos investimentos. Candlestick → Um método para ampliar lucros na Bolsa de Valores é uma obra bem estruturada e totalmente ilustrada. A preocupação do autor em utilizar uma linguagem clara e acessível a torna leve e de fácil assimilação, mesmo para leigos. Cada padrão de análise abordado possui um modelo com

sua figura clássica, facilitando a identificação. Depois das características, das peculiaridades e dos fatores psicológicos do padrão, é apresentado o gráfico de um caso real aplicado a uma ação negociada na Bovespa. Este livro possui, ainda, um índice resumido dos padrões para pesquisa rápida na utilização cotidiana.

Compre agora e leia


AVALIANDO
EMPRESAS
INVESTINDO
EM AÇÕES
A APLICAÇÃO PRÁTICA DA
ANÁLISE FUNDAMENTALISTA NA
AVALIAÇÃO DE EMPRESAS
novatec
CARLOS ALBERTO DEBASTIANI
FELIPE AUGUSTO RUSSO

Avaliando Empresas, Investindo em Ações

Debastiani, Carlos Alberto

9788575225974

224 páginas

Compre agora e leia

Avaliando Empresas, Investindo em Ações é um livro destinado a investidores que desejam conhecer, em detalhes, os métodos de análise que integram a linha de trabalho da escola fundamentalista, trazendo ao leitor, em linguagem clara e acessível, o conhecimento profundo dos elementos necessários a uma análise criteriosa da saúde financeira das empresas, envolvendo indicadores de balanço e de mercado, análise de liquidez e dos riscos pertinentes a fatores setoriais e conjunturas econômicas nacional e internacional. Por meio de exemplos práticos e ilustrações, os autores exercitam os conceitos teóricos abordados, desde os

fundamentos básicos da economia até a formulação de estratégias para investimentos de longo prazo.

Compre agora e leia


Marcos Abe
MANUAL DE
ANÁLISE
TÉCNICA
ESSÊNCIA E ESTRATÉGIAS AVANÇADAS
TUDO O QUE UM INVESTIDOR PRECISA SABER PARA
PROSPERAR NA BOLSA DE VALORES ATÉ EM TEMPOS DE CRISE
novatec

Manual de Análise Técnica

Abe, Marcos

9788575227022

256 páginas

Compre agora e leia

Este livro aborda o tema Investimento em Ações de maneira inédita e tem o objetivo de ensinar os investidores a lucrarem nas mais diversas condições do mercado, inclusive em tempos de crise. Ensinará ao leitor que, para ganhar dinheiro, não importa se o mercado está em alta ou em baixa, mas sim saber como operar em cada situação. Com o Manual de Análise Técnica o leitor aprenderá: - os conceitos clássicos da Análise Técnica de forma diferenciada, de maneira que assimile não só os princípios, mas que desenvolva o raciocínio necessário para utilizar os gráficos como meio de interpretar os movimentos da massa de investidores do mercado; - identificar oportunidades para lucrar na bolsa de valores, a

longo e curto prazo, até mesmo em mercados baixistas; um sistema de investimentos completo com estratégias para abrir, conduzir e fechar operações, de forma que seja possível maximizar lucros e minimizar prejuízos; estruturar e proteger operações por meio do gerenciamento de capital. Destina-se a iniciantes na bolsa de valores e investidores que ainda não desenvolveram uma metodologia própria para operar lucrativamente.

Compre agora e leia