Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow

Sumário

Prefácio

O Tsunami do Aprendizado de Máquina

Em 2006, Geoffrey Hinton et al. publicou um artigo (*https://homl.info/136*)[1] demonstrando como treinar uma rede neural profunda capaz de reconhecer algarismos escritos à mão com uma precisão de última geração (> 98%). Chamou-se a técnica de "aprendizado profundo" [*Deep Learning*]. Uma rede neural profunda é um modelo (bastante) simplificado do nosso córtex cerebral, constituído por uma pilha de camadas de neurônios artificiais. Na época, treinar uma rede neural profunda era basicamente uma tarefa impossível[2], e a maioria dos pesquisadores havia desistido da ideia no fim dos anos 1990. Esse artigo reacendeu o interesse da comunidade científica e, em pouco tempo, muitos artigos novos comprovavam que o aprendizado profundo não só era possível como era capaz de resultados surpreendentes (com a ajuda de uma capacidade de processamento monstruosa e de quantidades de dados imensas), que não poderiam se igualar a nenhuma outra técnica de aprendizado de máquina (AM). Esse entusiasmo logo se estendeu a muitas outras áreas do aprendizado de máquina.

Cerca de uma década depois, o aprendizado de máquina conquistou todos os segmentos industriais: está no coração de grande parte dos produtos de alta tecnologia de hoje, classifica os resultados de pesquisa na web, viabiliza o reconhecimento de voz dos smartphones, faz recomendações de vídeos e derrotou o campeão mundial no jogo Go. Antes que você perceba, estará dirigindo seu carro.

O Aprendizado de Máquina em Seus Projetos

Claro que você está entusiasmado com o aprendizado de máquina e gostaria de participar da festa!

E se você tivesse um robô doméstico e, talvez, lhe desse um cérebro? E reconhecimento facial? Ou quem sabe o ensinasse a andar?

Ou talvez sua empresa tenha toneladas de dados (logs de usuários, dados financeiros, de produção, de sensores de máquinas, estatísticas de atendimento ao cliente, relatórios de RH etc.), e muito provavelmente você poderia desenterrar algum tesouro escondido se soubesse onde procurar. Com o aprendizado de máquina, é possível fazer isso e muito mais (*https://homl.info/usecases*):

- Segmentar os clientes e identificar a melhor estratégia de marketing para cada grupo.
- Recomendar produtos para cada cliente com base no que clientes similares compraram.
- Detectar as transações suscetíveis à fraude.
- Prever o faturamento do próximo ano.

Seja lá qual for a razão, você decidiu estudar o aprendizado de máquina e usá-lo em seus projetos. Ótima ideia!

1 Geoffrey E. Hinton et al., "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation* 18 (2006): 1527–1554.

2 Apesar do fato de as redes neurais convolucionais de aprendizado profundo de Yann Lecun terem funcionado bem para o reconhecimento de imagens desde a década de 1990, elas não eram de uso geral.

Objetivo e Abordagem

Este livro parte do princípio de que você não sabe quase nada sobre aprendizado de máquina. O objetivo é apresentar a lógica, as ferramentas e os conceitos necessários para implementar programas capazes de *aprender a partir dos dados*.

Abordaremos muitas técnicas, desde as mais simples às mais comumente utilizadas (como a regressão linear) e até algumas das técnicas de aprendizado profundo que vencem competições com frequência. Em vez de implementar nossos próprios modelos de cada algoritmo, utilizaremos frameworks executáveis do Python:

- A Scikit-Learn (*http://scikit-learn.org/*) é bem fácil de usar e implementa muitos algoritmos do AM de maneira eficiente, por isso é uma excelente porta de entrada para o aprendizado de máquina.
- A TensorFlow (*https://tensorflow.org/*) é uma biblioteca mais complexa para cálculo numérico distribuído. Possibilita treinar e executar grandes redes neurais de maneira eficiente, distribuindo os cálculos entre centenas de servidores com múltiplas GPUs (unidades de processamento gráfico). A TensorFlow (TF) foi criada no Google e suporta muitas das aplicações em larga escala do aprendizado de máquina. Em novembro de 2015, ela se tornou uma biblioteca de código aberto.
- A Keras (*https://keras.io/*) é uma API de aprendizado profundo de alto nível que facilita o treinamento e a execução de redes neurais. Ela roda com a TensorFlow, Theano ou Microsoft Cognitive Toolkit (anteriormente conhecido como CNTK). A TensorFlow já tem a implementação dessa API, chamada *tf.keras*, que suporta alguns recursos avançados da TensorFlow (por exemplo, a capacidade de carregar dados com eficiência).

O livro favorece uma abordagem prática, viabilizando uma compreensão intuitiva do aprendizado de máquina por meio de exemplos objetivos e concretos, e um pouco de teoria. Embora possa ler este livro sem usar o computador, recomendo muitíssimo que você treine com os exemplos de código disponíveis online do Jupyter Notebooks em: *https://github.com/ageron/handson-ml2*.

Pré-requisitos

Este livro pressupõe que você tenha alguma experiência de programação em Python e que esteja familiarizado com as principais bibliotecas científicas do Python, sobretudo a NumPy (*http://numpy.org/*), Pandas (*http://pandas.pydata.org/*) e Matplotlib (*http://matplot lib.org/*).

Além disso, se quiser entender como funcionam as bibliotecas, deve ter um conhecimento razoável de matemática avançada, (cálculo, álgebra linear, probabilidade e estatística). Se ainda não conhece o Python, o *http://learnpython.org/* é um ótimo lugar para começar. O tutorial oficial no python.org (*https://docs.python.org/3/tutorial/*) também é muito bom.

Se nunca usou o Jupyter, o Capítulo 2 lhe mostrará os primeiros passos da instalação: uma ótima ferramenta para ter sempre à mão. Caso não esteja familiarizado com as bibliotecas do Python, o Jupyter Notebooks tem alguns tutoriais. Há também um tutorial rápido de matemática sobre álgebra linear.

Roteiro

Este livro foi organizado em duas partes. A Parte I, *Os Fundamentos do Aprendizado de Máquina*, aborda os seguintes tópicos:

- O que é aprendizado de máquina? Quais problemas ele tenta resolver? Quais são as principais categorias e os conceitos fundamentais dos sistemas do aprendizado de máquina?
- Os principais passos de um típico projeto de aprendizado de máquina.
- Aprendizado ajustando um modelo aos dados.
- Otimizar a função de custos.
- Manipular, limpar e preparar os dados.
- Selecionar e desenvolver funcionalidades.
- Selecionar um modelo e ajustar os hiperparâmetros ao usar a validação cruzada.
- Os principais desafios do aprendizado de máquina, especialmente o subajuste [underfitting] e o sobreajuste [overfitting] (trade-off do viés/variância).
- Os algoritmos de aprendizado mais comuns: regressão linear e polinomial, regressão logística, k-ésimo vizinho mais próximo [k-Nearest Neighbors], máquinas de vetores de suporte [SVM], árvores de decisão, florestas aleatórias e métodos do ensemble.
- Reduzir a dimensionalidade dos dados de treinamento para combater a "maldição da dimensionalidade".
- Outras técnicas de aprendizado não supervisionado, incluindo clusterização [clustering], estimativa de densidade e detecção de anomalias.

A Parte II, *Redes Neurais e Aprendizado Profundo*, aborda os seguintes tópicos:

- O que são redes neurais e para que servem?
- Construir e treinar redes neurais usando a TensorFlow e a Keras.
- As arquiteturas de redes neurais mais importantes: redes neurais do tipo feedforward para dados tabulares, redes convolucionais para visão computacional, redes recorrentes e redes de memória de longo prazo (LSTM) para processamento de sequências, encoders/decoders e transformadores para processamento de linguagem natural, redes autoencoders e adversárias gerativas (GANs) para aprendizado gerativo.
- Técnicas para treinamento de redes neurais profundas.
- Como criar um agente (por exemplo, um bot em um jogo) que pode aprender boas estratégias por tentativa e erro, usando o aprendizado por reforço.
- Carregando e pré-processando grandes quantidades de dados com eficiência.
- Treinamento e implementação de modelos TensorFlow em grande escala.

A Parte I se baseia na Scikit-Learn, enquanto a Parte II usa a TensorFlow e a Keras.

Não dê um passo maior que a perna: embora o aprendizado profundo seja, sem sombra de dúvidas, uma das áreas mais interessantes do aprendizado de máquina, você deve dominar os princípios básicos primeiro. Além do mais, grande parte dos problemas pode ser resolvida com técnicas mais simples, como os métodos floresta aleatória e ensemble (discutidos na Parte I). O aprendizado profundo se adéqua melhor aos problemas complexos, como reconhecimento de imagem e de voz ou processamento de linguagem natural, desde que se tenha dados, capacidade de processamento e paciência suficientes.

Mudanças na Segunda Edição

Esta segunda edição tem seis objetivos principais:

1. Abordar as noções adicionais de AM: mais técnicas de aprendizado não supervisionado (clusterização, detecção de anomalias, estimativa de densidade e modelos de mistura); mais técnicas para treinar redes profundas (redes autonormalizadas); técnicas adicionais de visão computacional (Xception, SENet, detecção de objetos com YOLO e segmentação semântica usando R-CNN); manipulação de sequências usando redes neurais convolucionais (CNNs, incluindo WaveNet); processamento de linguagem natural usando redes neurais recorrentes (RNNs), CNNs e transformadores; e GANs.
2. Abranger as bibliotecas e APIs adicionais (Keras, a Data API, agentes TF para aprendizado por reforço) e o treinamento e a implementação de modelos do TF em escala usando a API de estratégias de distribuição, TF-Serving e a plataforma Google Cloud AI. Breve apresentação do TF Transform, TFLite, TF Addons/Seq2Seq, e da TensorFlow.js.
3. Analisar alguns dos últimos resultados importantes sobre a pesquisa de aprendizado de máquina.
4. Fazer a migração de todos os capítulos da TensorFlow para a TensorFlow 2 e usar a implementação da Keras API (tf.keras) da TensorFlow sempre que possível.
5. Atualizar os exemplos de código para usar as versões mais recentes do Scikit-Learn, NumPy, Pandas, Matplotlib e outras bibliotecas.
6. Esclarecer algumas seções e corrigir alguns erros, graças ao excelente feedback dos leitores.

