

Mineração de texto e análise de texto



Este capítulo cobre

- Compreendendo a importância da mineração de texto
- Apresentando os conceitos mais importantes da mineração de texto
- Trabalhando em um projeto de mineração de texto

A maior parte da informação humana registrada no mundo está na forma de texto escrito.

Todos nós aprendemos a ler e escrever desde a infância para que possamos nos expressar através da escrita e aprender o que os outros sabem, pensam e sentem. Usamos essa habilidade o tempo todo ao ler ou escrever um e-mail, um blog, mensagens de texto ou este livro, por isso não é de admirar que a linguagem escrita seja natural para a maioria de nós. As empresas estão convencidas de que pode ser encontrado muito valor nos textos que as pessoas produzem, e com razão porque contêm informações sobre o que essas pessoas gostam, não gostam, o que sabem ou gostariam de saber, anseiam e desejam, a sua saúde ou humor actual, e muito mais. Muitas destas coisas podem ser relevantes para empresas ou investigadores, mas nenhuma pessoa consegue ler e interpretar sozinha este tsunami de material escrito. Mais uma vez, precisamos recorrer aos computadores para fazer o trabalho por nós.

Infelizmente, porém, a linguagem natural não é tão "natural" para os computadores quanto para os humanos. Obtendo significado e filtrando o que não é importante

o importante ainda é algo em que um ser humano é melhor do que qualquer máquina. Felizmente, dados os cientistas podem aplicar técnicas específicas de mineração de texto e de análise de texto para encontrar informações relevantes em pilhas de texto que, de outra forma, levariam séculos para ler eles mesmos.

A *mineração de texto* ou *análise de texto* é uma disciplina que combina ciência da linguagem e ciência da computação com técnicas estatísticas e de aprendizado de máquina. **A mineração de texto é usada para analisar textos e transformá-los em uma forma mais estruturada.** Então é preciso isso forma estruturada e tenta extrair insights dela. Ao analisar o crime da polícia relatórios, por exemplo, a mineração de texto ajuda a reconhecer pessoas, lugares e tipos de crimes dos relatórios. Depois, esta nova estrutura é usada para obter informações sobre a evolução dos crimes. Veja a figura 8.1.

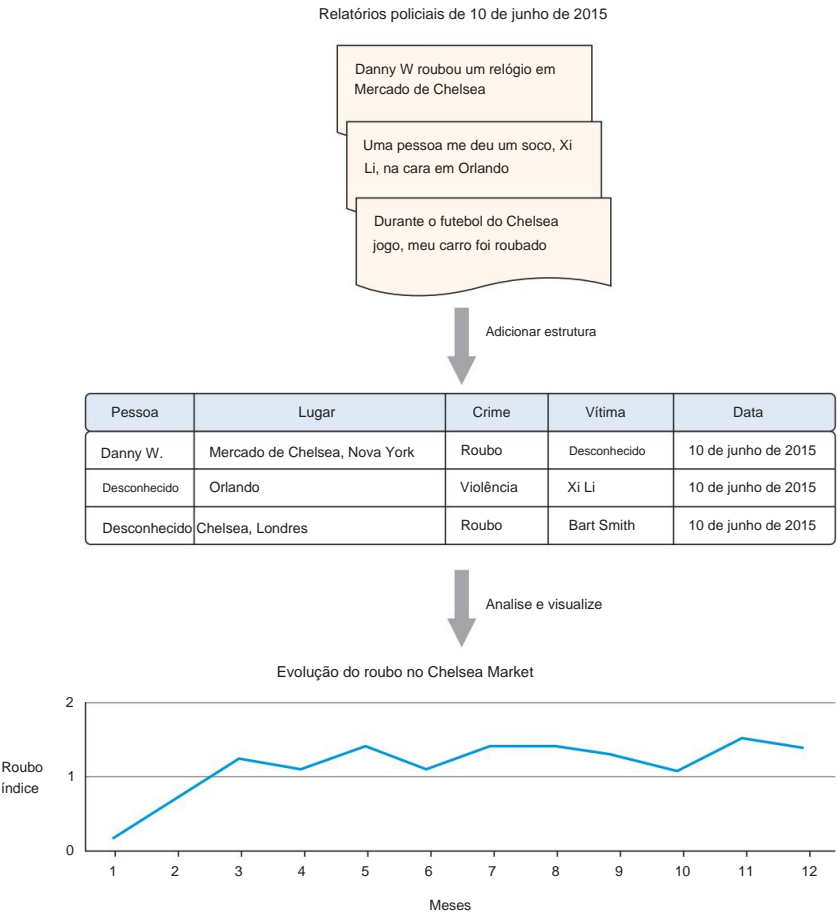


Figura 8.1 Na análise de texto, (normalmente) o primeiro desafio é estruturar o texto de entrada; então ele pode ser completamente analisado.

Embora a linguagem não se limite à linguagem natural, o foco deste capítulo será o *Processamento de Linguagem Natural (PNL)*. Exemplos de linguagens não naturais seriam registros de máquina, matemática e código Morse. Tecnicamente, mesmo as línguas Esperanto, Klingon e Dragão não estão no campo das línguas naturais porque foram inventadas deliberadamente em vez de evoluir ao longo do tempo; eles não vieram “naturais” para nós.

Estas últimas línguas são, no entanto, adequadas à comunicação natural (fala, escrita); eles têm uma gramática e um vocabulário como todas as línguas naturais, e as mesmas técnicas de mineração de texto poderiam ser aplicadas a eles.

8.1 Mineração de texto no mundo real

No seu dia a dia você já se deparou com aplicações de mineração de texto e linguagem natural. O preenchimento automático e os corretores ortográficos analisam constantemente o texto que você digita antes de enviar um e-mail ou mensagem de texto. Quando o Facebook completa automaticamente seu status com o nome de um amigo, ele faz isso com a ajuda de uma técnica chamada *reconhecimento de entidade nomeada*, embora isso seja apenas um componente de seu repertório. O objetivo não é apenas detectar que você está digitando um substantivo, mas também adivinhar que você está se referindo a uma pessoa e reconhecer quem pode ser. Outro exemplo de reconhecimento de entidade nomeada é mostrado na figura 8.2. O Google sabe que o Chelsea é um clube de futebol, mas responde de forma diferente quando questionado sobre uma pessoa.

O Google usa muitos tipos de mineração de texto ao apresentar os resultados de uma consulta. O que vem à sua mente quando alguém diz “Chelsea”? Chelsea poderia ser muitas coisas: uma pessoa; um clube de futebol; um bairro em Manhattan, Nova Iorque ou Londres; um mercado de alimentos; uma exposição de flores; e assim por diante. O Google sabe disso e retorna respostas diferentes para a pergunta “Quem é Chelsea?” versus “O que é Chelsea?”

Para fornecer a resposta mais relevante, o Google deve fazer (entre outras coisas) o seguinte:

- Pré-processar todos os documentos coletados para entidades nomeadas
- Executar identificação de idioma
- Detectar a que tipo de entidade você está se referindo
- Corresponder uma consulta a um resultado
- Detectar o tipo de conteúdo a ser retornado (PDF, sensível a adultos)

Este exemplo mostra que a mineração de texto não envolve apenas o significado direto do texto em si, mas também envolve metaatributos como idioma e tipo de documento.

O Google usa mineração de texto para muito mais do que responder perguntas. Além de proteger os usuários do Gmail contra spam, ele também divide os e-mails em diferentes categorias, como sociais, atualizações e fóruns, conforme mostrado na figura 8.3.

É possível ir muito além de responder perguntas simples quando você combina texto com outras lógicas e matemáticas.

The figure consists of two screenshots of Google search results. The top screenshot is for the query "what is chelsea" and shows results for Chelsea F.C., Chelsea, London, and Urban Dictionary. The bottom screenshot is for the query "who is chelsea" and shows results for Chelsea Handler and Chelsea F.C. Both screenshots show the search bar, navigation tabs (Web, Images, Maps, News, Videos, More), and search tools. The results are organized into a grid with a sidebar on the right for the "what is chelsea" query.

what is chelsea

About 202,000,000 results (0,48 seconds)

Chelsea F.C. - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Chelsea_F.C.
 Chelsea Football Club / tʃɛlsɪ / are a professional football club based in Fulham, London, who play in the Premier League, the highest level of English football. Founded in 1905, the club have spent most of their history in the top tier of English football.
 2014–15 Chelsea FC season - Roman Abramovich - Stamford Bridge - Eden Hazard

Chelsea, London - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Chelsea_London
 Chelsea is an affluent area in central London, bounded to the south by the River Thames. Its frontage runs from Chelsea Bridge along the Chelsea Embankment, Cheyne Walk, Lots Road and Chelsea Harbour.
 History - The borough of artists - Swinging Chelsea and today - Sports

Urban Dictionary: Chelsea
www.urbandictionary.com/define.php?term=Chelsea
 Chelsea is a beautiful creature of a peculiar nature. She is often starving or not hungry in the least, but she is dangerous in her hungry state. Possibly the sexiest ...

Chelsea F.C.
 Football club
 Chelsea Football Club are a professional football club based in Fulham, London, who play in the Premier League, the highest level of English football. Founded in 1905, the club have spent most of their history in the top tier of English football. [Wikipedia](#)
Manager: José Mourinho
League: Premier League
Arena/Stadium: Stamford Bridge
Training ground: Cobham Training Centre
Founded: March 10, 1905
Founders: Gus Mears, Joseph Mears

who is chelsea

About 198,000,000 results (0,37 seconds)

Chelsea Handler - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Chelsea_Handler
 Chelsea Joy Handler (born February 25, 1975) is an American comedian, actress, author, television host, producer, and activist for gay rights. She hosted a late-night talk show called Chelsea Lately on the E! network from 2007 to 2014, and is currently preparing to host a show on Netflix in 2016.
 Ted Harbert - Chelsea Lately - Uganda Be Kidding Me: Live - Ford Pinto

See results about
[Chelsea F.C. \(Football club\)](#)
Manager: José Mourinho
League: Premier League

Feedback

Figura 8.2 As diferentes respostas às perguntas “Quem é o Chelsea?” e “O que é Chelsea?” implica que o Google usa técnicas de mineração de texto para responder a essas perguntas.

The screenshot shows an email inbox interface with three tabs: Primary, Social, and Promotions. Below the tabs, there are two email entries, both from "Stack Exchange". The first entry has a subject line "2 new items in your Stack Exchange inbox - The follow" and the second entry has a subject line "1 new item in your Stack Exchange inbox - The follow".