Alguns capítulos foram adicionados, outros foram reescritos e alguns, reordenados. Acesse para mais detalhes sobre o que mudou na segunda edição: *https://homl.info/changes2* [conteúdo em inglês].

Outros Recursos

Há muitos recursos excelentes disponíveis a respeito do aprendizado de máquina. Por exemplo, o curso de AM de Andrew Ng no Coursera (*https://homl.info/ngcourse*) é incrível, embora exija um investimento de tempo significativo (meses).

Há também muitos sites interessantes sobre aprendizado de máquina, incluindo, claro, o excepcional Guia do Usuário do Scikit-Learn (*https://homl.info/skdoc*). Você também pode gostar do Dataquest (*https://www.dataquest.io/*), que oferece tutoriais interativos muito interessantes e blogs de AM, como aqueles listados no Quora (*https://homl.info/1*). Por fim, o site Deep Learning (*http://deeplearning.net/*) tem uma lista ótima de recursos para continuarmos a aprender.

Há também muitos outros livros introdutórios sobre aprendizado de máquina:

- *Data Science do Zero*, de Joel Grus (Alta Books). Apresenta os conceitos básicos do aprendizado de máquina e implementa alguns dos principais algoritmos em puro Python (do zero, como o nome sugere).
- *Machine Learning: An Algorithmic Perspective*, de Stephen Marsland (Chapman and Hall). É uma ótima introdução ao aprendizado de máquina que aborda uma ampla gama de tópicos em profundidade, com exemplos de código em Python (também a partir do zero, mas utilizando o NumPy).
- *Python Machine Learning*, de Sebastian Raschka (Packt Publishing). É também uma ótima introdução ao aprendizado de máquina e incentiva o uso das bibliotecas de código aberto do Python (Pylearn 2 e Theano).

- *Deep Learning with Python,* de François Chollet (Manning). Livro muito prático que aborda uma grande variedade de tópicos de forma clara e concisa, como seria de esperar do autor da excelente biblioteca Keras. Dá preferência aos exemplos de código sobre teoria matemática.
- *The Hundred-Page Machine Learning Book,* de Andriy Burkov. Livro pequeno que abrange uma variedade impressionante de tópicos, apresentando-os de forma bem acessível sem fugir das equações matemáticas.
- *Learning from Data*, de Yaser S. Abu-Mostafa, Malik Magdon-Ismail e Hsuan-Tien Lin (AMLBook). Por meio de uma abordagem bastante teórica sobre AM, esta obra proporciona insights profundos, especialmente sobre a compensação do viés/variância (veja o Capítulo 4).
- *Inteligência Artificial*, 3ª Edição, de Stuart Russell e Peter Norvig (Campus). É um livro ótimo (e gigante) que aborda uma quantidade incrível de tópicos, incluindo o aprendizado de máquina. Ajuda a esclarecer muitas coisas a respeito do AM.

Por último, uma ótima maneira de aprender é entrar em sites de competição como o kaggle.com, que lhe possibilita praticar suas habilidades usando problemas reais com a ajuda e os insights de alguns dos melhores profissionais de AM existentes.

Convenções Usadas Neste Livro

As seguintes convenções tipográficas são usadas neste livro:

Itálicos

Indica condições novas, URLs, endereços de e-mail, nomes de arquivos e extensões de arquivos.

`Fonte monoespaçada`

Usada para listagens de programas, bem como dentro de parágrafos para referenciar elementos do programa: nomes de variáveis ou funções, bancos de dados, tipos de dados, variáveis de ambiente, declarações e palavras-chave.

`Fonte monoespaçada em negrito`

Mostra comandos ou outro texto que deve ser digitado pelo usuário.

`Fonte monoespaçada em itálico`

Mostra o texto que deve ser substituído por valores fornecidos pelo usuário ou por valores determinados pelo contexto.

Este elemento significa uma dica ou sugestão.

Este significa uma nota geral.

Este indica alerta ou cautela.

Exemplos de Códigos

Disponibilizamos uma série de Jupyter Notebooks repletos de material complementar, como exemplos de código e exercícios, disponíveis para download em: *https://github.com/ageron/handson-ml2*.

Neste livro, alguns dos exemplos de código deixam de lado seções ou detalhes repetitivos que são óbvios ou que não estão relacionados ao aprendizado de máquina com o intuito de se concentrar nas partes importantes do código e abrir caminho para abordar mais tópicos. Caso queira exemplos de código completos, todos estão disponíveis nos Jupyter Notebooks.

Repare que, quando os exemplos de código exibem algumas saídas, eles são mostrados com prompts de comando do Python (>>> and ...), como em uma shell do Python, para distinguir claramente o código das saídas. Por exemplo, este código define a função `square()`, depois calcula e exibe o quadrado de 3:

```
>>> def square(x):
...     return x ** 2
...
>>> result = square(3)
>>> result
9
```

Quando o código não exibe nada, os prompts não são usados. No entanto, o resultado às vezes pode ser mostrado como um comentário, assim:

```
def square(x):
    return x ** 2
result = square(3) # o resultado é 9
```

Usando Exemplos de Código

O propósito deste livro é ajudá-lo a alcançar seus objetivos. Em geral, se um código de exemplo for apresentado, você poderá utilizá-lo nos programas e documentações. Não é necessário entrar em contato conosco para obter permissão de uso, a menos que esteja reproduzindo uma parte significativa do código. Por exemplo, escrever um programa que utiliza vários blocos de código deste livro não requer permissão. Vender ou distribuir um CD-ROM com exemplos dos livros da Alta Books exigirá permissão. Responder a uma pergunta citando este livro e mencionando um exemplo de código não requer permissão, mas a inserção de uma quantidade substancial de exemplos de código referente a esta obra na documentação do seu produto exige permissão.

Agradecemos, mas não exigimos que você use citações ou referência. Uma referência geralmente inclui o autor, o título, o editor e a data de publicação. Por exemplo: "GÉRON, Aurélien. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow*: Rio de Janeiro: Alta Books, 2021."

Agradecimentos

Nunca, nem em sonho, imaginei que a primeira edição deste livro atingisse um público tão grande. Recebi muitas mensagens dos leitores, muitas perguntas, algumas simpáticas indicando erratas e a maioria me enviando palavras de incentivo. Mal posso expressar o quanto sou grato a todos esses leitores pelo imenso apoio. Muitíssimo obrigado a todos.

Por favor, não pense duas vezes em registrar os problemas no GitHub (*https://homl.info/issues2*), caso encontre erros nos exemplos de código ou se quiser perguntar algo. Alguns leitores também compartilharam como este livro os ajudou a conseguir seu primeiro emprego ou a solucionar um problema concreto em que estavam trabalhando. É o tipo de feedback extremamente motivador. Se você achar este livro proveitoso, adoraria que compartilhasse sua história comigo, em particular (por exemplo, via LinkedIn, em *https://www.linkedin.com/in/aurelien-geron/*) ou publicamente (com um tuíte ou por meio de uma avaliação na Amazon).

Sou também imensamente grato a todas as pessoas incríveis que, mesmo com a correria do dia a dia, tiraram um tempo para revisar meu livro com tanto cuidado. Em particular, gostaria de agradecer a François Chollet por revisar todos os capítulos baseados na Keras e na TensorFlow e por me fornecer um excelente feedback detalhado. Uma vez que a Keras é um dos principais acréscimos desta segunda edição, contar com um autor como François foi inestimável. Recomendo o livro *Deep Learning with Python,* de François (*https://homl.info/cholletbook*) (Manning): tem a concisão, a clareza e a profundidade da própria biblioteca Keras. Um agradecimento especial também a Ankur Patel, que revisou todos os capítulos desta segunda edição e me deu um ótimo feedback, sobretudo no Capítulo 9, que aborda técnicas de aprendizado não supervisionado. Ele poderia escrever um livro inteiro sobre o assunto… Espere aí, ele escreveu! Confira o *Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data* (*https://homl.info/patel*) (O'Reilly). Muitíssimo obrigado também a Olzhas Akpambetov, que revisou todos os capítulos da Parte II do livro, testou boa parte do código e deu ótimas sugestões. Sou grato a Mark Daoust, Jon Krohn, Dominic Monn e Josh Patterson por revisarem a Parte II deste livro de forma tão minuciosa e por partilharem seus conhecimentos. Eles não pouparam nenhum esforço e deram um feedback de grande valia.

Enquanto escrevia esta segunda edição, tive a felicidade de ser muito ajudado pelos membros da equipe da TensorFlow — em particular Martin Wicke, que respondeu incansavelmente a dezenas de minhas perguntas e encaminhou o restante para as pessoas certas, dentre elas: Karmel Allison, Paige Bailey, Eugene Brevdo, William Chargin, Daniel "Wolff" Dobson, Nick Felt, Bruce Fontaine, Goldie Gadde, Sandeep Gupta, Priya Gupta, Kevin Haas, Konstantinos Katsiapis ,Viacheslav Kovalevskyi, Allen Lavoie, Clemens Mewald, Dan Moldovan, Sean Morgan, Tom O'Malley, Alexandre Passos, André Susano Pinto, Anthony Platanios, Oscar Ramirez, Anna Revinskaya, Saurabh Saxena, Ryan Sepassi, Jiri Simsa, Xiaodan Song, Christina Sorokin, Dustin Tran, Todd Wang, Pete Warden (que também revisou a primeira edição), Edd Wilder-James e Yuefeng Zhou. Vocês me ajudaram muito. Agradeço imensamente a todos, e aos outros membros da equipe da TensorFlow, não apenas pela ajuda, mas também por criar uma ótima biblioteca! Agradecimentos especiais a Irene Giannoumis e Robert Crowe, da equipe TFX, por revisarem os Capítulos 13 e 19 cuidadosamente.

Muito obrigado também à equipe incrível da O'Reilly, sobretudo Nicole Taché, que me deu um feedback elucidativo, sempre cheia de ânimo, encorajadora e prestativa: eu não poderia sequer sonhar com uma editora melhor. Muito obrigado a Michele Cronin, que foi muito solícita (e paciente) no início desta segunda edição, e a Kristen Brown, editora de produção da segunda edição, que acompanhou todas as etapas (ela também coordenou as correções e atualizações para cada reimpressão da primeira edição). Agradeço também a Rachel Monaghan e Amanda Kersey pelo trabalho de editoração completo (respectivamente para a primeira e a segunda edição) e a Johnny O'Toole, que cuidou do relacionamento com a Amazon e respondeu a muitas de minhas perguntas. Obrigado também a Marie Beaugureau, Ben Lorica, Mike Loukides e Laurel Ruma por acreditarem neste projeto e me ajudarem a definir seu escopo. Agradeço a Matt Hacker e toda a equipe da Atlas por responderem a todas as minhas perguntas técnicas sobre formatação, AsciiDoc e LaTeX, e a Nick Adams, Rebecca Demarest, Rachel Head, Judith McConville, Helen Monroe, Helen Monroe, Karen Montgomery, Rachel Roumeliotis e todos da O'Reilly que contribuíram para este livro.

Gostaria de agradecer aos meus colegas do Google, em especial à equipe de classificação de vídeos do YouTube, por me ensinar muito sobre o aprendizado de máquina. Nunca teria escrito a primeira edição deste livro sem a ajuda deles. Agradecimentos especiais aos meus gurus pessoais de AM: Clément Courbet, Julien Dubois, Mathias Kende, Daniel Kitachewsky, James Pack, Alexander Pak, Anosh Raj, Vitor Sessak, Wiktor Tomczak, Ingrid von Glehn e Rich Washington. E obrigado a todos com quem trabalhei no YouTube e nas equipes maravilhosas de pesquisa do Google, em Mountain View. Muito obrigado a Martin Andrews, Sam Witteveen e Jason Zaman por me receberem no grupo Google Developer Experts em Singapura, com o grande apoio de Soonson Kwon e por todas as discussões excelentes que tivemos sobre o aprendizado profundo e a TensorFlow. Qualquer pessoa que se interesse por aprendizado profundo em Singapura deve participar do encontro do Deep Learning Singapore (*https://homl.info/meetupsg*). Jason merece um agradecimento especial por compartilhar parte de sua experiência em TFLite no Capítulo 19!

Jamais esquecerei as pessoas generosas que revisaram a primeira edição deste livro: David Andrzejewski, Lukas Biewald, Justin Francis, Vincent Guilbeau, Eddy Hung, Karim Matrah, Grégoire Mesnil, Salim Sémaoune, Iain Smears, Michel Tessier, Ingrid von Glehn, Pete Warden e claro, o meu caro irmão Sylvain. Agradecimentos especiais a Haesun Park, que me deu um excelente feedback e identificou diversos erros à medida que traduzia para o idioma coreano a primeira edição deste livro. Ele também traduziu os Jupyter Notebooks para o coreano, sem mencionar a documentação da TensorFlow. Eu não falo uma palavra em coreano, mas, a julgar pela qualidade de seus comentários, todas as suas traduções devem ser excelentes! Haesun também gentilmente contribuiu com algumas das soluções para os exercícios desta segunda edição.

Por último, mas não menos importante, sou infinitamente grato à minha digníssima esposa, Emmanuelle, e aos nossos três filhos maravilhosos, Alexandre, Rémi e Gabrielle, por me incentivarem a trabalhar com afinco neste livro e pela curiosidade insaciável: explicar alguns dos conceitos mais difíceis deste livro para minha esposa e filhos me ajudou a desanuviar a mente e a aprimorar muitos aspectos da obra. E eles ainda me traziam café e cookies! O que mais eu poderia querer neste mundo?

PARTE I

OS CONCEITOS BÁSICOS DO APRENDIZADO DE MÁQUINA

O Cenário do Aprendizado de Máquina

Quando a maioria das pessoas ouve "aprendizado de máquina", logo imagina um robô: um mordomo confiável ou um Exterminador do Futuro, dependendo de para quem você perguntar. No entanto, o aprendizado de máquina não povoa apenas o imaginário futurista; ele já está entre nós. Na verdade, há algumas décadas o AM foi introduzido como reconhecimento ótico de caracteres (OCR) em alguns aplicativos específicos, mas o primeiro aplicativo que realmente se popularizou e conquistou o mundo na década de 1990, melhorando a vida de centenas de milhões de pessoas, foi o *filtro de spam*. Não era exatamente um robô autoconsciente da Skynet com IA, mas pode ser tecnicamente classificado como aprendizado de máquina (uma máquina que aprendeu tão bem que raramente é necessário marcar um e-mail como spam). O filtro de spam foi seguido por centenas de aplicativos AM que agora, silenciosamente, integram centenas de produtos e funcionalidades, de recomendações a buscas por voz, que são utilizados regularmente.

Onde começa e onde termina o aprendizado de máquina? O que significa exatamente para uma máquina *aprender* alguma coisa? Se eu fizer um download de uma cópia da Wikipédia, meu computador "aprenderá" algo? Será que de repente ele fica mais inteligente? Neste capítulo, começaremos esclarecendo o que é o aprendizado de máquina e por que você vai querer utilizá-lo.

Logo, antes de iniciarmos a exploração do mundo do aprendizado de máquina, analisaremos seu mapa e conheceremos suas principais regiões e os cenários mais conhecidos: aprendizado supervisionado vs. não supervisionado, aprendizado online vs. aprendizado em batch, aprendizado baseado em instâncias (IBL) vs. aprendizado baseado em modelo. Em seguida, analisaremos o fluxo de trabalho de um típico projeto de AM, discutiremos os principais desafios que poderá enfrentar e mostraremos como avaliar e aperfeiçoar um sistema de aprendizado de máquina.

Este capítulo apresenta muitos conceitos básicos (e jargões) que todo cientista de dados deve saber de cor. Será uma visão geral de alto nível (o único capítulo sem muito código), tudo simplificado, mas você deve garantir que entendeu tudo perfeitamente antes de continuar. Então, pegue um café e mãos à obra!

Se você já conhece todos os conceitos básicos do aprendizado de máquina, pule diretamente para o Capítulo 2. Se não tiver certeza, tente responder a todas as perguntas listadas no fim do capítulo antes de continuar.

O que É Aprendizado de Máquina?

Aprendizado de máquina é a ciência (e a arte) da programação de computadores de modo que eles possam aprender com os dados. Veja uma definição mais generalizada:

> [Aprendizado de máquina é o] campo de estudo que possibilita aos computadores a habilidade de aprender sem explicitamente programá-los.
>
> — Arthur Samuel, 1959

Agora uma definição orientada à engenharia:

> Alega-se que um programa de computador aprende pela experiência *E* em relação a algum tipo de tarefa *T* e alguma medida de desempenho *P* se o seu desempenho em *T*, conforme medido por P, melhora com a experiência *E*.
>
> — Tom Mitchell, 1997

Por exemplo, seu filtro de spam é um programa de aprendizado de máquina que pode aprender a marcar os e-mails como spam (por exemplo, os marcados pelos usuários) e como corriqueiros (não spam, também chamados de "ham"). Os exemplos utilizados pelo sistema para o aprendizado se chamam *conjuntos de treinamentos*. Cada exemplo de treinamento se chama *instância de treinamento* (ou *amostra*). Nesse caso, a tarefa *T* sinaliza os e-mails novos como spam, a experiência *E* é o dado de treinamento e a medida de desempenho *P* precisa ser definida; por exemplo, você pode utilizar a proporção de e-mails sinalizados corretamente. Essa medida de desempenho particular se chama *acurácia* e é usada frequentemente em tarefas de classificação.

Se fizer o download de uma cópia da Wikipédia, seu computador terá bastante dados, mas não terá repentinamente um melhor desempenho em nenhuma tarefa. Logo, fazer o download de uma cópia da Wikipédia não é aprendizado de máquina.

Por que Usar o Aprendizado de Máquina?

Pense em como você desenvolveria um filtro de spam usando as técnicas de programação tradicionais (Figura 1-1):

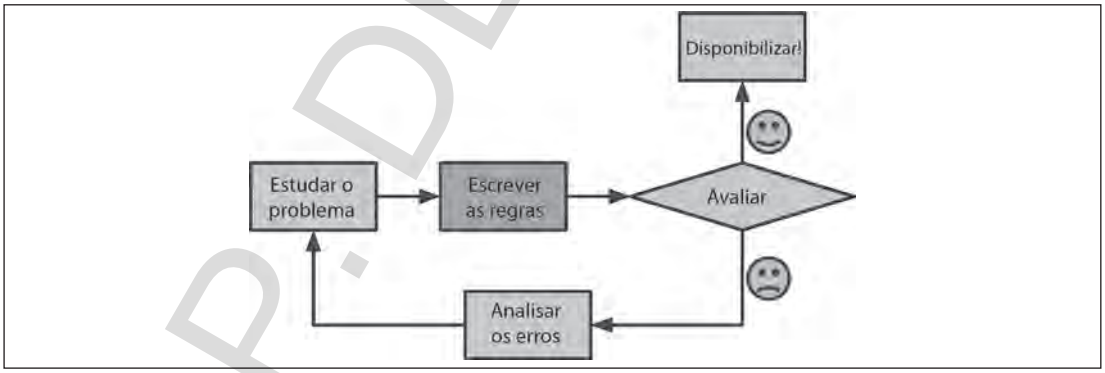

Figura 1-1. Abordagem tradicional

1. Primeiro, você identificaria as características do spam. Talvez você perceba que algumas palavras ou frases ("Para você", "Cartão de crédito", "De graça" e "Oferta imperdível") costumam aparecer muito no campo do assunto. Talvez você repare em outros padrões no nome do remetente, no corpo do e-mail e assim por diante.
2. Segundo, você escreveria um algoritmo de detecção para cada um dos padrões observados, e, se fossem detectados, seu programa sinalizaria esses e-mails como spam.
3. Por último, você testaria seu programa e repetiria os passos 1 e 2 até que ele estivesse bom o suficiente para ser disponibilizado.