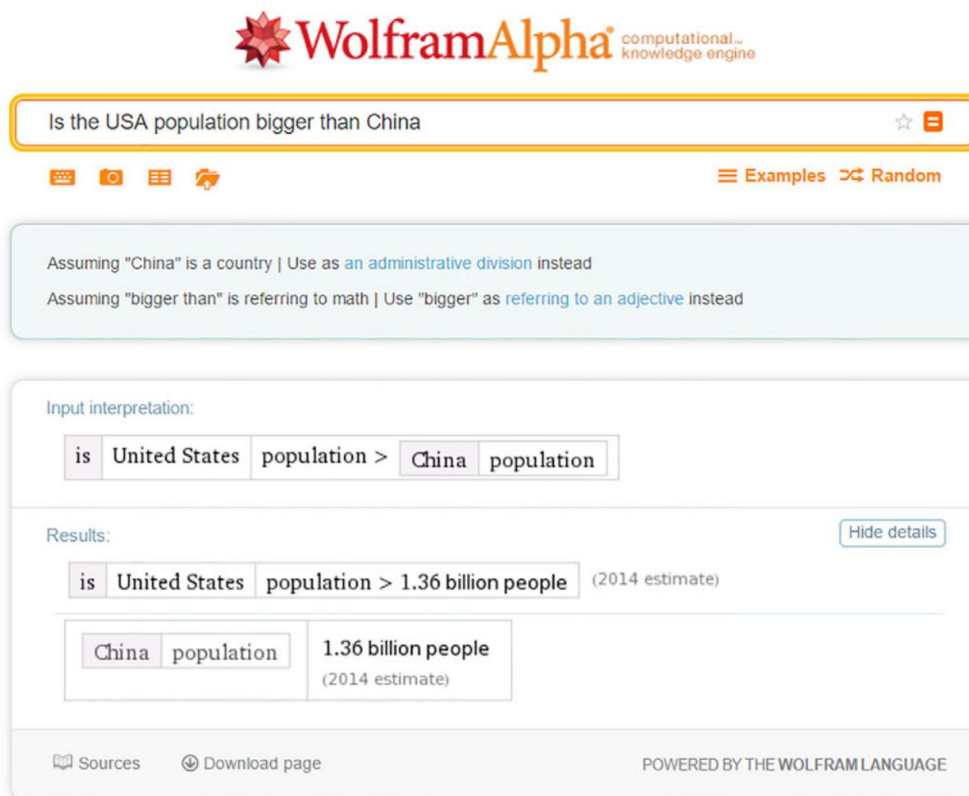
Primary **Social** **Promotions**

☐ **Stack Exchange** **2 new items in your Stack Exchange inbox - The follow**

☐ **Stack Exchange** **1 new item in your Stack Exchange inbox - The follow**

Figura 8.3 Os e-mails podem ser automaticamente divididos por categoria com base no conteúdo e na origem.

Isto permite a criação de *motores de raciocínio automático* impulsionados pela linguagem natural. consultas. A Figura 8.4 mostra como o “Wolfram Alpha”, um mecanismo de conhecimento computacional, utiliza mineração de texto e raciocínio automático para responder à pergunta “A população dos EUA é maior que a da China?”



The screenshot shows the Wolfram Alpha interface. At the top is the Wolfram Alpha logo with the tagline 'computational knowledge engine'. Below the logo is a search bar containing the query 'Is the USA population bigger than China'. To the right of the search bar are icons for a star and a menu. Below the search bar are icons for various input methods (keyboard, voice, image, etc.) and links for 'Examples' and 'Random'. A light blue box below the search bar contains assumptions: 'Assuming "China" is a country | Use as an administrative division instead' and 'Assuming "bigger than" is referring to math | Use "bigger" as referring to an adjective instead'. The main content area shows the 'Input interpretation:' section with the query broken down into 'is', 'United States', 'population >', 'China', and 'population'. Below this is the 'Results:' section, which shows the same query broken down into 'is', 'United States', 'population > 1.36 billion people (2014 estimate)', 'China', 'population', and '1.36 billion people (2014 estimate)'. At the bottom of the results section are links for 'Sources' and 'Download page', and a note that the engine is 'POWERED BY THE WOLFRAM LANGUAGE'.

Figura 8.4 O mecanismo Wolfram Alpha usa mineração de texto e raciocínio lógico para responder a uma pergunta.

Se isto não for suficientemente impressionante, o IBM Watson surpreendeu muitos em 2011, quando o A máquina foi configurada contra dois jogadores humanos em um jogo de *Jeopardy*. O *perigo* é um Quiz show americano onde as pessoas recebem a resposta a uma pergunta e os pontos são pontuado por adivinhar a pergunta correta para essa resposta. Veja a figura 8.5.

É seguro dizer que esta rodada vai para a inteligência artificial. IBM Watson é um programa cognitivo mecanismo que pode interpretar a linguagem natural e responder perguntas com base em uma ampla base de conhecimento.



Figura 8.5 IBM Watson vence o Jeopardy contra jogadores humanos.

A mineração de texto tem muitas aplicações, incluindo, entre outras, as seguintes:

- Identificação de entidades
- Detecção de plágio
- Identificação de tópicos
- Agrupamento de texto
- Tradução
- Resumo automático de texto
- Detecção de fraude
- Filtragem de spam
- Análise de sentimento

A mineração de texto é útil, mas é difícil? Desculpe desapontar: sim, é.

Ao observar os exemplos do Wolfram Alpha e do IBM Watson, você deve ter ficado com a impressão de que a mineração de texto é fácil. Infelizmente não. Na realidade, a mineração de texto é uma tarefa complicada e mesmo muitas coisas aparentemente simples não podem ser feitas de forma satisfatória.

Por exemplo, assuma a tarefa de adivinhar o endereço correto. A Figura 8.6 mostra como é difícil retornar o resultado exato com certeza e como o Google Maps solicita mais informações ao procurar por "Springfield". Nesse caso, um ser humano não teria se saído melhor sem contexto adicional, mas essa ambigüidade é um dos muitos problemas que você enfrenta em um aplicativo de mineração de texto.

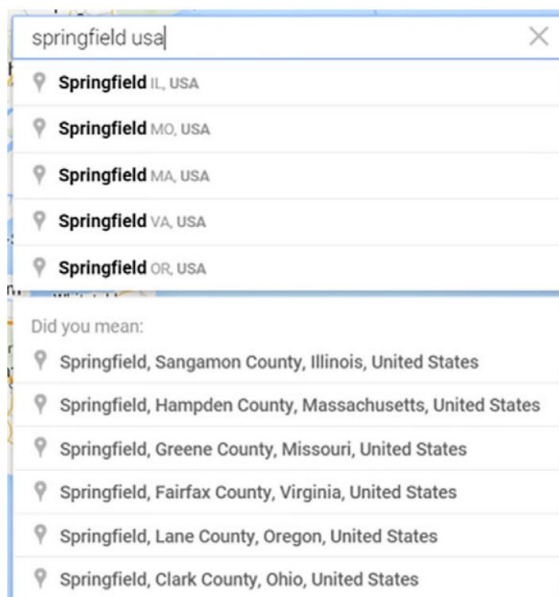


Figura 8.6 O Google Maps pede mais contexto devido à ambiguidade da consulta “Springfield”.

Outro problema são os *erros ortográficos* e as *diferentes formas ortográficas (corretas)* de uma palavra. Pegar as três referências a seguir a Nova York: “NY”, “Nova York” e “Nova York”. Para humano, é fácil perceber que todos se referem à cidade de Nova York. Por causa da maneira como nosso cérebro interpreta o texto, entender o texto com erros ortográficos é algo natural para nós; as pessoas podem nem notá-los. Mas para um computador estas são strings não relacionadas a menos que usemos algoritmos para dizer que eles estão se referindo à mesma entidade. Relacionado problemas são sinônimos e o uso de pronomes. Tente designar a pessoa certa para o pronome “ela” nas próximas frases: “John deu flores aos pais de Marleen quando ele conheceu seus pais pela primeira vez. Ela ficou muito feliz com esse gesto.” Bastante fácil, certo? Não para um computador.

Podemos resolver muitos problemas semelhantes com facilidade, mas muitas vezes eles são difíceis para um máquina. Podemos treinar algoritmos que funcionam bem em um problema específico em um escopo bem definido, mas algoritmos mais gerais que funcionam em todos os casos são outra fera completamente. Por exemplo, podemos ensinar um computador a reconhecer e recuperar dados dos EUA números de contas do texto, mas isso não generaliza bem para números de contas de outros países.

Os algoritmos de linguagem também são sensíveis ao contexto em que a linguagem é usada, mesmo se a linguagem em si permanecer a mesma. Modelos ingleses não funcionarão para árabe e vice-versa vice-versa, mas mesmo se continuarmos com o inglês – um algoritmo treinado para dados do Twitter provavelmente não ter um bom desempenho em textos jurídicos. Vamos ter isso em mente quando passarmos para o estudo de caso do capítulo: não existe uma solução perfeita e que sirva para todos em mineração de texto.

8.2 Técnicas de mineração de texto

Durante nosso próximo estudo de caso, abordaremos o problema da *classificação de textos*: classificar automaticamente textos não categorizados em categorias específicas. Para passar dos dados textuais brutos ao nosso destino final, precisaremos de algumas técnicas de mineração de dados que requerem informações básicas para que possamos usá-los de forma eficaz. O primeiro conceito importante na mineração de texto é o “saco de palavras”.

8.2.1 Saco de palavras

Para construir nosso modelo de classificação, usaremos a abordagem do saco de palavras. *Saco de palavras* é a forma mais simples de estruturar dados textuais: todo documento é transformado em um vetor de palavras. Se uma determinada palavra estiver presente no vetor ela será rotulada como “Verdadeira”; os outros são rotulados como “Falsos”. A Figura 8.7 mostra um exemplo simplificado disso, caso existam apenas dois documentos: um sobre o programa de televisão *Game of Thrones* e outro sobre ciência de dados. Os dois vetores de palavras juntos formam a *matriz de termos do documento*. A matriz de termos do documento contém uma coluna para cada termo e uma linha para cada documento. Os valores são seus para decidir. Neste capítulo usaremos binário: o termo está presente? Verdadeiro ou falso.

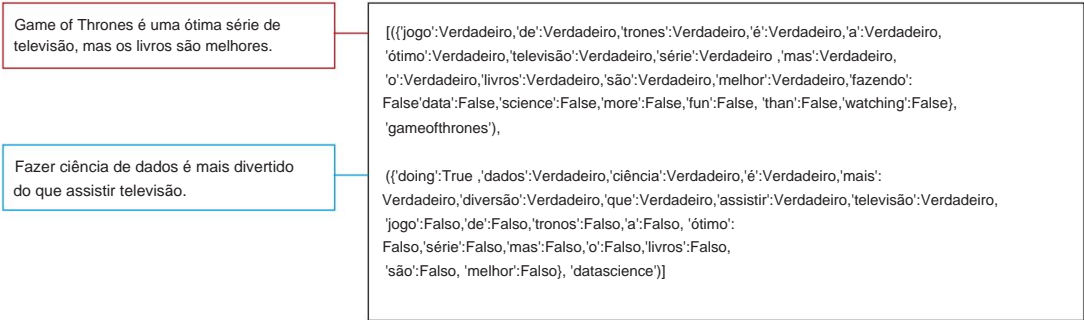


Figura 8.7 Um texto é transformado em um conjunto de palavras rotulando cada palavra (termo) com “Verdadeiro” se estiver presente no documento e “Falso” se não estiver.

O exemplo da figura 8.7 dá uma ideia dos dados estruturados que precisaremos para iniciar a análise de texto, mas é bastante simplificado: nem uma única palavra foi filtrada e nenhuma lematização (falaremos disso mais tarde) foi aplicada. Um grande corpus pode ter milhares de palavras únicas. Se todos tiverem que ser rotulados assim, sem qualquer filtragem, é fácil perceber que poderemos acabar com um grande volume de dados. *O pacote de palavras codificadas em binário*, como mostrado na figura 8.7, é apenas uma maneira de estruturar os dados; existem outras técnicas.

Frequência do Prazo — Frequência Inversa do Documento (TF-IDF)

Uma fórmula bem conhecida para preencher a matriz de prazos de documentos é TF-IDF ou Frequência de Prazos multiplicado pela frequência inversa do documento. Conjunto binário de palavras atribui Verdadeiro ou Falso (o termo existe ou não), enquanto as frequências simples contam o número de vezes que o termo ocorreu. TF-IDF é um pouco mais complicado e leva em conta quantas vezes um termo ocorreu no documento (TF). TF pode ser uma contagem de termos simples, uma contagem binária (Verdadeiro ou Falso) ou uma contagem de termos em escala logarítmica. Depende do que funciona melhor para você. Caso TF seja uma frequência de termo, a fórmula de TF é a seguinte:

$$TF = \text{pés}, d$$

TF é a frequência (f) do termo (t) no documento (d).

Mas o TF-IDF também leva em consideração todos os outros documentos por causa do Inverso Frequência do documento. O IDF dá uma ideia de quão comum a palavra é em todo o corpus: quanto maior a frequência do documento, mais comuns e mais comuns são as palavras são menos informativos. Por exemplo, as palavras “um” ou “o” provavelmente não fornecem informações específicas sobre um texto. A fórmula do IDF com escala logarítmica é a mais forma comumente usada de IDF:

$$IDF = \log(N / \{d \text{ } \ddot{y} \ddot{y} D : t \text{ } \ddot{y} \ddot{y} d\})$$

sendo N o número total de documentos no corpus, e o $\{d \text{ } \ddot{y} \ddot{y} D : t \text{ } \ddot{y} \ddot{y} d\}$ sendo a quantidade de documentos (d) em que aparece o termo (t).

A pontuação do TF-IDF diz o seguinte sobre um termo: quão importante é esta palavra para distinguir este documento dos demais do corpus? A fórmula do TF-IDF é assim

$$\frac{1F}{FDI} \quad f = t, d \quad \ddot{y} \log \ddot{y} N d \quad P : td \text{ } \ddot{y} \ddot{y} \quad \bigg| \quad \ddot{y}$$

Não usaremos TF-IDF, mas ao definir suas próximas etapas na mineração de texto, isso deve ser uma das primeiras coisas que você encontrará. TF-IDF também é o que foi usado pelo Elasticsearch nos bastidores no capítulo 6. É uma boa opção se você quiser usar o TF-IDF para análise de texto; deixe a mineração de texto para software especializado, como SOLR ou Elastic-search, e pegue a matriz de documento/termo para análise de texto a partir daí.

Antes de chegar ao verdadeiro conteúdo das palavras, muitas outras etapas de manipulação de dados são realizadas lugar:

• *Tokenização* — O texto é cortado em pedaços chamados “tokens” ou “termos”. Esses tokens são a unidade de informação mais básica que você usará em seu modelo. O termos geralmente são palavras, mas isso não é uma necessidade. Frases inteiras podem ser usadas para análise. Usaremos *unigramas*: termos que consistem em uma palavra. Muitas vezes, porém, é útil incluir *bigramas* (duas palavras por ficha) ou *trigramas* (três palavras por token) para capturar significado extra e aumentar o desempenho de seus modelos.

Porém, isso tem um custo, porque você está construindo vetores de termos maiores incluindo bigramas e/ou trigramas na equação.

• *Interromper a filtragem de palavras* — *todo* idioma vem com palavras que têm pouco valor no texto análises porque são usadas com frequência. NLTK vem com uma pequena lista de inglês pare palavras que podemos filtrar. Se o texto for transformado em palavras, muitas vezes faz sentido para livrar o vetor de palavras dessas palavras irrelevantes com pouca informação.

• *Minúsculas*—*Palavras* com letras maiúsculas aparecem no início de uma frase, outros porque são nomes próprios ou adjetivos. Não ganhamos nenhum valor agregado fazendo essa distinção em nossa matriz de termos, então todos os termos serão definidos em letras minúsculas.

Outra técnica de preparação de dados é a *derivação*. Este requer mais elaboração.

8.2.2 Lematização e lematização A lematização é

o processo de trazer as palavras de volta à sua forma raiz; assim você acaba com menor variação nos dados. Isto faz sentido se as palavras têm significados semelhantes, mas são escrito de forma diferente porque, por exemplo, um está no plural. Tentativas de stemização unificar cortando partes da palavra. Por exemplo, “aviões” e “avião” ambos tornar-se “avião”.

Outra técnica, chamada *lematização*, tem esse mesmo objetivo, mas o faz de uma forma mais forma gramaticalmente sensível. Por exemplo, embora tanto a lematização quanto a lematização reduziria “carros” a “carro”, a lematização também pode trazer de volta verbos conjugados para suas formas não conjugadas, como “são” para “ser”. Qual você usa depende do seu caso, e a lematização lucra fortemente com a marcação de POS (marcação de parte da fala). *POS Tagging* é o processo de atribuir um rótulo gramatical a cada parte de uma frase. Você provavelmente fez isso manualmente na escola como exercício de linguagem. Veja a frase “*Game of Thrones* é uma série de televisão”. Se aplicarmos POS Tagging nele, obteremos

{("jogo": "NN"), ("de": "IN"), ("tronos": "NNS"), ("é": "VBZ"), ("a": "DT"), ("televisão": "NN"), ("série": "NN")}

NN é um substantivo, IN é uma preposição, NNS é um substantivo no plural, VBZ é uma terceira pessoa verbo no singular e DT é um determinante. A Tabela 8.1 apresenta a lista completa.

Tabela 8.1 Uma lista de todas as tags POS

Marcação	Significado	Marcação	Significado
CC	Conjunção coordenativa	CD	Número cardinal
DT	Determinador	EX	Existencial
AA	Palavra estrangeira	EM	Preposição ou conjunção subordinada
JJ	Adjetivo	JJR	Adjetivo, comparativo
JJS	Adjetivo, superlativo	LS	Marcador de item de lista
Módico	Modal	NN	Substantivo, singular ou massivo

Tabela 8.1 Uma lista de todas as tags POS (continuação)

Marcação	Significado	Marcação	Significado
NNS	Substantivo, plural	NNP	Nome próprio, singular
NNPS	Nome próprio, plural	PDT	Predeterminador
PDV	Final possessivo	PRP	Pronome pessoal
PRP\$	Pronome possessivo	RB	Advérbio
RBR	Advérbio, comparativo	Advérbio	RBS, superlativo
PR	Partícula	Símbolo	\$IM
UH	Interjeição	VB	Verbo, forma básica
DTV	Verbo, pretérito	Verbo	VBG, gerúndio ou particípio presente
VDN	Verbo, particípio passado	VBP	Verbo, presente não 3ª pessoa do singular
VBZ	Verbo, 3ª pessoa do singular presente	Determinador	WDT Wh
WP	Pronome Wh	WP\$	Possessivo wh-pronome
WRB	Wh-advérbio		

POS Tagging é um caso de uso de tokenização de frases em vez de tokenização de palavras. Depois que a marcação do PDV for concluída, você ainda poderá prosseguir com a tokenização de palavras, mas um POS Tagger requer frases inteiras. Combinar marcação de PDV e lematização é provavelmente fornecerá dados mais limpos do que usar apenas um lematizador. Por uma questão de simplicidade vamos atenha-se à derivação no estudo de caso, mas considere esta uma oportunidade para elaborar o exercício.

Agora sabemos as coisas mais importantes que usaremos para fazer a limpeza de dados e manipulação (mineração de texto). Para nossa análise de texto, vamos adicionar o classificador de árvore de decisão ao nosso repertório.

8.2.3 Classificador de árvore de decisão

A parte de análise de dados do nosso estudo de caso também será mantida simples. Vamos testar um Ingênuo Classificador Bayes e um classificador de árvore de decisão. Como visto no capítulo 3, o Naïve Bayes classificador é chamado assim porque considera cada variável de entrada independente de todos os outros, o que é ingênuo, especialmente na mineração de texto. Tomemos os exemplos simples de “ciência de dados”, “análise de dados” ou “jogo dos tronos”. Se cortarmos nossos dados em unigramas obtemos as seguintes variáveis separadas (se ignorarmos a lematização e coisas assim): “dados”, “ciência”, “análise”, “jogo”, “de” e “tronos”. Obviamente os links serão perdidos. Isto pode, em por sua vez, ser superado pela criação de bigramas (ciência de dados, análise de dados) e trigramas (A Guerra dos Tronos).

O classificador de árvore de decisão, entretanto, não considera as variáveis independentes umas das outras e cria ativamente *variáveis de interação e intervalos*. Uma *interação*

variável é uma variável que combina outras variáveis. Por exemplo, “dados” e “ciência” podem ser bons preditores por si só, mas provavelmente os dois que ocorrem simultaneamente no mesmo texto podem ter seu próprio valor. Um balde é um pouco o oposto. Em vez de combinar duas variáveis, uma variável é dividida em várias novas. Isso faz sentido para variáveis numéricas. A Figura 8.8 mostra a aparência de uma árvore de decisão e onde você pode encontrar interação e agrupamento.

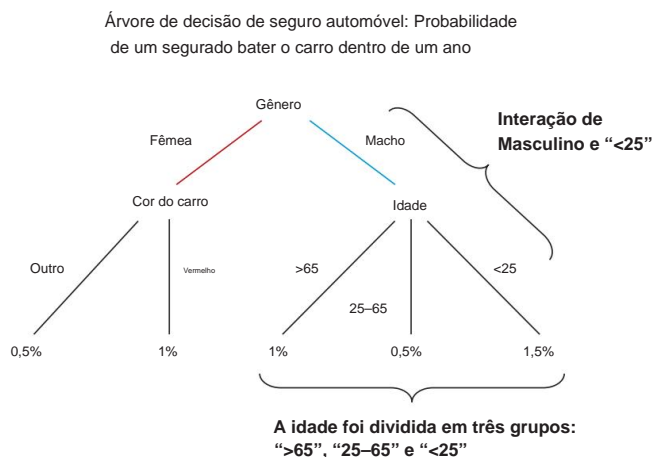


Figura 8.8 Modelo fictício de árvore de decisão. Uma árvore de decisão cria automaticamente buckets e supõe interações entre variáveis de entrada.

Enquanto Naïve Bayes supõe a independência de todas as variáveis de entrada, uma árvore de decisão é construída sobre a suposição de interdependência. Mas como ele constrói essa estrutura? Uma árvore de decisão tem alguns critérios possíveis que pode usar para dividir em ramos e decidir quais variáveis são mais importantes (estão mais próximas da raiz da árvore) do que outras. O que usaremos no classificador de árvore de decisão NLTK é “informações

ganho.” Para compreender o ganho de informação, primeiro precisamos olhar para a entropia. A *entropia* é uma medida de imprevisibilidade ou caos. Um exemplo simples seria o sexo de um bebê. Quando uma mulher está grávida, o sexo do feto pode ser masculino ou feminino, mas não sabemos qual é. Se você adivinhasse, teria 50% de chance de acertar (mais ou menos, porque a distribuição de gênero não é 100% uniforme). Porém, durante a gravidez você tem a oportunidade de fazer um ultrassom para determinar o sexo do feto. Um ultrassom nunca é 100% conclusivo, mas quanto mais avançado o desenvolvimento fetal, mais preciso ele se torna. Esse ganho de precisão, ou *ganho de informação*, ocorre porque a incerteza ou a entropia diminuem. Digamos que um ultrassom com 12 semanas de gravidez tenha 90% de precisão na determinação do sexo do bebê. Ainda existe uma incerteza de 10%, mas o ultrassom reduziu a incerteza

Probabilidade de feto identificado como
mulher – ultrassonografia com 12 semanas

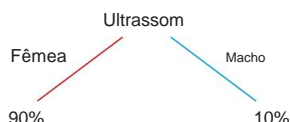


Figura 8.9 Árvore de decisão com uma variável: a conclusão do médico ao observar um ultrassom durante a gravidez. Qual é a probabilidade do feto ser do sexo feminino?

de 50% a 10%. Esse é um discriminador muito bom. Uma árvore de decisão segue isto mesmo princípio, conforme mostrado na figura 8.9.