Se o problema for difícil, seu programa eventualmente se tornará uma extensa lista de regras complexas — a manutenção não será nada fácil.

Em contrapartida, um filtro de spam baseado em técnicas de aprendizado de máquina aprende automaticamente quais palavras e frases são bons indicadores de spam, ao detectar os padrões de palavras inusitadamente frequentes em exemplos de spam quando comparados aos exemplos dos e-mails "ham" (Figura 1-2). O programa é bem menor, de fácil manutenção e, certamente, mais preciso.

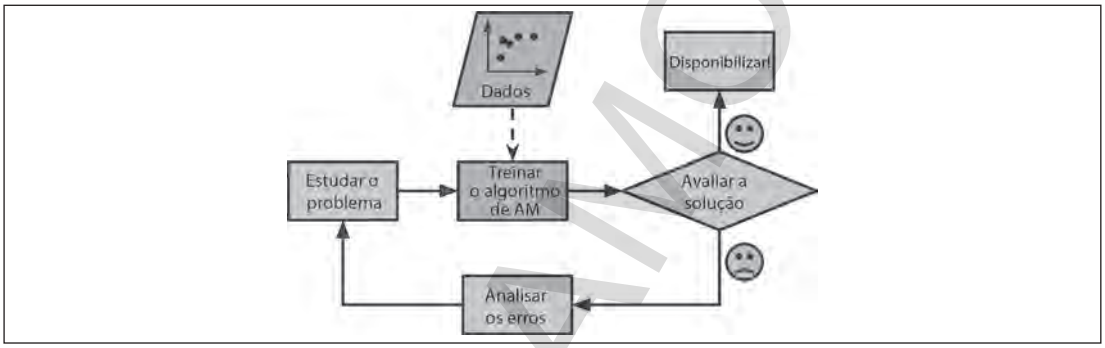

Figura 1-2. Abordagem do aprendizado de máquina

E se os spammers perceberem que todos os seus e-mails com "Para você" estão sendo bloqueados? Eles poderão começar a escrever "Só para você". Um filtro de spam que utiliza técnicas de programação tradicionais precisaria ser atualizado para sinalizar os e-mails "Só para você". Se os spammers continuam burlando seu filtro de spam, será preciso escrever novas regras interminavelmente.

Por outro lado, um filtro de spam baseado em técnicas de aprendizado de máquina percebe automaticamente que "Só para você" se tornou frequente no spam sinalizado pelos usuários, e, assim, começa a marcá-los sem a sua intervenção (Figura 1-3).

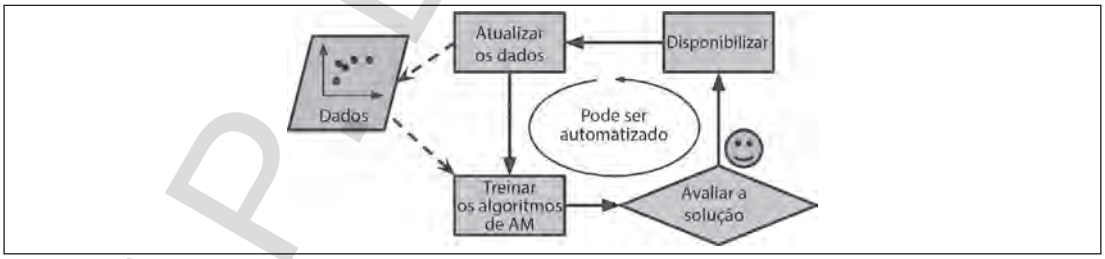

Figura 1-3. Adaptando-se automaticamente à mudança

Outro campo em que o aprendizado de máquina se destaca é o de problemas muito complexos para as abordagens tradicionais ou que não têm um algoritmo conhecido. Por exemplo, considere o reconhecimento de voz. Digamos que você deseja começar com o básico e escreva um programa capaz

de distinguir as palavras “one” e “two”. Você deve perceber que a palavra “two” começa com um som chiado e agudo (“T”), então você apela para a codificação rígida e programa um algoritmo que calcula a intensidade do som e o utiliza para distinguir “one” e “two” — obviamente essa técnica não será dimensionada a milhares de palavras faladas por milhões de pessoas diferentes em ambientes confusos e em centenas de idiomas. A melhor solução (pelo menos hoje) seria escrever um algoritmo que aprenda sozinho por meio de muitas gravações de exemplos para cada palavra.

Por fim, o aprendizado de máquina pode ajudar os seres humanos a aprender (Figura 1-4): os algoritmos do AM podem ser inspecionados para que vejamos o que eles aprenderam (ainda que para alguns algoritmos isso possa ser complicado). Por exemplo, uma vez que o filtro foi treinado para o spam, ele pode ser simplesmente inspecionado e evidenciar uma lista de palavras e combinações previstas que ele acredita serem as mais prováveis. Às vezes, isso revelará correlações não esperadas, ou tendências novas, resultando em uma melhor compreensão do problema. Aplicar técnicas do AM para se aprofundar em grandes quantidades de dados pode ajudar na descoberta de padrões que não eram explícitos. Isso se chama *mineração de dados*.

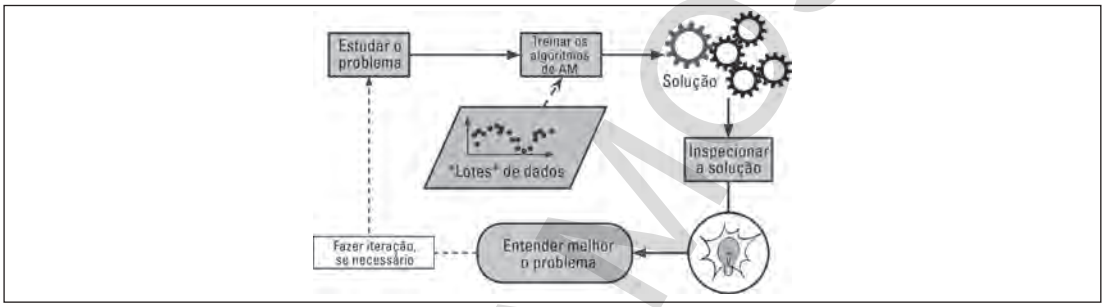

Figura 1-4. O aprendizado de máquina pode ajudar no ensino de humanos

Resumindo, o aprendizado de máquina é ótimo para:

- Problemas para os quais as soluções atuais exigem muitos ajustes finos ou extensas listas de regras: um algoritmo de aprendizado de máquina geralmente simplifica e tem um desempenho melhor do que a abordagem tradicional.
- Problemas complexos para os quais não existe uma boa solução quando utilizamos uma abordagem tradicional: talvez as melhores técnicas de aprendizado de máquina encontrem uma solução.
- Adaptabilidade de ambientes: um sistema de aprendizado de máquina pode se adaptar a novos dados.
- Entendimento de problemas complexos e grandes quantidades de dados.

Exemplos de Aplicações

Vejamos alguns exemplos reais de tarefas de aprendizado de máquina e suas respectivas técnicas:

Análise de imagens de produtos em uma linha de produção a fim de classificá-los automaticamente

Classificação de imagens normalmente realizada usando redes neurais convolucionais (CNNs; veja o Capítulo 14).

Detecção de tumores a partir de exames de imagens cerebrais

Algoritmos de segmentação semântica, em que cada pixel da imagem é classificado (como queremos determinar a localização exata e a forma dos tumores), geralmente usando também as CNNs.

Classificação automática de artigos de notícias

Processamento de linguagem natural (PNL) e, mais especificamente, classificação de texto, que pode ser abordada utilizando as redes neurais recorrentes (RNNs), CNNs ou transformadores (veja o Capítulo 16).

Sinalização automática de comentários ofensivos em fóruns de discussão

Classificação de texto, usando as mesmas ferramentas de PNL.

Resumo automático de documentos extensos

Ramo da PNL chamado resumo de texto, novamente usando as mesmas ferramentas.

Criação de um chatbot ou de um assistente pessoal

Envolve muitos componentes da PNL, dentre eles a compreensão de linguagem natural (NLU) e módulos de resposta a perguntas.

Previsão do faturamento da sua empresa no próximo ano, com base em muitas métricas de desempenho

Tarefa de regressão (ou seja, predição de valores) que pode ser abordada usando qualquer modelo de regressão como: modelo de regressão linear ou regressão polinomial (veja o Capítulo 4), modelo SVM de regressão (veja o Capítulo 5), floresta aleatória de regressão (veja Capítulo 7) ou rede neural artificial (veja o Capítulo 10). Se quiser englobar sequências de métricas de desempenho anteriores, use RNNs, CNNs ou transformadores (veja os Capítulos 15 e 16).

Fazer seu app responder aos comandos de voz

Reconhecimento de voz que exige o processamento de amostras de áudio: como são sequências longas e complexas, normalmente são processadas usando RNNs, CNNs ou transformadores (veja os Capítulos 15 e 16).

Detecção de fraudes com cartão de crédito

Detecção de anomalia (veja o Capítulo 9).

Segmentação de clientes com base em suas compras, para que você possa elaborar uma estratégia de marketing diferente para cada segmento

Uso da clusterização (veja o Capítulo 9).

Representação de um conjunto de dados complexos e de alta dimensão em um diagrama claro e criterioso

Visualização de dados, em geral, engloba técnicas de redução de dimensionalidade (veja o Capítulo 8).