Se outro teste de gênero tiver mais poder preditivo, poderá tornar-se a raiz do árvore com o exame de ultrassom nos galhos, e isso pode continuar até a gente correr fora de variáveis ou observações. Podemos ficar sem observações, porque a cada divisão de ramificação, também dividimos os dados de entrada. Esta é uma grande fraqueza da árvore de decisão, porque no nível da folha a robustez da árvore é interrompida se poucas observações forem esquerda; as árvores de decisão começam a ajustar demais os dados. *Overfitting* permite que o modelo erre aleatoriedade para correlações reais. Para contrariar isto, uma árvore de decisão é *podada*: a sua ramos sem sentido são deixados de fora do modelo final.

Agora que vimos as novas técnicas mais importantes, vamos mergulhar no estudo de caso.

8.3 Estudo de caso: Classificação de postagens do Reddit

Embora a mineração de texto tenha muitas aplicações, no estudo de caso deste capítulo nos concentramos na *classificação de documentos*. Como apontado anteriormente neste capítulo, é exatamente isso que o Google faz quando organiza seus e-mails em categorias ou tenta distinguir spam de e-mails regulares. Ele também é amplamente utilizado por contact centers que processam dúvidas ou reclamações recebidas de clientes: as reclamações por escrito passam primeiro por um filtro de detecção de tópico para que possam ser atribuídas às pessoas corretas para tratamento. Documento a classificação também é um dos recursos obrigatórios dos sistemas de monitoramento de mídias sociais. Os tweets monitorados, postagens em fóruns ou no Facebook, artigos de jornais e muitos outros outros recursos da Internet recebem rótulos de tópico. Dessa forma, eles podem ser reutilizados em relatórios. A *análise de sentimento* é um tipo específico de classificação de texto: é o autor de uma postagem negativo, positivo ou neutro em alguma coisa? Esse “algo” pode ser reconhecido com reconhecimento de entidade.

Neste estudo de caso, nos basearemos em postagens do Reddit, site também conhecido como a autopromocionada “primeira página da internet”, e tentaremos treinar um modelo capaz de distinguir se alguém está falando sobre “ciência de dados”. ou “jogo dos tronos”.

O resultado final pode ser uma apresentação do nosso modelo ou uma aplicação interativa completa. No capítulo 9 focaremos na construção de aplicativos para o usuário final, então por enquanto nos limitaremos a apresentar nosso modelo de classificação.

Para atingir nosso objetivo precisaremos de toda a ajuda e ferramentas que pudermos obter, e isso acontece Python está mais uma vez pronto para fornecê-los.

8.3.1 Conheça o kit de ferramentas de linguagem natural

Python pode não ser a linguagem com execução mais eficiente do mundo, mas tem uma pacote maduro para mineração de texto e processamento de linguagem: o *Natural Language Toolkit* (NLTK). NLTK é uma coleção de algoritmos, funções e trabalhos anotados que irão orientá-lo nos primeiros passos na mineração de texto e no processamento de linguagem natural. O NLTK também está excelentemente documentado em nltk.org. NLTK, no entanto, não é frequentemente usado para trabalho de nível de produção, como outras bibliotecas, como scikit-learn.

Instalando o NLTK e seus corpora

Instale o NLTK com seu instalador de pacote favorito. Caso você esteja usando o Anaconda, vem instalado com a configuração padrão do Anaconda. Caso contrário, você pode optar por “pip” ou “fácil_instalar”. Feito isso você ainda precisa instalar os modelos e corpora incluído para que seja totalmente funcional. Para isso, execute o seguinte código Python:

```

import nltk
nltk.download()

```

Dependendo da sua instalação, isso lhe dará um pop-up ou mais opções de linha de comando.

A Figura 8.10 mostra a caixa pop-up que você obtém ao emitir o comando `nltk.download()`.

Você pode baixar todos os corpora se quiser, mas neste capítulo usaremos apenas de “punkt” e “stopwords”. Este download será mencionado explicitamente no código que vem com este livro.

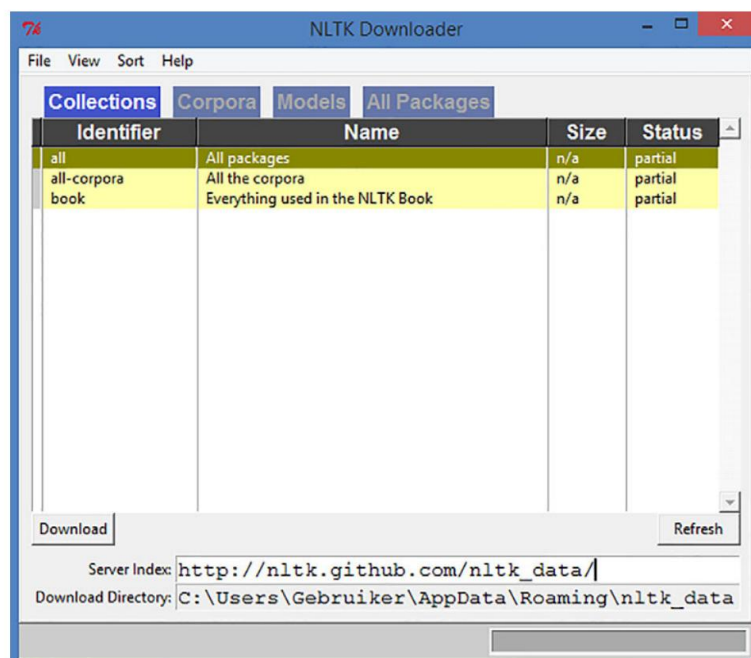


Figura 8.10 Escolha todos os pacotes para concluir totalmente a instalação do NLTK.

Dois arquivos de notebook IPython estão disponíveis para este capítulo:

• *Coleta de dados* – conterà a parte de coleta de dados do estudo de caso deste capítulo. • *Preparação e análise de dados* — Os dados armazenados são submetidos à preparação de dados e depois submetido à análise.

Todo o código do próximo estudo de caso pode ser encontrado nesses dois arquivos na mesma sequência e também pode ser executado como tal. Além disso, dois gráficos interativos estão disponíveis para download:

• *forceGraph.html* — Representa os 20 principais recursos do nosso modelo Naïve Bayes • *Sunburst.html* — Representa os quatro principais ramos do nosso modelo de árvore de decisão

Para abrir essas duas páginas HTML, é necessário um servidor HTTP, que você pode obter usando Python e uma janela de comando:

• Abra uma janela de comando (Linux, Windows, o que você quiser). • Vá para a pasta que contém os arquivos HTML e seus arquivos de dados JSON: `decisionTreeData.json` para o diagrama sunburst e `NaiveBayesData.json` para o gráfico de força. É importante que os arquivos HTML permaneçam no mesmo local que seus arquivos de dados ou você terá que alterar o JavaScript no arquivo HTML. • Crie um servidor HTTP Python com o seguinte

comando: `python -m Simple-Servidor HTTP 8000`

• Abra um navegador e vá para `localhost:8000`; aqui você pode selecionar os arquivos HTML, como mostrado na figura 8.11.

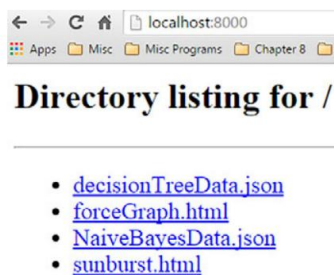


Figura 8.11 Servidor HTTP Python servindo a saída deste capítulo

Os pacotes Python que usaremos neste capítulo:

• NLTK—Para mineração de texto • PRAW—Permite baixar postagens do Reddit • SQLite3 —Permite armazenar dados no formato SQLite • Matplotlib—Uma biblioteca de plotagem para visualizar dados

Certifique-se de instalar todas as bibliotecas e corpora necessários antes de prosseguir. Antes de mergulharmos na ação, entretanto, vamos dar uma olhada nas etapas que seguiremos para atingir nosso objetivo de criar um modelo de classificação de tópicos.

8.3.2 Visão geral do processo de ciência de dados e etapa 1: O objetivo da pesquisa

Para resolver este exercício de mineração de texto, utilizaremos mais uma vez o processo de ciência de dados. A Figura 8.12 mostra o processo de ciência de dados aplicado ao nosso caso de classificação do Reddit.

Nem todos os elementos representados na figura 8.12 podem fazer sentido neste momento, e o resto do capítulo é dedicado a resolver isso na prática enquanto trabalhamos em direção ao nosso objetivo de pesquisa: criar um modelo de classificação capaz de distinguir postagens sobre “ciência de dados” de postagens sobre “Game of Thrones”. Sem mais delongas, vamos buscar nossos dados.

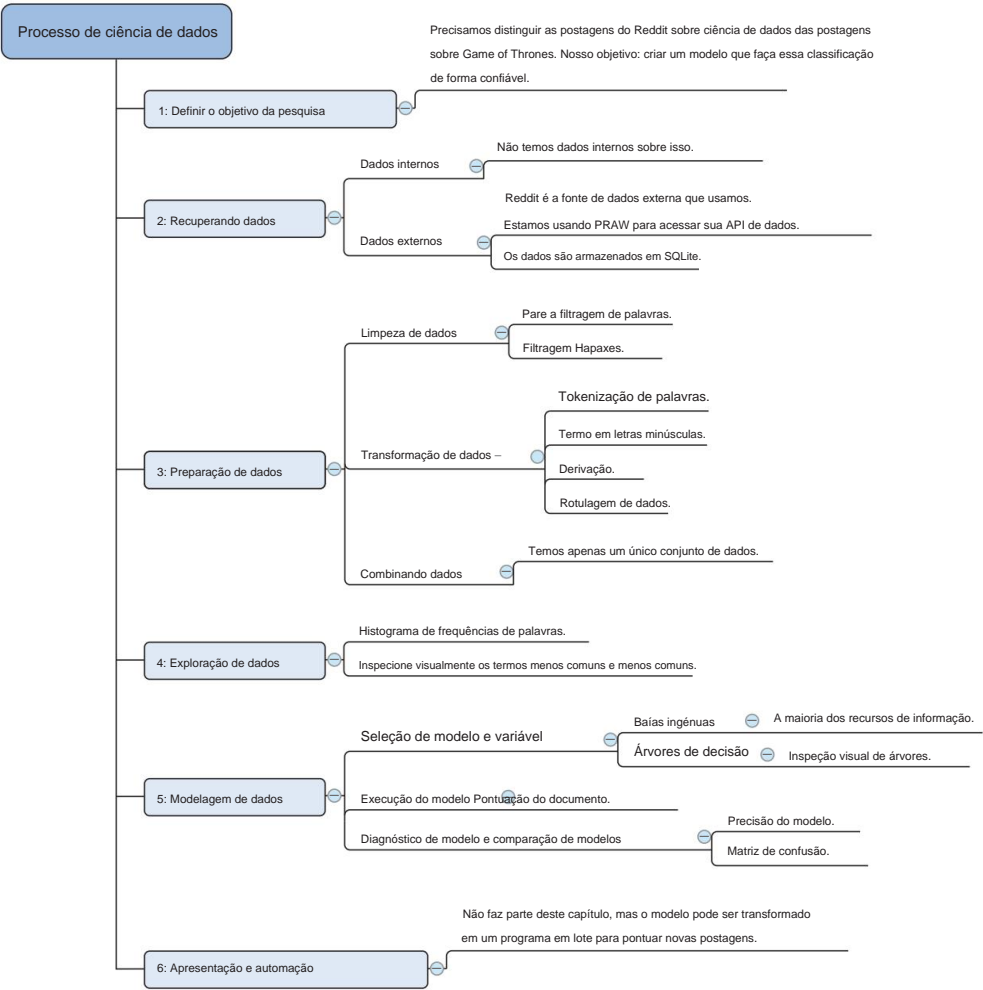


Figura 8.12 Visão geral do processo de ciência de dados aplicado ao estudo de caso de classificação de tópicos do Reddit

8.3.3 Etapa 2: Recuperação de dados

Usaremos dados do Reddit para este caso e, para aqueles que não estão familiarizados com o Reddit, reserve um tempo para se familiarizar com seus conceitos em www.reddit.com.

O Reddit se autodenomina “a primeira página da internet” porque os usuários podem postar coisas eles acham interessante e/ou encontrados em algum lugar na internet, e apenas essas coisas considerados interessantes por muitas pessoas são apresentados como “populares” em sua página inicial. Você poderia dizer que o Reddit dá uma visão geral das tendências na internet. Qualquer usuário pode postar em uma categoria predefinida chamada “subreddit”. Quando uma postagem é feita, outros usuários podem comentar sobre ele e podem votar positivamente se gostarem do conteúdo ou votar negativamente se não gostarem. Como uma postagem sempre faz parte de um subreddit, temos esses metadados à nossa disposição quando nos conectamos à API do Reddit para obter nossos dados. Estamos efetivamente buscando dados rotulados porque assumiremos que uma postagem no subreddit “gameofthrones” tem algo a ver com “gameofthrones”.

Para obter nossos dados, usamos a biblioteca oficial da API Python do Reddit chamada PRAW. Assim que obtivermos os dados que precisamos, iremos armazená-los em um arquivo leve, semelhante a um banco de dados chamado SQLite. SQLite é ideal para armazenar pequenas quantidades de dados porque não requer qualquer configuração para usar e responderá a consultas SQL como qualquer relacional regular banco de dados faz. Qualquer outro meio de armazenamento de dados serve; se você preferir bancos de dados Oracle ou Post-gres, Python possui uma excelente biblioteca para interagir com eles sem a necessidade para escrever SQL. SQLAlchemy também funcionará para arquivos SQLite. A Figura 8.13 mostra os dados etapa de recuperação dentro do processo de ciência de dados.

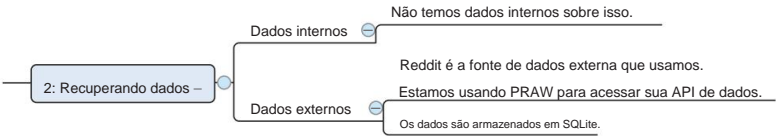


Figura 8.13 A etapa de recuperação de dados do processo de ciência de dados para um caso de classificação de tópicos do Reddit

Abra seu interpretador Python favorito; é hora de agir, como mostra a listagem 8.1. Primeiro precisamos coletar nossos dados do site Reddit. Se você ainda não o fez, use `pip install praw` ou `conda install praw` (Anaconda) antes de executar o seguinte script.

NOTA O código para a etapa 2 também pode ser encontrado no arquivo IPython “Capítulo 8 coleção de dados.” Está disponível na seção de download deste livro.

buscar a qualquer solicitação, embora possamos solicitar mais posteriormente, quando as pessoas tiverem

postou coisas novas. Na verdade, podemos executar a solicitação da API periodicamente e coletar dados ao longo do tempo. Embora a qualquer momento você esteja limitado a mil postagens, nada impede impedi-lo de aumentar seu próprio banco de dados ao longo dos meses. Vale a pena notar o O script a seguir pode levar cerca de uma hora para ser concluído. Se você não sente vontade de esperar, sintase à vontade para prosseguir e usar o arquivo SQLite para download. Além disso, se você executá-lo agora, você provavelmente não obterá exatamente a mesma saída de quando foi executado pela primeira vez para criar a saída mostrado neste capítulo.

Vejamos nossa função de recuperação de dados, conforme mostrado na listagem a seguir.

Listagem 8.2 Recuperação e armazenamento de dados do Reddit em SQLite

Campos específicos do tópico são anexados à lista. Usamos apenas o título e o texto ao longo do exercício, mas o ID do tópico seria útil para construir seu próprio banco de dados (maior) de tópicos.

```
def prawGetData(limite, subredditName):
    tópicos = r.get_subreddit(subredditName).get_hot(limit=limite)
    commentInsert = []

    tópicoInsert = []
    tópicoNBR = 1
    para tópico em tópicos:
        if (float(tópicoNBR)/limite)*100 em xrange(1,100):
            imprimir '***** TÓPICO:' + str(topic.id) + str((float(tópicoNBR)/limite)*100)
        + ' *****COMPLETO: '
        + '% ****'

        tópicoNBR += 1
        tentar:
            topicInsert.append((topic.title,topic.selftext,topic.id,
                               nome do subreddit))

        exceto:
            passar

        tentar:
            para comentar em topic.comments:
                commentInsert.append((comment.body,comment.id,
                                       topic.title,topic.selftext,topic.id,subredditName))
            exceto:
                passar
```

De subreddits, obtenha os 1.000 tópicos mais populares (no nosso caso).

Esta parte é uma impressão informativa e não é necessária para que o código funcione. Ele apenas informa sobre o andamento do download.

Anexe comentários a uma lista. Eles não são usados no exercício, mas agora você os tem para experimentação.

Insira todos os tópicos em SQLite base de dados.

```
passar
imprimir '*****'
imprima 'INSERINDO DADOS NO SQLITE'
c.executemany("INSERT INTO tópicos VALUES (?,?,,?,?)", topicInsert)
imprimir 'TÓPICOS INSERIDOS'
c.executemany("INSERT INTO comentários VALUES (?,?,,?,?,?)", commentInsert)
imprimir 'COMENTÁRIOS INSERIDOS'
conexão.commit()
```

```
para assunto em subreddits:
    prawGetData(limit=limite, subredditName=assunto)
```

Confirmar alterações (inserções de dados) no banco de dados. Sem o commit, nenhum dado será inserido.

Insira tudo comentários em Banco de dados SQLite.

A função é executada para todos os subreddits que especificado anteriormente.

A função `prawGetData()` recupera os tópicos “mais quentes” em seu subreddit, anexa isso para um array e, em seguida, obtém todos os comentários relacionados. Isso continua até que mil tópicos sejam alcançados ou não existam mais tópicos para buscar e tudo seja armazenado em o banco de dados SQLite. As declarações impressas existem para informá-lo sobre seu progresso para reunir mil tópicos. Tudo o que nos resta fazer é executar a função para cada subreddit.

Se você quiser que esta análise incorpore mais de dois subreddits, isso é uma questão de adicionar uma categoria extra à matriz de subreddits.

Com os dados coletados, estamos prontos para prosseguir com a preparação dos dados.

8.3.4 Etapa 3: Preparação dos dados

Como sempre, a preparação dos dados é a etapa mais importante para obter resultados corretos. Para mineração de texto isso é ainda mais verdadeiro, já que nem começamos com dados estruturados.

O próximo código está disponível online como arquivo IPython “Capítulo 8 preparação de dados e análise.” Vamos começar importando as bibliotecas necessárias e preparando o SQLite banco de dados, conforme mostrado na listagem a seguir.

Listagem 8.3 Mineração de texto, bibliotecas, dependências de corpora e conexão de banco de dados SQLite

```
importar sqlite3
importar nltk
importar matplotlib.pyplot como plt
de coleções importar OrderedDict importar aleatoriamente
```

**Importe todas
as
bibliotecas necessárias**

```
nltk.download('ponto')
nltk.download('palavras irrelevantes')
```

**Baixe os corpora que
utilizamos**

```
conn = sqlite3.connect('reddit.db') c = conn.cursor()
```

**Faça uma conexão com o banco de dados
SQLite que contém nossos dados do Reddit**

Caso você ainda não tenha baixado o corpus completo do NLTK, faremos agora o download do parte dele usaremos. Não se preocupe se você já baixou, o script irá detectar se seu corpora está atualizado.

Nossos dados ainda estão armazenados no arquivo Reddit SQLite, então vamos criar uma conexão com eles.

Mesmo antes de explorarmos nossos dados, sabemos de pelo menos duas coisas que precisamos fazer para limpe os dados: pare a filtragem de palavras e letras minúsculas.

Uma função geral de filtro de palavras nos ajudará a filtrar as partes impuras. Vamos criar um na listagem a seguir.

Listagem 8.4 Filtragem de palavras e funções de letras minúsculas

```
def wordFilter(excluído,wordrow):
    filtrado = [palavra por palavra na linha de palavras se a palavra não estiver excluída]
    retornar filtrado

stopwords = nltk.corpus.stopwords.words('english') def lowerCaseArray(wordrow):
lowercased = [word.lower() para palavra em
               wordrow]
               retornar em minúsculas
```

A função **wordFilter()** removerá um termo de uma série de termos

A variável de palavra de parada contém palavras de parada em inglês por padrão presentes no NLTK

Função **lowerCaseArray()** transforma qualquer termo em sua versão em minúsculas

As palavras irrelevantes em inglês serão as primeiras a sair de nossos dados. O código a seguir nos fornecerá estas palavras de parada:

```
stopwords = nltk.corpus.stopwords.words('inglês')
imprimir palavras irrelevantes
```

A Figura 8.14 mostra a lista de palavras irrelevantes em inglês no NLTK.

```
stopwords = nltk.corpus.stopwords.words('english')
print stopwords
```

[u'i', u'me', u'my', u'myself', u'we', u'our', u'ours', u'ourselves', u'you', u'your', u'yours', u'yourself', u'yourselves', u'he', u'him', u'his', u'himself', u'she', u'her', u'hers', u'herself', u'it', u'its', u'itself', u'they', u'them', u'their', u'theirs', u'themselves', u'what', u'which', u'who', u'whom', u'this', u'that', u'these', u'those', u'am', u'is', u'are', u'was', u'were', u'be', u'been', u'being', u'have', u'has', u'had', u'having', u'do', u'does', u'did', u'doing', u'a', u'an', u'the', u'and', u'but', u'if', u'or', u'because', u'as', u'until', u'while', u'of', u'at', u'by', u'for', u'with', u'about', u'against', u'between', u'into', u'through', u'during', u'before', u'after', u'above', u'below', u'to', u'from', u'up', u'down', u'in', u'out', u'on', u'off', u'over', u'under', u'again', u'further', u'then', u'once', u'here', u'there', u'when', u'where', u'why', u'how', u'all', u'any', u'both', u'each', u'few', u'more', u'most', u'other', u'some', u'such', u'no', u'nor', u'not', u'only', u'own', u'same', u'so', u'than', u'too', u'very', u's', u't', u'can', u'will', u'just', u'don', u'should', u'now']

Figura 8.14 Lista de palavras irrelevantes em inglês no NLTK

Com todos os componentes necessários instalados, vamos dar uma olhada em nossa primeira função de processamento de dados na listagem a seguir.