Recomendação de um produto no qual um cliente possa se interessar, com base em compras anteriores

Sistema de recomendação. Uma das abordagens é fornecer os dados das compras anteriores (e outras informações sobre o cliente) a uma rede neural artificial (veja o Capítulo 10) e fazer com que ela evidencie a próxima compra mais provável. Essa rede neural normalmente seria treinada em sequências anteriores de compras de todos os clientes.

Criar um bot inteligente para um jogo

Via de regra, usa-se o aprendizado por reforço (RL; veja o Capítulo 18), que é um ramo do aprendizado de máquina que treina agentes (como bots) a fim de escolher as ações que maximizarão suas recompensas ao longo do tempo (por exemplo, um bot pode receber uma recompensa sempre que o jogador perde alguns pontos de vida), em um determinado ambiente (como o jogo). O famoso programa AlphaGo, que derrotou o campeão mundial no jogo Go, foi desenvolvido com o uso do RL.

Essa lista é praticamente infinita, mas espero que você tenha uma ideia do alcance e da complexidade impressionante das tarefas que o aprendizado de máquina pode abordar e os tipos de técnicas que usaria para cada tarefa.

Tipos de Sistemas do Aprendizado de Máquina

Existem tantos tipos diferentes de sistemas de aprendizado de máquina que ajuda e muito classificá-los em categorias amplas, com base nos seguintes critérios:

- Serem ou não treinados com supervisão humana (aprendizado supervisionado, não supervisionado e semissupervisionado e aprendizado por reforço).
- Se podem ou não aprender gradativamente em tempo real (aprendizado online vs. aprendizado em batch).
- Se funcionam simplesmente comparando novos pontos de dados com pontos de dados conhecidos, ou se detectam padrões em dados de treinamento e criam um modelo preditivo, como os cientistas (aprendizado baseado em instâncias vs. aprendizado baseado em modelo).

Esses critérios não são restritivos; você pode combiná-los da forma que quiser. Por exemplo, um filtro de spam de última geração pode aprender instantaneamente com a utilização de um modelo de rede neural profundo treinado a partir de exemplos de spam e ham, o que faz dele um sistema de aprendizado supervisionado online, baseado em modelos.

Vamos analisar cada um desses critérios com um pouco mais de atenção.

Aprendizado Supervisionado/Não Supervisionado

Os sistemas de aprendizado de máquina podem ser classificados de acordo com a quantidade e o tipo de supervisão que recebem durante o treinamento. Existem quatro categorias principais de aprendizado: supervisionado, não supervisionado, semissupervisionado e aprendizado por reforço.

Aprendizado supervisionado

No aprendizado supervisionado, o conjunto de treinamento que você fornece ao algoritmo inclui as soluções desejadas, chamadas de *rótulos* ou *labels* (Figura 1-5).

Figura 1-5. Um conjunto de treinamento rotulado para classificação de spam (um exemplo de aprendizado supervisionado)

A *classificação* é uma típica tarefa de aprendizado supervisionado. O filtro de spam é um bom exemplo: ele é treinado com muitos exemplos de e-mails junto às *classes* (spam ou ham) e deve aprender a classificar e-mails novos. Outra típica tarefa é prever um valor numérico *alvo* [*target*][1], como o preço de um carro, dado um conjunto de *características* [*features*][2] (quilometragem, tempo de uso, marca etc.) chamados *preditores*. Esse tipo de tarefa se chama *regressão* (Figura 1-6).[3] Para treinar o sistema, é necessário fornecer muitos exemplos de carros, incluindo seus preditores e rótulos (ou seja, seus preços).

1 N. da T.: Alvo é a variável dependente ou variável de resposta, aquilo que você está tentando predizer.

2 N. da T.: Feature se refere a uma característica que descreve um objeto. Seja lá qual for o atributo do objeto, ele pode ser tratado como uma feature.

3 Curiosidade: Esse nome esquisito é um termo estatístico cunhado por Francis Galton, enquanto ele estudava o fato de que os filhos de pessoas altas costumam ter a estatura mais baixa que os pais. Como a estatura das crianças era menor, ele chamou de *regressão à média*, nome dado aos métodos usados para analisar as correlações entre as variáveis.

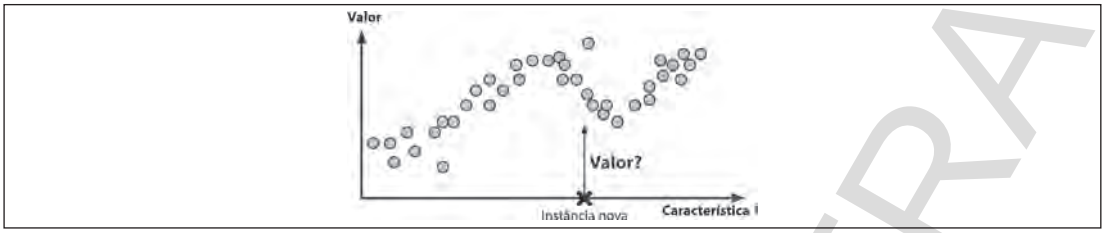

Figura 1-6. Um problema de regressão: prever um valor, dada a entrada de uma característica (geralmente existem diversas entradas de característica e, às vezes, diversos valores de saída)

Em aprendizado de máquina, um *atributo* é um tipo de dado (por exemplo, "quilometragem"), ao passo que uma *característica* assume vários significados; dependendo do contexto, geralmente significa um atributo mais o seu valor (por exemplo: "Quilometragem = 15.000"). Muitas pessoas utilizam as palavras *atributo* e *característica* como sinônimos.

Observe que alguns algoritmos de regressão também podem ser utilizados para classificação e vice-versa. Por exemplo, a regressão logística é comumente utilizada para classificação, pois consegue gerar um valor correspondente à probabilidade de pertencer a uma determinada classe (por exemplo, 20% de chance de ser spam).

Seguem alguns dos algoritmos mais importantes do aprendizado supervisionado (abordados neste livro):

- K-ésimo vizinho mais próximo.
- Regressão linear.
- Regressão logística.
- Máquinas de vetores de suporte (SVMs).
- Árvores de decisão e florestas aleatórias.
- Redes neurais.[4]

Aprendizado não supervisionado

No *aprendizado não supervisionado*, como você pode imaginar, os dados de treinamento não são rotulados (Figura 1-7). O sistema tenta aprender sem um professor.

Figura 1-7. Conjunto de treinamento não rotulado para aprendizado não supervisionado

4 Algumas arquiteturas de redes neurais podem ser não supervisionadas, como autoencoders e máquinas de Boltzmann restritas (RBM). Também podem ser semissupervisionadas, como redes de crenças profundas e pré-treinamento sem supervisão.

Seguem alguns dos algoritmos mais importantes de aprendizado não supervisionado (a maioria deles é abordada nos Capítulo 8 e 9):

- Clusterização
  - — K-Means (Clusterização K-média).
  - — DBSCAN (clusterização espacial baseada em densidade de aplicações com ruído).
  - — Análise de cluster hierárquica (HCA).
- Detecção de anomalias e de novidades
  - — One-class SVM.
  - — Floresta de isolamento.
- Visualização e redução da dimensionalidade
  - — Análise de Componentes Principais (ACP).
  - — Kernel ACP.
  - — LLE (método de redução de dimensionalidade não linear [Locally Linear Embedding]).
  - — t-SNE (método de incorporação estocástica de vizinhos distribuídos [Distributed Stochastic Neighbor Embedding]).
- Aprendizado de regras por associação
  - — Apriori.
  - — Eclat.

Por exemplo, digamos que você tenha muitos dados sobre os visitantes do seu blog. Você quer executar um algoritmo de *clusterização* com o objetivo de tentar detectar grupos de visitantes semelhantes (Figura 1-8). Em nenhum momento você informa ao algoritmo a qual grupo o visitante pertence: ele encontrará essas relações sem sua ajuda. Por exemplo, ele pode observar que 40% dos visitantes são homens que adoram histórias em quadrinhos e em geral leem seu blog à noite, enquanto 20% são jovens amantes de ficção científica e o visitam durante os finais de semana. Se você utilizar um algoritmo de *clusterização hierárquica*, ele também poderá subdividir cada grupo em grupos menores. Isso pode ajudá-lo a direcionar suas postagens para cada um desses grupos.

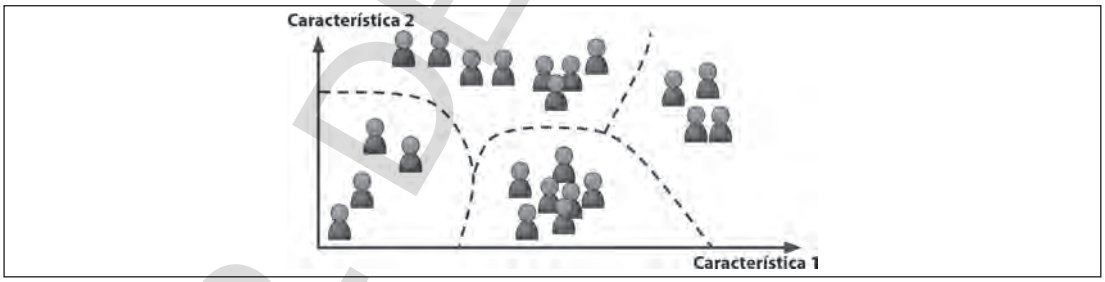

Figura 1-8. Clusterização

Os algoritmos de *visualização* também são bons exemplos de algoritmos de aprendizado não supervisionado: você lhes fornece muitos dados complexos e não rotulados, e eles geram uma representação 2D ou 3D de seus dados que podem ser facilmente plotados (Figura 1-9). Esses algoritmos tentam preservar o máximo da estrutura (por exemplo, tentam impedir que clusters separados no espaço de entrada se sobreponham na visualização), a fim de que você possa entender como os dados estão organizados e talvez identificar padrões inesperados.

A *redução da dimensionalidade* é uma tarefa relacionada cujo objetivo é simplificar os dados sem perder muita informação. Para tal, você pode fazer o merge de diversas características correlacionadas em uma. Por exemplo, a quilometragem de um carro pode estar bastante correlacionada com seu tempo de uso, de modo que o algoritmo da redução de dimensionalidade fará o merge em uma característica que representa o desgaste do carro. Isso se chama de *extração de características*.

Não raro, é uma boa ideia tentar reduzir a dimensão dos dados de treinamento usando um algoritmo de redução de dimensionalidade e fornecê-lo a outro algoritmo do aprendizado de máquina (como um algoritmo de aprendizado supervisionado). Esse algoritmo será executado mais rapidamente, os dados ocuparão menos espaço em disco e na memória e, em alguns casos, podem ter um melhor desempenho também.

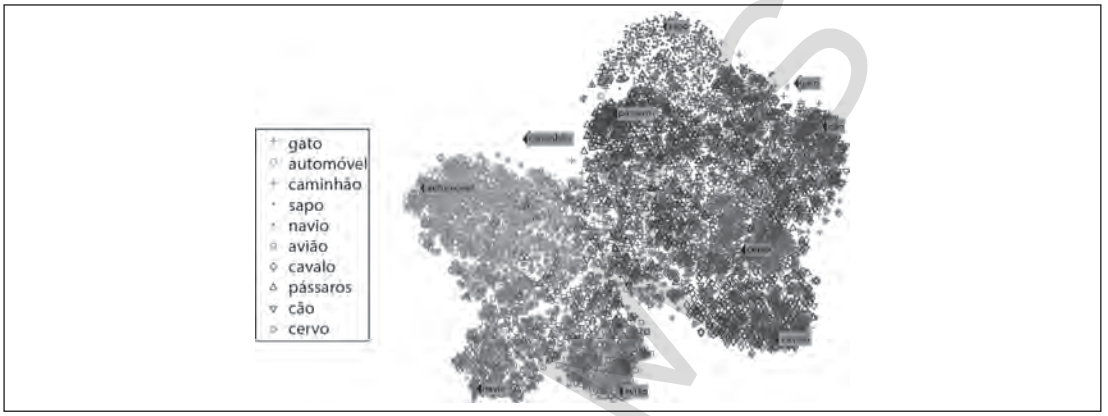

Figura 1-9. Exemplo de uma visualização t-SNE destacando cluster semântico[5]

Outra tarefa fundamental não supervisionada é a *detecção de anomalias* — por exemplo, a detecção de transações incomuns em cartões de crédito para evitar fraudes, identificar defeitos de fabricação ou remover automaticamente outliers de um conjunto de dados, antes de fornecê-lo a outro algoritmo de aprendizado. Na maioria das vezes, exibe-se o sistema em instâncias normais durante o treinamento, por isso ele aprende a reconhecê-las e, quando vê uma instância nova, é capaz de afirmar se ela parece normal ou se é uma provável anomalia (veja a Figura 1-10). Uma tarefa bem semelhante é a *detecção de novidade*: visa detectar instâncias novas que parecem diferentes de todas as instâncias no conjunto de treinamento. Isso exige um conjunto de treinamento muito "limpo", desprovido de qualquer instância que você gostaria que o algoritmo detectasse. Por exemplo, se você tem milhares de fotos de cães e 1% delas representa Chihuahuas, um algoritmo de detecção de novidade não deve tratar as novas fotos de Chihuahuas como novidades. Em contrapartida, algoritmos de detecção de anomalias podem considerar esses cães tão raros e tão diferentes de outros que provavelmente os classificariam como anomalias (nada contra os Chihuahuas).

5 Observe como os animais estão bem separados dos veículos, como os cavalos estão próximos aos cervos, mas longe dos pássaros, e assim por diante. Imagem reproduzida com permissão de Richard Socher et al., "Zero-Shot Learning Through CrossModal Transfer", *Proceedings of the 26th International Conference on Neural Information Processing Systems* 1 (2013): 935–943.

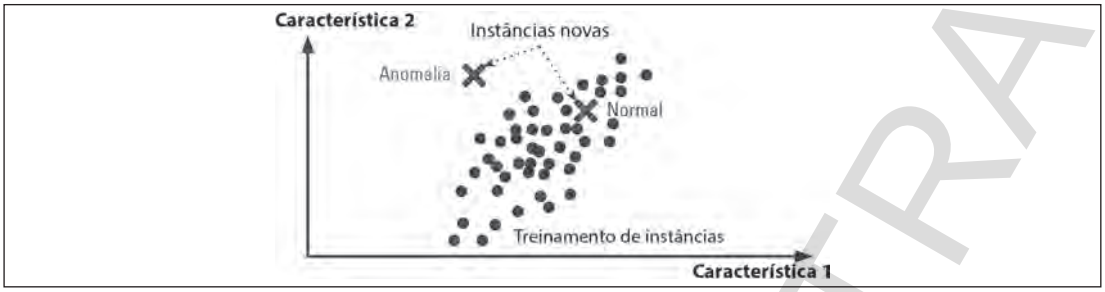

Figura 1-10. Detecção de anomalia

Finalmente, outra tarefa comum não supervisionada é o *aprendizado de regras por associação*, cujo objetivo é investigar grandes quantidades de dados e descobrir relações interessantes entre os atributos. Por exemplo, suponha que você seja dono de um supermercado. Executar uma regra de associação em seus registros de vendas pode revelar que as pessoas que compram molho de churrasco e batatas fritas também estão propensas a comprar carnes. Assim sendo, talvez você queira colocar esses itens próximos uns dos outros.

Aprendizado semissupervisionado

Como rotular os dados geralmente consome tempo e dinheiro, você terá uma grande quantidade de instâncias não rotuladas e poucas instâncias rotuladas. Alguns algoritmos podem lidar com dados parcialmente rotulados. Isso se chama *aprendizado semissupervisionado* (Figura 1-11).

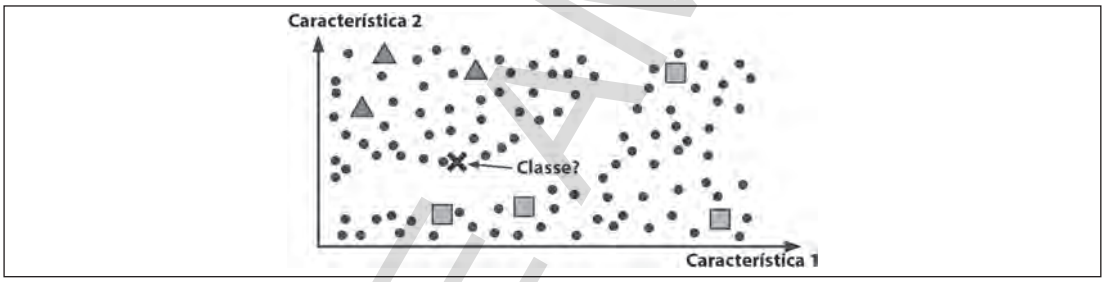

Figura 1-11. Aprendizado semissupervisionado com duas classes (triângulos e quadrados): os exemplos não rotulados (círculos) ajudam a classificar uma instância nova (a cruz) na classe triângulo em vez de na classe quadrado, ainda que esteja mais próxima dos quadrados rotulados

Alguns serviços de hospedagem de fotos, como o Google Fotos, são bons exemplos disso. Ao fazer o upload de todas as fotos de família, o aplicativo reconhecerá automaticamente que a mesma pessoa (A) aparece nas fotos 1, 5 e 11, enquanto outra pessoa (B) aparece nas fotos 2, 5 e 7. Essa é a parte não supervisionada do algoritmo (clusterização). Agora, o sistema apenas precisa que você informe quem são essas pessoas. Acrescente somente um rótulo por pessoa[6] e ele será capaz de nomear todas, o que é útil para pesquisar fotos.

A maior parte dos algoritmos de aprendizado semissupervisionado são combinações de algoritmos supervisionados e não supervisionados. Por exemplo, as *redes neurais de crenças profundas* (DBNs) são baseadas em componentes não supervisionados, as chamadas *máquinas restritas de Boltzmann* (RBMs), empilhados uns em cima dos outros. As RBMs são treinadas sequencialmente de forma não supervisionada, logo todo o sistema se aperfeiçoa usando técnicas de aprendizado supervisionado.

6 Isso quando o sistema funciona perfeitamente. Na prática, muitas vezes ele cria alguns clusters por pessoa e, às vezes, mistura duas pessoas parecidas, logo é necessário fornecer alguns rótulos por pessoa e limpar manualmente alguns clusters.

Aprendizado por reforço

Já o *aprendizado por reforço* é uma criatura bem diferente. O sistema de aprendizado, chamado de *agente* nesse contexto, pode assistir o ambiente, selecionar e executar ações e obter *recompensas* em troca (ou *penalidades* na forma de recompensas negativas, conforme mostrado na Figura 1-12). Ele deve aprender sozinho qual é a melhor estratégia, chamada de *política*, para obter o maior número de recompensas ao longo do tempo. Uma política define qual ação o agente deve escolher quando está em determinada situação.

Por exemplo, muitos robôs implementam algoritmos de aprendizado por reforço para aprender a andar. O programa AlphaGo da DeepMind também é um bom exemplo de aprendizado por reforço: ele apareceu nas manchetes em maio de 2017 quando venceu o campeão mundial Ke Jie no jogo Go. Ele aprendeu sua política de vitória analisando milhões de jogos e depois jogando muitos jogos contra si mesmo. Repare que o aprendizado foi desativado durante os jogos contra o campeão. O AlphaGo estava somente usando a política que havia aprendido.

Figura 1-12. Aprendizado por reforço

Aprendizado em batch e online

Outro critério utilizado para classificar os sistemas de aprendizado de máquina é se o sistema pode ou não aprender de forma incremental a partir da entrada de um fluxo de dados.

Aprendizado em batch (por ciclo)

No *aprendizado em batch*, o sistema é incapaz de aprender de forma incremental: ele deve ser treinado usando todos os dados disponíveis. Via de regra, isso demandará muito tempo e recursos computacionais, portanto normalmente é realizado offline. Primeiro, o sistema é treinado, em seguida é implementado em produção e roda sem aprender mais nada, somente aplicando o que aprendeu. Isso se chama *aprendizado offline*.