Listagem 8.5 Primeira função e execução de preparação de dados

Usaremos `data['all_words']` para exploração de dados.

```
def processamento_dados(sql):
    c.execute(sql)
    dados = {'wordMatrix': [], 'all_words': []}
    linha = c.fetchone()
    enquanto a linha não é nenhuma:
        wordrow = nltk.tokenize.word_tokenize(row[0]+" "+row[1]) wordrow_lowercased =
        lowerCaseArray(wordrow)
        wordrow_nostopwords = wordFilter(stopwords, wordrow_lowercased)
        dados['all_words'].extend(wordrow_nostopwords)
        dados['wordMatrix'].append(wordrow_nostopwords) linha = c.fetchone()
```

Crie um ponteiro
para dados AWLite.

Buscar dados
linha por linha.

`row[0]` é o
título, `row[1]` é o
texto do tópico;
nós os transformamos
em um único
blob de texto.

dados de retorno

```
subreddits = ['datascience', 'gameofthrones'] dados = {}

para assunto em subreddits:
    dados[assunto] = processamento_de_dados(sql="SELECT
        topicTitle, topicText, topicCategory FROM tópicos
        WHERE topicCategory = '"+assunto+"")
```

`data['wordMatrix']` é uma matriz composta
por vetores de palavras; 1 vetor por
documento.

Obtenha um novo
documento do banco de dados SQLite.

Nossos subreddits
conforme definidos anteriormente.

Chame a função de
processamento de dados
para cada subreddit.

Nossa função `data_processing()` recebe uma instrução SQL e retorna a matriz de termos do documento. Ele faz isso percorrendo os dados, uma entrada (tópico do Reddit) por vez.

tempo e combina o título do tópico e o texto do corpo do tópico em um único vetor de palavras com o uso de tokenização de palavras. Um *tokenizer* é um script de manipulação de texto que corta o texto em peças. Você tem muitas maneiras diferentes de tokenizar um texto: você pode dividi-lo em frases ou palavras, pode dividi-lo por espaço e pontuação, ou pode levar em conta outros caracteres, e assim por diante. Aqui optamos pelo tokenizer de palavras NLTK padrão.

Esta palavra tokenizer é simples; tudo o que faz é dividir o texto em termos se houver um espaço entre as palavras. Em seguida, colocamos o vetor em minúsculas e filtramos as palavras irrelevantes. Observação como a ordem é importante aqui; uma palavra de parada no início de uma frase não seria ser filtrado se primeiro filtrarmos as palavras irrelevantes antes de colocá-las em minúsculas. Por exemplo, em "Eu gosto Game of Thrones", o "I" não estaria em letras minúsculas e, portanto, não seria filtrado fora. Criamos então uma matriz de palavras (matriz termo-documento) e uma lista contendo todos as palavras. Observe como estendemos a lista sem filtrar por duplos; desta forma podemos criar um histograma nas ocorrências de palavras durante a exploração de dados. Vamos executar o função para nossas duas categorias de tópicos.

A Figura 8.15 mostra o primeiro vetor de palavras da categoria "datascience".

```
imprimir dados['datascience']['wordMatrix'][0]
```

```
print data['datascience']['wordMatrix'][0]

[u'data', u'science', u'freelancing', u'"m"', u'currently', u'master
s', u'program', u'studying', u'business', u'analytics', u'"m"', u'try
ing', u'get', u'data', u'freelancing', u'.', u'"m"', u'still', u'lear
ning', u'skill', u'set', u'typically', u'see', u'right', u'"m"', u'fa
irly', u'proficient', u'sql', u'know', u'bit', u'r.', u'freelancer
s', u'find', u'jobs', u'?']
```

Figura 8.15 O primeiro vetor de palavras da categoria “datascience” após a primeira tentativa de processamento de dados

Isso certamente parece poluído: as pontuações são mantidas como termos separados e várias palavras nem foram divididos. Uma exploração mais aprofundada dos dados deverá esclarecer algumas coisas para nós.

8.3.5 Etapa 4: Exploração de dados

Agora temos todos os nossos termos separados, mas o tamanho dos dados nos impede de ter uma boa noção se está limpo o suficiente para uso real. Ao olhar para um único vetor, já identificamos alguns problemas: várias palavras não foram divididas corretamente e o vetor contém muitos termos de um único caractere. Termos de caractere único podem ser bons diferenciadores de tópicos em certos casos. Por exemplo, um texto económico conterá mais sinais \$, £ e ¢ do que um texto médico. Mas na maioria dos casos estes termos de um caractere são inúteis. Primeiro, vamos dar uma olhada na distribuição de frequência de nossos termos.

```
wordfreqs_cat1 = nltk.FreqDist(dados['datascience']['all_words'])
plt.hist(wordfreqs_cat1.values(), bins = intervalo(10))
plt.show()
wordfreqs_cat2 = nltk.FreqDist(dados['gameofthrones']['all_words'])
plt.hist(wordfreqs_cat2.values(), bins = intervalo(20))
plt.show()
```

Ao desenhar um histograma da distribuição de frequência (figura 8.16) notamos rapidamente que a maior parte dos nossos termos ocorre apenas em um único documento.

Termos de ocorrência única como esses são chamados de *hapaxes* e, em termos de modelo, são inútil porque uma única ocorrência de um recurso nunca é suficiente para construir um confiável modelo. Esta é uma boa notícia para nós; eliminar esses hapaxes diminuirá significativamente nossos dados sem prejudicar nosso modelo eventual. Vejamos alguns desses single-terms de ocorrência.

```
imprimir wordfreqs_cat1.hapaxes()
imprimir wordfreqs_cat2.hapaxes()
```

Os termos que vemos na figura 8.17 fazem sentido e, se tivéssemos mais dados, provavelmente ocorreriam mais frequentemente.

```
imprimir wordfreqs_cat1.hapaxes()
imprimir wordfreqs_cat2.hapaxes()
```

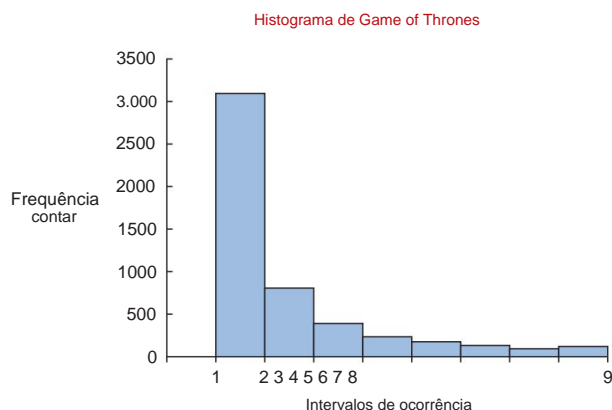
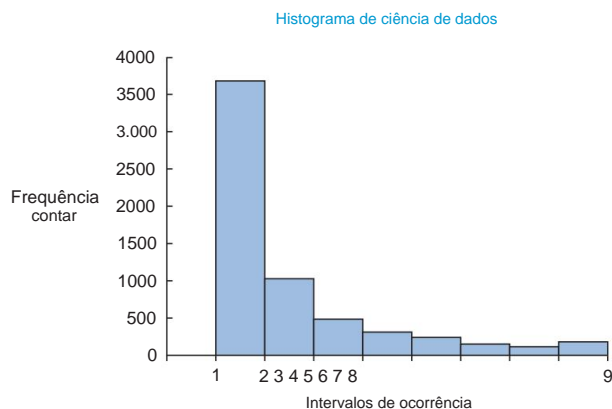


Figura 8.16 Este histograma de frequências de termos mostra que as matrizes de termos de “ciência de dados” e “jogo dos tronos” têm mais de 3.000 termos que ocorrer uma vez.

Least frequent terms within data science posts

```
print wordfreqs_cat1.hapaxes()
```

```
[u'post-grad', u'marching', u'cytoscape', u'wizardry', u''pure", u'i
mmature', u'socrata', u'filenotfoundexception', u'side-by-side', u'b
ringing', u'non-experienced', u'zestimate', u'formatting*', u'sustai
```

Least frequent terms within Game of Thrones posts

```
print wordfreqs_cat2.hapaxes()
```

```
[u'hordes', u'woods', u'comically', u'pack', u'seventy-seven', u''co
ntext", u'shaving', u'kennels', u'differently', u'screaming', u'her-
', u'complainers', u'sailed', u'contributed', u'payoff', u'hallucina
```

Figura 8.17 Termos de ocorrência única “Ciência de dados” e “Game of Thrones” (hapaxes)

Muitos desses termos são grafias incorretas de outros úteis, como: Jaimie é

Jaime (Lannister), Milisandre seria Melisandre e assim por diante. Um *jogo decente de*

O dicionário de sinônimos específico dos tronos pode nos ajudar a encontrar e substituir esses erros ortográficos por um algoritmo de pesquisa difusa. Isso prova que a limpeza de dados na mineração de texto pode continuar indefinidamente se você desejar; manter o esforço e o retorno em equilíbrio é crucial aqui.

Vamos agora dar uma olhada nas palavras mais frequentes.

```
imprimir wordfreqs_cat1.most_common(20)
```

```
imprimir wordfreqs_cat2.most_common(20)
```

A Figura 8.18 mostra o resultado da pergunta sobre as 20 palavras mais comuns para cada categoria.

Most frequent words within data science posts

```
print wordfreqs_cat1.most_common(20)
```

```
[('u'.', 2833), ('u',', 2831), ('u'data', 1882), ('u'?', 1190), ('u'scien  
ce', 887), ('u')', 812), ('u'(', 739), ('u''m", 566), ('u':', 548), ('u'w  
ould', 427), ('u''s", 323), ('u'like', 321), ('u"n't", 288), ('u'get', 2  
52), ('u'know', 225), ('u''ve", 213), ('u'scientist', 211), ('u'!', 20  
9), ('u'work', 204), ('u'job', 199)]
```

Most frequent words within Game of Thrones posts

```
print wordfreqs_cat2.most_common(20)
```

```
[('u'.', 2909), ('u',', 2478), ('u'[, 1422), ('u']', 1420), ('u'?', 113  
9), ('u''s", 886), ('u"n't", 494), ('u')', 452), ('u'(', 426), ('u's5', 3  
99), ('u':', 380), ('u'spoilers', 332), ('u'show', 325), ('u'would', 31  
1), ('u''''', 305), ('u''''', 276), ('u'think', 248), ('u'season', 244),  
(u'like', 243), ('u'one', 238)]
```

Figura 8.18 As 20 palavras mais frequentes para as postagens “ciência de dados” e “jogo dos tronos”

Agora, isso parece encorajador: várias palavras comuns parecem específicas para seus tópicos.

Palavras como “dados”, “ciência” e “estação” provavelmente se tornarão bons diferenciadores. Outra coisa importante a notar é a abundância de termos de caractere único como "." e ","; vamos nos livrar disso.

Com esse conhecimento extra, vamos revisar nosso roteiro de preparação de dados.

8.3.6 Etapa 3 revisitada: preparação de dados adaptada

Esta breve exploração de dados já chamou nossa atenção para alguns ajustes óbvios podemos fazer para melhorar nosso texto. Outro importante é a redução dos termos.

Listagem 8.6 O processamento de dados do Reddit revisado após a exploração dos dados

Inicializa lematizador da biblioteca NLTK.

A matriz de palavras irrelevantes define termos a serem removidos/ignorados.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

Busque dados (postagens do reddit)
um por um no banco de dados SQLite.

**Execute a nova
função de processamento
de dados para ambos os subreddits.**

Observe as mudanças desde a última função `data_processing()`. Nosso tokenizador agora é um tokenizador de expressão regular. Expressões regulares não fazem parte deste livro e são muitas vezes considerado difícil de dominar, mas tudo o que este simples faz é cortar o texto em palavras. Para palavras, qualquer combinação alfanumérica é permitida (`\w`), portanto não há mais caracteres especiais ou pontuações. Também aplicamos a palavra lematizador e removemos um lista de palavras de parada extras. E todos os hapaxes são removidos no final porque tudo precisa ser estancado primeiro. Vamos executar nossa preparação de dados novamente.

Se fizéssemos a mesma análise exploratória de antes, veríamos que faz mais sentido, e não temos mais hapaxes.

```
imprimir wordfreqs_cat1.hapaxes()
imprimir wordfreqs_cat2.hapaxes()
```

Vamos pegar novamente as 20 principais palavras de cada categoria (veja a figura 8.19).

Top 20 most common "Data Science" terms after more intense data cleansing

```
wordfreqs_cat1 = nltk.FreqDist(data['datascience']['all_words'])
print wordfreqs_cat1.most_common(20)
```

```
[(u'data', 1971), (u'scienc', 955), (u'would', 418), (u'work', 368), (u'use', 347), (u'program', 343), (u'learn', 342), (u'like', 341), (u'get', 325), (u'scientist', 310), (u'job', 268), (u'cours', 265), (u'look', 257), (u'know', 239), (u'statist', 228), (u'want', 225), (u've', 223), (u'python', 205), (u'year', 204), (u'time', 196)]
```

Top 20 most common "Game of Thrones" terms after more intense data cleansing

```
wordfreqs_cat2 = nltk.FreqDist(data['gameofthrones']['all_words'])
print wordfreqs_cat2.most_common(20)
```

```
[(u's5', 426), (u'spoiler', 374), (u'show', 362), (u'episod', 300), (u'think', 289), (u'would', 287), (u'season', 286), (u'like', 282), (u'book', 271), (u'one', 249), (u'get', 236), (u'sansa', 232), (u'scene', 216), (u'cersei', 213), (u'know', 192), (u'go', 188), (u'king', 183), (u'throne', 181), (u'see', 177), (u'character', 177)]
```

Figura 8.19 As 20 palavras mais frequentes em postagens do Reddit sobre “ciência de dados” e “jogo dos tronos” após a preparação dos dados

Podemos ver na figura 8.19 como a qualidade dos dados melhorou notavelmente. Além disso, observe como certas palavras são encurtadas por causa do radical que aplicamos. Por exemplo, “ciência” e “ciências” tornaram-se “ciência”; “cursos” e “curso” tornaram-se “curso” e assim por diante. Os termos resultantes não são palavras reais, mas ainda assim interpretáveis. Se você insistir em que seus termos permaneçam como palavras reais, a lematização seria o caminho ir.

Com o processo de limpeza de dados “concluído” (observação: um exercício de limpeza de mineração de texto quase nunca pode ser totalmente concluído), tudo o que resta são algumas transformações de dados para colocá-los no formato de saco de palavras.

Primeiro, vamos rotular todos os nossos dados e também criar uma amostra de validação de 100 observações por categoria, conforme mostrado na listagem a seguir.

Listagem 8.7 Transformação final de dados e divisão de dados antes da modelagem

A amostra de validação é composta por dados não rotulados dos dois subreddits: 100 observações de cada conjunto de dados. Os rótulos são mantidos em um conjunto de dados separado.

A amostra de validação será usada para determinar as falhas do modelo através da construção de uma matriz de confusão.

comprimento de retenção = 100

```
rotulado_data1 = [(palavra, 'datascience') para palavra em
    dados['datascience']['wordMatrix']['holdoutLength:]]
rotulado_data2 = [(palavra, 'gameofthrones') para palavra em
    dados['gameofthrones']['wordMatrix']['holdoutLength:]]
dados_rotulados = []
dados_rotulados.extend(dados_rotulados1)
dados_rotulados.extend(dados_rotulados2)
```

Criamos um único conjunto de dados com cada vetor de palavras marcado como 'datascience' ou 'gameofthrones'. Mantemos parte dos dados de lado para an

```
holdout_data = dados['datascience']['wordMatrix']['holdoutLength]
holdout_data.extend(dados['gameofthrones']['wordMatrix']['holdoutLength])
holdout_data_labels = ([('datascience') para
    _ em xrange(holdoutLength)] + [('gameofthrones') para
    _ em xrange(holdoutComprimento)])
```

em

```
dados['datascience']['all_words_dedup'] =
    lista(OrderedDict.fromkeys(
        dados['datascience']['all_words']))
dados['gameofthrones']['all_words_dedup'] =
    lista(OrderedDict.fromkeys(
        dados['gameofthrones']['all_words']))
todas_palavras = []
all_words.extend(dados['datascience']['all_words_dedup'])
all_words.extend(dados['gameofthrones']['all_words_dedup'])
all_words_dedup =
    list(OrderedDict.fromkeys(all_words))
```

Uma lista de todos os termos exclusivos é criada para construir o conjunto de dados de palavras que necessidade de treinar ou pontuar um modelo.

Dados para modelo
treinamento e
testar é
primeiro
embaralhado.

```
dados_preparados = [(palavra: (palavra em x[0]) para palavra
    em todas_palavras_dedup}, x[1]) para x em dados_rotulados]
preparado_holdout_data = [(palavra: (palavra em x[0])
    para palavra em all_words_dedup})
    para x em holdout_data]
```

Os dados são transformados em um formato binário de palavras.

```
aleatório.shuffle(dados_preparados)
train_size = int(len(prepared_data) * 0,75)
trein = preparado_dados[:train_size]
teste = preparado_dados[train_size:]
```

O tamanho dos dados de treinamento será de 75% do total e os 25% restantes serão usados para testar o desempenho do modelo.

A amostra de validação será usada para nosso teste final do modelo e para a criação de um matriz de confusão. Uma *matriz de confusão* é uma forma de verificar o desempenho de um modelo em dados anteriormente não vistos. A matriz mostra quantas observações foram corretas e classificado incorretamente.

Antes de criar ou treinar e testar dados, precisamos dar um último passo: despejar os dados em um formato de pacote de palavras onde cada termo recebe um “Verdadeiro” ou “Falso” rótulo dependendo de sua presença naquela postagem específica. Também precisamos fazer isso para o amostra de validação não rotulada.

Nossos dados preparados agora contêm todos os termos de cada vetor, conforme mostrado na figura 8.20.

```
imprimir dados_preparados[0]
```

```
print prepared_data[0]

({u'sunspear': False, u'profici': False, u'pardon': False, u'selye
s': False, u'four': False, u'davo': False, u'sleev': False, u'slee
:
u'daeron': False, u'portion': False, u'emerg': False, u'fifti': Fals
e, u'decemb': False, u'defend': False, u'sincer': False}, 'datascien
ce')
```

Figura 8.20 Um conjunto binário de palavras prontas para modelagem são dados muito esparsos.

Criamos uma matriz grande, mas esparsa, que nos permite aplicar técnicas do capítulo 5, se necessário. era grande demais para ser manuseado em nossa máquina. Com uma mesa tão pequena, no entanto, não há precisamos disso agora e podemos embaralhar e dividir os dados em um treinamento e

Conjunto de teste.

Embora a maior parte dos seus dados deva sempre ir para o treinamento do modelo, existe uma proporção de divisão ideal. Aqui optamos por uma divisão de 3 a 1, mas fique à vontade para brincar com isso. O mais observações você tem, mais liberdade você tem aqui. Se você tiver poucas observações, precisará alocar relativamente mais para treinar o modelo. Agora estamos prontos para passar para a parte mais gratificante: a análise de dados.

8.3.7 Etapa 5: Análise de dados

Para nossa análise ajustaremos dois algoritmos de classificação aos nossos dados: Naïve Bayes e decisão árvores. Naïve Bayes foi explicado no capítulo 3 e na árvore de decisão anteriormente neste capítulo.

Vamos primeiro testar o desempenho do nosso classificador Naïve Bayes. NLTK vem com um classificador, mas fique à vontade para usar algoritmos de outros pacotes, como SciPy.

```
classificador = nltk.NaiveBayesClassifier.train(trem)
```

Com o classificador treinado, podemos usar os dados do teste para obter uma medida da precisão geral.

```
nltk.classify.accuracy(classificador, teste)
```

```
nltk.classify.accuracy(classifier, test)
0.9681528662420382
```

Figura 8.21 A precisão da classificação é uma medida que representa qual porcentagem de observações foi classificada corretamente nos dados de teste.

A precisão dos dados de teste é estimada em mais de 90%, conforme visto na figura 8.21.

A *precisão da classificação* é o número de observações classificadas corretamente como uma porcentagem do número total de observações. Esteja ciente, porém, que isso pode ser diferente em seu caso se você usou dados diferentes.

```
nltk.classify.accuracy(classificador, teste)
```

Esse é um bom número. Agora podemos recostar-nos e relaxar, certo? Não, na verdade não. Vamos teste-o novamente na amostra de validação de 200 observações e desta vez crie uma matriz de confusão.

```
dados_classificados = classifier.classify_many(prepared_holdout_data)
cm = nltk.ConfusionMatrix(holdout_data_labels, dados_classificados)
imprimir cm
```

A matriz de confusão na figura 8.22 nos mostra que 97% é provavelmente um exagero porque temos 28 (23 + 5) casos classificados incorretamente. Novamente, isso pode ser diferente com o seu dados se você mesmo preencheu o arquivo SQLite.

	g
	a
d	m
a	e
t	o
a	f
s	t
c	h
i	r
e	o
n	n
c	e
e	s

datascience	<77>23
gameofthrones	5<95>

(row = reference; col = test)

Figura 8.22 A matriz de confusão do modelo Naïve Bayes mostra que 28 (23 + 5) observações de 200 foram classificadas incorretamente

Vinte e oito erros de classificação significam que temos uma precisão de 86% na amostra de validação. Isso precisa ser comparado à atribuição aleatória de uma nova postagem à “ciência de dados” ou grupo “gameofthrones”. Se os distribuíssemos aleatoriamente, poderíamos esperar um

precisão de 50%, e nosso modelo parece ter um desempenho melhor do que isso. Vejamos o que usa para determinar as categorias investigando os recursos mais informativos do modelo.

```
imprimir(classificador.show_most_informative_features(20))
```

A Figura 8.23 mostra os 20 principais termos capazes de distinguir entre as duas categorias.