Caso deseje que um sistema de aprendizado em batch tenha acesso a dados novos (como um novo tipo de spam), você precisa treinar uma nova versão do sistema a partir do zero no conjunto completo de dados (e não apenas com os dados novos, mas também com os antigos) e, em seguida, descontinuar o sistema antigo e substituí-lo pelo novo. Felizmente, todo o processo de treinamento, avaliação e disponibilização de um sistema de aprendizado de máquina pode ser facilmente automatizado (como mostrado na Figura 1-3). Assim sendo, mesmo um sistema de aprendizado em batch pode se adaptar às mudanças. Basta atualizar os dados e treinar uma nova versão do sistema a partir do zero sempre que necessário.

É uma solução simples que geralmente funciona bem, mas o treinamento usando o conjunto completo de dados pode levar muitas horas; portanto, você normalmente treina um novo sistema apenas a cada 24 horas ou semanalmente. Caso seu sistema precise se adaptar a dados que mudam rapidamente (por exemplo, prever os preços das ações), você precisa de uma solução mais responsiva.

Além disso, o treinamento no conjunto completo de dados exige mais recursos computacionais (CPU, espaço de memória, espaço em disco, E/S do disco, E/S de rede etc.). Se você tem muitos dados e seu sistema é automatizado para treinar todos os dias a partir do zero, esse processo custará uma fortuna. Se a quantidade de dados for gigantesca, talvez seja impossível utilizar um algoritmo de aprendizado em batch.

Por fim, caso o sistema precise aprender de forma autônoma e tenha recursos limitados (por exemplo, um aplicativo de smartphone ou um veículo do tipo rover explorando a superfície de Marte), fazer o upload de grandes quantidades de dados de treinamento e usar os inúmeros recursos para treinar por dias e horas a fio provocaria um bug do tipo showstopper. Felizmente, uma opção melhor em todos esses casos seria utilizar algoritmos capazes de aprender de forma incremental.

Aprendizado online (incremental)

No *aprendizado online*, é possível treinar o sistema incrementalmente, fornecendo as instâncias de dados de forma sequencial, individual ou em pequenos grupos, chamados de *mini-batches*. Cada etapa do aprendizado é rápida e tem um custo baixo, assim o sistema pode aprender instantaneamente os dados novos em tempo real, assim que eles entram (veja a Figura 1-13).

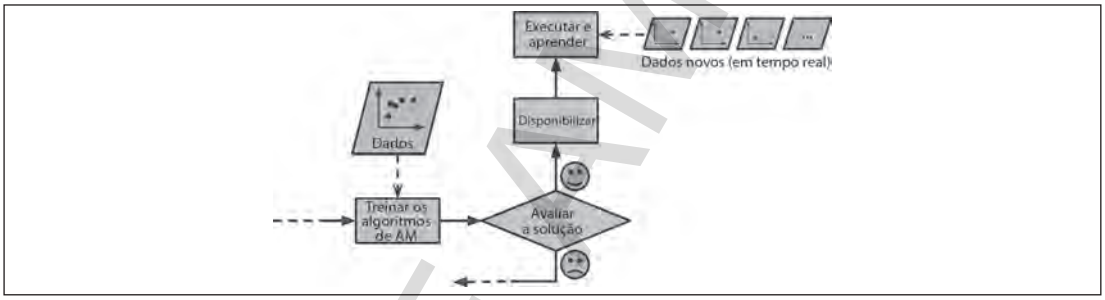

Figura 1-13. No aprendizado online, um modelo é treinado e disponibilizado em produção e, em seguida, continua aprendendo à medida que novos dados são recebidos

O aprendizado online é excelente para sistemas que recebem dados em um fluxo contínuo (por exemplo, preços das ações) e precisam se adaptar a mudanças rápido ou autonomamente. Também é uma boa opção se seus recursos computacionais são limitados: uma vez que um sistema de aprendizado online aprendeu novas instâncias de dados, ele não precisa mais delas, logo você pode descartá-las (a menos que queira fazer o rollback para um estágio anterior e "voltar" os dados). O que pode economizar bastante espaço.

Os algoritmos de aprendizado online também podem ser utilizados para treinar sistemas em grandes conjuntos de dados que não cabem na memória principal de uma máquina (isso se chama aprendizado *out-of-core*). O algoritmo faz o upload de parte dos dados, executa uma etapa do treinamento nesses dados e repete o processo até que ele tenha sido executado em todos os dados (veja a Figura 1-14).

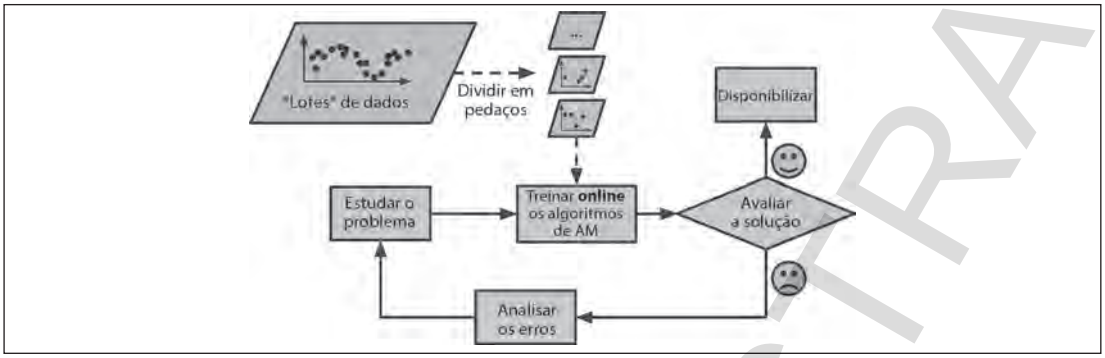

Figura 1-14. Usando o aprendizado online para lidar com grandes conjuntos de dados

O aprendizado out-of-core geralmente é feito offline (ou seja, não no sistema ativo), portanto o aprendizado online pode ser um nome confuso. Pense nisso como um *aprendizado incremental*.

Um parâmetro importante dos sistemas de aprendizado online é a rapidez com que eles devem se adaptar às mudanças dos dados: isso se chama *taxa de aprendizagem*. Caso defina uma alta taxa de aprendizado, o sistema se adaptará rapidamente aos dados novos, mas também será propenso a se esquecer rapidamente dos antigos (você não quer que um filtro de spam sinalize apenas os tipos mais recentes de spam). Por outro lado, se você definir uma baixa taxa de aprendizado, o sistema terá mais inércia; ou seja, aprenderá mais devagar, porém também será menos suscetível ao apontar novos dados ou sequências de pontos de dados não representativos (outliers).

Um grande desafio no aprendizado online é que, se fornecermos dados ruins para o sistema, o desempenho diminuirá gradualmente. Se estamos falando de um sistema em tempo real, os clientes perceberão. Por exemplo, talvez os dados ruins sejam oriundos de um sensor com mau funcionamento em um robô, ou de alguém que envia um spam a um mecanismo de busca com o intuito de alcançar uma classificação alta nos resultados de pesquisa. Para mitigar esse risco, você precisa monitorar de perto o sistema e desativar o aprendizado imediatamente, caso identifique uma queda no desempenho. Talvez você também queira monitorar os dados de entrada e tomar providências quanto aos dados anormais (por exemplo, usando um algoritmo de detecção de anomalias).

Aprendizado baseado em instâncias versus aprendizado baseado em modelo

Outra forma de categorizar os sistemas de aprendizado de máquina é por meio da *generalização*. A maioria das tarefas de aprendizado de máquina faz predições. Isso implica que, dada uma série de exemplos de treinamento, o sistema precisa ser capaz de fazer boas predições para (generalizar) exemplos que nunca viu antes. Um bom rendimento do desempenho dos dados de treinamento é ótimo, mas não é o bastante; o verdadeiro objetivo é ter um bom desempenho em instâncias novas.

Existem duas abordagens principais no que diz respeito à generalização: aprendizado baseado em instâncias e aprendizado baseado em modelo.

Aprendizado baseado em instância

Possivelmente, a forma mais comum de aprendizado é a memorização. Se você desenvolvesse um filtro de spam dessa forma, ele apenas sinalizaria todos os e-mails idênticos em relação aos e-mails já marcados pelos usuários — não seria a pior solução, mas também não é uma das melhores.

Em vez de marcar somente e-mails idênticos em comparação aos e-mails de spam conhecidos, o filtro de spam pode ser programado para sinalizar também e-mails semelhantes aos e-mails conhecidos de spam. Isso exige uma *medida de similaridade* (SM) entre dois e-mails. Uma medida de similaridade (muito básica) entre dois e-mails poderia ser contar o número de palavras que eles têm em comum. O sistema marcaria um e-mail como spam se tivesse muitas palavras em comum com um e-mail de spam conhecido.

Isso se chama de *aprendizado baseado em instância*: o sistema aprende os exemplos por meio da memorização e depois generaliza em novos casos, ao empregar uma medida de similaridade a fim de compará-los a outros exemplos aprendidos (ou um conjunto deles). Por exemplo, na Figura 1-15, a instância nova seria classificada como um triângulo porque a maioria das instâncias semelhantes pertence a essa classe.