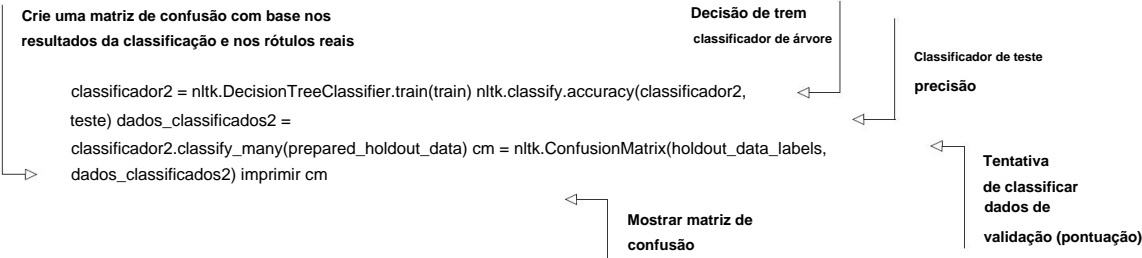
Most Informative Features			
data = True	datasc : gameof =	365.1 : 1.0	
scene = True	gameof : datasc =	63.8 : 1.0	
season = True	gameof : datasc =	62.4 : 1.0	
king = True	gameof : datasc =	47.6 : 1.0	
tv = True	gameof : datasc =	45.1 : 1.0	
kill = True	gameof : datasc =	31.5 : 1.0	
compani = True	datasc : gameof =	28.5 : 1.0	
analysi = True	datasc : gameof =	27.1 : 1.0	
process = True	datasc : gameof =	25.5 : 1.0	
appli = True	datasc : gameof =	25.5 : 1.0	
research = True	datasc : gameof =	23.2 : 1.0	
episod = True	gameof : datasc =	22.2 : 1.0	
market = True	datasc : gameof =	21.7 : 1.0	
watch = True	gameof : datasc =	21.6 : 1.0	
man = True	gameof : datasc =	21.0 : 1.0	
north = True	gameof : datasc =	20.8 : 1.0	
hi = True	datasc : gameof =	20.4 : 1.0	
level = True	datasc : gameof =	19.1 : 1.0	
learn = True	datasc : gameof =	16.9 : 1.0	
job = True	datasc : gameof =	16.6 : 1.0	

Figura 8.23 Os termos mais importantes no modelo de classificação Naïve Bayes

O termo “dados” recebe grande peso e parece ser o indicador mais importante de se um tópico pertence à categoria de ciência de dados. Termos como “cena”, “temporada”, “rei”, “tv” e “matar” são boas indicações de que o assunto é *Game of Thrones*, e não ciência de dados. Todas essas coisas fazem todo o sentido, então o modelo passou tanto na verificação de precisão quanto na verificação de sanidade.

O Naïve Bayes se sai bem, então vamos dar uma olhada na árvore de decisão na listagem a seguir.

Listagem 8.8 Treinamento e avaliação do modelo de árvore de decisão




```
nlk.classify.accuracy(classifier2, test)
```

```
0.9333333333333333
```

Figura 8.24 Precisão do modelo de árvore de decisão

Conforme mostrado na figura 8.24, a precisão prometida é de 93%.

Agora sabemos que não devemos confiar apenas neste único teste, por isso recorreremos mais uma vez a uma matriz de confusão num segundo conjunto de dados, como mostra a figura 8.25.

A Figura 8.25 mostra uma história diferente. Nessas 200 observações da amostra de validação, o modelo de árvore de decisão tende a ser bem classificado quando a postagem é sobre *Game of Thrones*, mas falha miseravelmente quando confrontado com as postagens de ciência de dados. Parece que a modelo tem preferência por *Game of Thrones*, e você pode culpar isso? Vamos dar uma olhada no modelo real, embora neste caso usaremos o Naïve Bayes como modelo final.

```
imprimir(classificador2.pseudocódigo(profundidade=4))
```

	g
	a
d	m
a	e
t	o
a	f
s	t
c	h
i	r
e	o
n	n
c	e
e	s
-----+-----	
datascience	<26>74
gameofthrones	2<98>
-----+-----	
(row = reference; col = test)	

A árvore de decisão possui, como o nome sugere, um modelo semelhante a uma árvore, conforme mostrado na figura 8.26.

Figura 8.25 Matriz de confusão no modelo de árvore de decisão

O Naïve Bayes considera todos os termos e tem pesos atribuídos, mas o modelo da árvore de decisão os percorre sequencialmente, seguindo o caminho da raiz até os ramos externos e folhas. A Figura 8.26 mostra apenas as quatro camadas superiores, começando com o termo "dados". Se "dados" estiverem presentes na postagem, é sempre ciência de dados. Se não for possível encontrar "dados", ele verifica o termo "aprender" e assim continua. Uma possível razão pela qual esta árvore de decisão não está funcionando bem é a falta de poda. Quando uma árvore de decisão é construída ela tem muitas folhas, muitas vezes demais. Uma árvore é então podada até um certo nível para minimizar o sobreajuste.

Uma grande vantagem das árvores de decisão são os efeitos implícitos de interação entre as palavras que ela

```
if data == False:
    if learn == False:
        if python == False:
            if tool == False: return 'gameofthrones'
            if tool == True: return 'datascience'
        if python == True: return 'datascience'
    if learn == True:
        if go == False:
            if wrong == False: return 'datascience'
            if wrong == True: return 'gameofthrones'
        if go == True:
            if upload == False: return 'gameofthrones'
            if upload == True: return 'datascience'
if data == True: return 'datascience'
```

Figura 8.26 Representação da estrutura em árvore do modelo de árvore de decisão

leva em consideração na construção das filiais. Quando vários termos juntos Se criar uma classificação mais forte do que termos únicos, a árvore de decisão superará, na verdade, o Naïve Bayes. Não entraremos em detalhes sobre isso aqui, mas considere este das próximas etapas que você pode seguir para melhorar o modelo.

Agora temos dois modelos de classificação que nos dão uma ideia de como os dois conteúdos dos subreddits diferem. O último passo seria compartilhar essas informações recém-descobertas com outras pessoas.

8.3.8 Etapa 6: Apresentação e automação

Como último passo, precisamos usar o que aprendemos e transformá-lo em uma aplicação útil ou apresentar nossos resultados a outras pessoas. O último capítulo deste livro discute a construção de uma aplicação interativa, pois este é um projeto em si. Por enquanto nos contentaremos com um bom maneira de transmitir nossas descobertas. Um belo gráfico ou, melhor ainda, um gráfico interativo, pode captar o olho; é a cereja do bolo da apresentação. Embora seja fácil e tentador representar os números propriamente ditos ou, no máximo, um gráfico de barras, seria bom dar um passo adiante.

Por exemplo, para representar o modelo Naïve Bayes, poderíamos usar um gráfico de força (figura 8.27), onde o tamanho da bolha e do link representam o quão fortemente relacionada uma palavra está com os subreddits de “jogo dos tronos” ou “ciência de dados”. Observe como as palavras nos balões são frequentemente cortadas; lembre-se de que isso se deve à lematização que aplicamos.

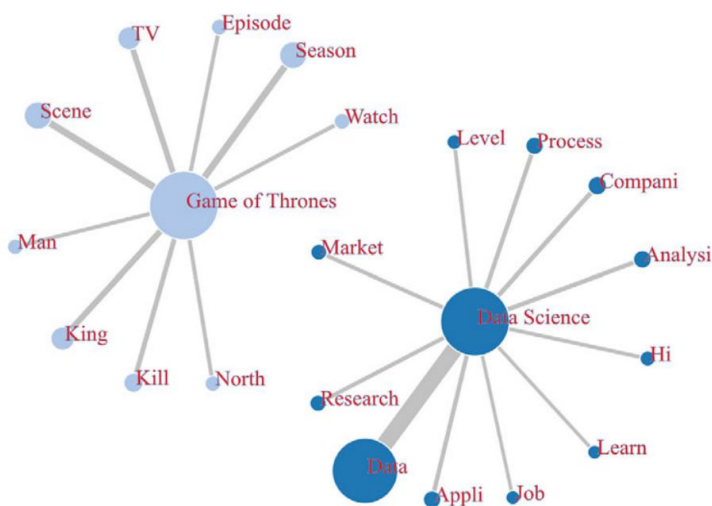


Figura 8.27 Gráfico de força interativo com os 20 principais termos significativos de Naïve Bayes e seus pesos

Embora a figura 8.27 em si seja estática, você pode abrir o arquivo HTML “forceGraph.html” para aproveitar o efeito do gráfico de força d3.js conforme explicado anteriormente neste capítulo. d3.js está fora de o escopo deste livro, mas você não precisa de um conhecimento elaborado de d3.js para usá-lo. Um extenso conjunto de exemplos pode ser usado com ajustes mínimos no código fornecido em <https://github.com/mbostock/d3/wiki/Gallery>. Tudo que você precisa é de bom senso e

um mínimo de conhecimento de JavaScript. O código para o exemplo do gráfico de força pode ser encontrado em <http://bl.ocks.org/mbostock/4062045>.

Também podemos representar a nossa árvore de decisão de uma forma bastante original. Poderíamos optar por uma versão sofisticada de um diagrama de árvore real, mas o seguinte diagrama sunburst é mais original e igualmente divertido de usar.

A Figura 8.28 mostra a camada superior do diagrama sunburst. É possível aumentar o zoom clicando em um segmento circular. Você pode diminuir o zoom clicando no círculo central. O código deste exemplo pode ser encontrado em <http://bl.ocks.org/metmajer/5480307>.



Figura 8.28 Diagrama Sunburst criado a partir dos quatro ramos superiores do modelo de árvore de decisão

Mostrar seus resultados de forma original pode ser a chave para um projeto de sucesso. As pessoas nunca apreciarão o esforço que você fez para alcançar seus resultados se você não puder comunicá-los e se eles forem significativos para elas. Uma visualização de dados original aqui e ali certamente ajuda nisso.

8.4 Resumo

• A mineração de texto é amplamente usada para identificação de entidades, detecção de plágio, identificação de tópicos, tradução, detecção de fraudes, filtragem de spam e muito mais.

• Python possui um kit de ferramentas maduro para mineração de texto chamado NLTK, ou kit de ferramentas de linguagem natural. NLTK é bom para brincar e aprender o básico; para aplicações da vida real, entretanto, o Scikit-learn é geralmente considerado mais "pronto para produção".

Scikit-learn é amplamente utilizado nos capítulos anteriores. • A

preparação de dados textuais é mais intensiva do que a preparação de dados numéricos e envolve técnicas extras, como - *Stemming* - *Cortar* o final de uma palavra de maneira inteligente para que possa ser correspondida

com algumas versões conjugadas ou plurais desta palavra.

– *Lematização* - *Assim como* a lematização, destina-se a remover duplicatas, mas ao contrário derivando, analisa o significado da palavra.

– *Interromper a filtragem de palavras* – *Certas* palavras ocorrem com muita frequência para serem úteis e filtrá-las pode melhorar significativamente os modelos. As palavras irrelevantes geralmente são específicas do corpus.

– *Tokenização*—*Cortar* texto em pedaços. Os tokens podem ser palavras únicas, combinações de palavras (n-gramas) ou até mesmo frases inteiras.

– *Marcação de PDV*—*Marcação* de parte do discurso. Às vezes pode ser útil saber o que a função de uma determinada palavra dentro de uma frase é compreendê-la melhor.

• Em nosso estudo de caso, tentamos distinguir postagens do Reddit sobre "Game of Thrones" e postagens sobre "ciência de dados". Neste esforço, testamos os classificadores Naïve Bayes e de árvore de decisão. Naïve Bayes assume que todos os recursos são independentes uns dos outros; o classificador da árvore de decisão assume dependência, permitindo diferentes modelos.

• Em nosso exemplo, Naïve Bayes produziu o melhor modelo, mas muitas vezes o classificador de árvore de decisão faz um trabalho melhor, geralmente quando há mais dados disponíveis. • Determinamos

a diferença de desempenho usando uma matriz de confusão que calculamos

após a aplicação de ambos os modelos em dados novos (mas rotulados). • Ao

apresentar as descobertas a outras pessoas, pode ser útil incluir uma visualização de dados interessante, capaz de transmitir os seus resultados de uma forma memorável.