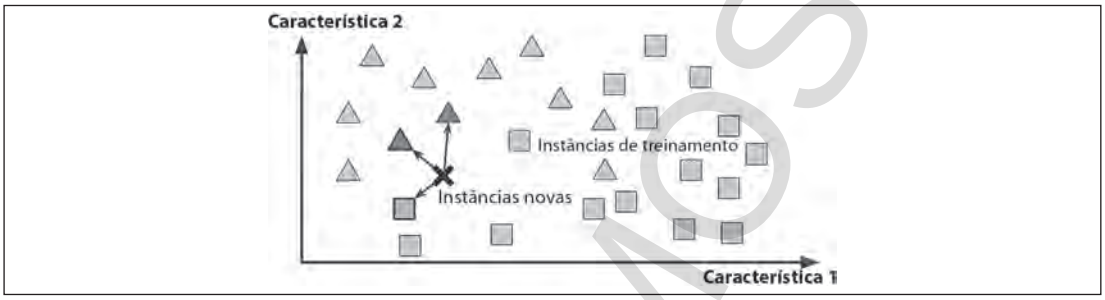

Figura 1-15. Aprendizado baseado em instância

Aprendizado baseado em modelo

Outro modo de generalização de um conjunto de exemplos seria construir um modelo desses exemplos e usá-lo para fazer *predições*. Isso se chama *aprendizado baseado em modelo* (Figura 1-16).

Por exemplo, digamos que você queira saber se o dinheiro traz felicidade às pessoas e faz o download dos dados do Índice de Vida Melhor no site da OCDE (*https://homl.info/4*), das estatísticas sobre o produto interno bruto (PIB) e da renda per capita do site do FMI (*https://homl.info/5*). Depois, você junta as tabelas e classifica o PIB por renda per capita. A Tabela 1-1 mostra um trecho do que você obtém.

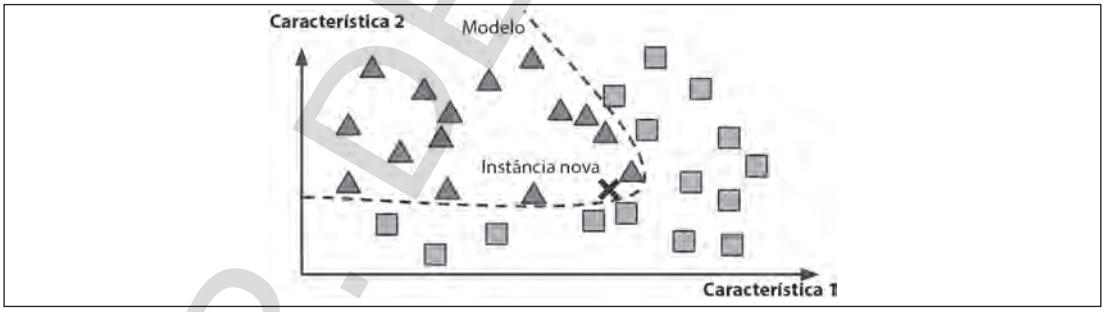

Figura 1-16. Aprendizado baseado em modelo

Tabela 1-1. O dinheiro traz felicidade às pessoas?

País	PIB por per capita (USD)	Satisfação de vida
Hungria	12.240	4,9
Coreia	27.195	5,8
França	37.675	6,5
Austrália	50.962	7,3
Estados Unidos	55.805	7,2

Vamos fazer um gráfico dos dados para esses países (Figura 1-17).

Ao que tudo indica, temos uma tendência! Ainda que os dados se classifiquem como *ruídos* (ou seja, parcialmente aleatórios), parece que a satisfação de vida aumenta mais ou menos linearmente à medida que o PIB per capita do país cresce. Desse modo, você modela a satisfação de vida como uma função linear do PIB per capita. Isso se chama *seleção do modelo*: você selecionou um modelo linear de satisfação de vida com apenas um atributo, o PIB per capita (Equação 1-1).

Equação 1-1. Um simples modelo linear

$$\text{satisfação\_de\_vida} = \theta_0 + \theta_1 \times \text{PIB\_per\_capita}$$

Esse modelo tem dois *parâmetros de modelo*, θ_0 e θ_1.[7] Ao ajustá-los, você pode fazer com que seu modelo represente qualquer função linear, conforme mostrado na Figura 1-18.

Antes de utilizar seu modelo, você precisa definir os valores dos parâmetros θ_0 e θ_1.

Como saber quais valores funcionarão melhor com o modelo? Para responder a essa pergunta, é necessário especificar uma medida de desempenho. Você pode definir uma *função de utilidade* (ou *função de avaliação*) que calcula o quanto o seu modelo é bom, ou uma *função de custo*, que calcula o quanto ele é ruim. Para problemas de regressão linear, as pessoas geralmente utilizam uma função de custo que calcula a distância entre as predições do modelo linear e os exemplos de treinamento; o objetivo é minimizar essa distância.

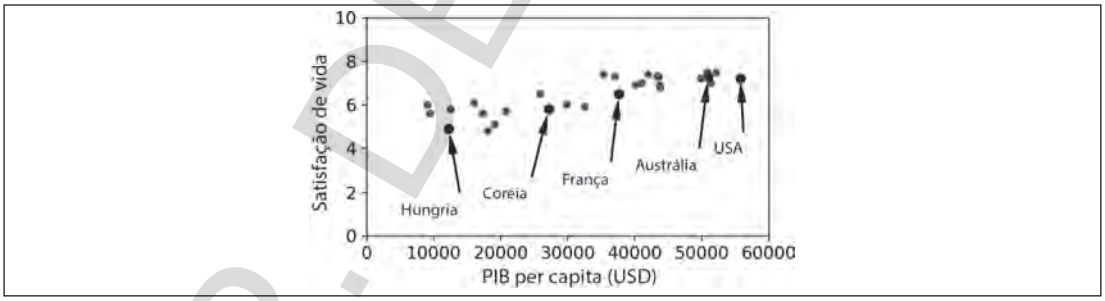

Figura 1-17. Você enxerga uma tendência?

7 Por convenção, a letra grega θ (teta) é usada com frequência para representar os parâmetros do modelo.

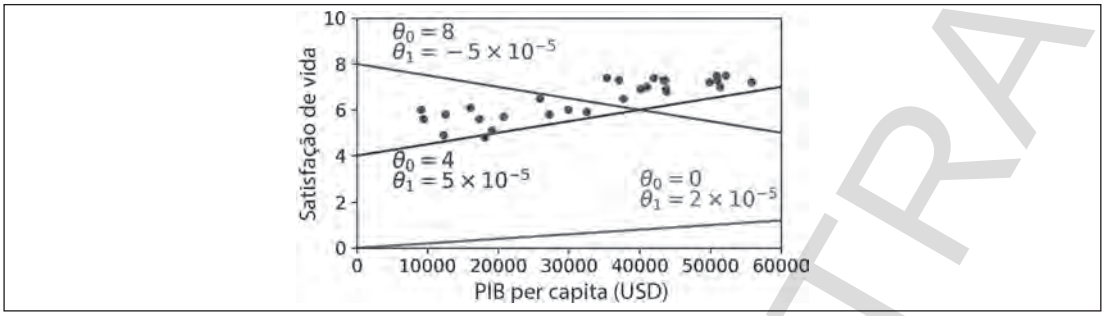

Figura 1-18. Alguns modelos lineares possíveis

Nesse momento, o algoritmo de regressão linear entra em cena: você o alimenta com seus exemplos de treinamento e ele identifica os parâmetros do modelo linear que melhor se adéquam aos dados. Isso se chama *treinar* o modelo. Em nosso caso, o algoritmo descobre que os valores dos parâmetros ideais são e $\theta_0 = 4{,}85$ e $\theta_1 = 4{,}91 \times 10^{-5}$.

Inexplicavelmente, a mesma palavra "modelo" pode se referir a um *tipo de modelo* (por exemplo, regressão linear), a um *modelo de arquitetura totalmente específico* (por exemplo, regressão linear com uma entrada e uma saída) ou ao *modelo final treinado*, pronto para ser usado em predições (por exemplo, regressão linear com uma entrada e uma saída, usando $\theta_0 = 4{,}85$ e $\theta_1 = 4{,}91 \times 10–5$). A seleção do modelo consiste em escolher o tipo de modelo especificando sua arquitetura por completo. Treinar um modelo significa executar um algoritmo a fim de identificar os parâmetros do modelo que melhor se adequem aos dados de treinamento (e, quem sabe, fazer boas predições a partir dos dados novos).

Agora, o modelo se ajusta o mais próximo possível dos dados do treinamento (para um modelo linear), como pode ver na Figura 1-19.

Você finalmente está pronto para executar o modelo e fazer predições. Por exemplo, digamos que você quer saber o nível de felicidade dos cipriotas, e os dados da OCDE não têm a resposta. Felizmente, você pode utilizar o seu modelo para realizar uma boa predição: consulte o PIB per capita do Chipre, encontre US$22.587 e, em seguida, aplique seu modelo e verifique a satisfação de vida, que estará mais ou menos próxima de 4,85 + 22.587 × 4,91 × 10-5 = 5,96.

Para aguçara curiosidade, o Exemplo 1-1 mostra o código Python que carrega os dados, os prepara,[8] cria um gráfico de dispersão para visualização, treina um modelo linear e ainda faz uma predição.[9]

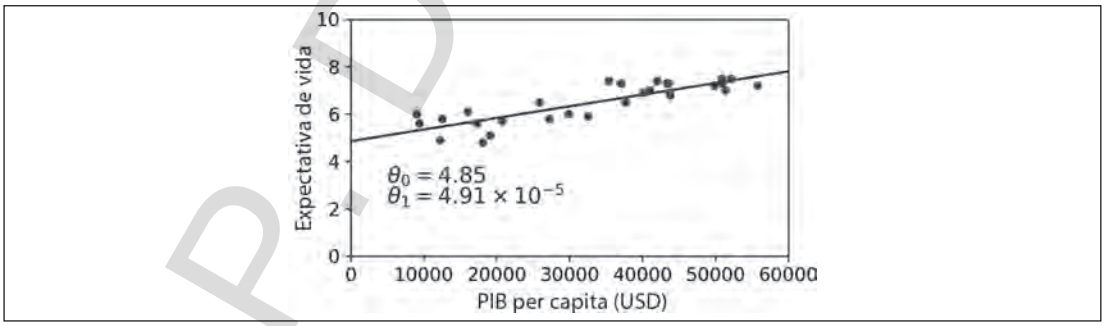

Figura 1-19. O modelo linear que melhor se ajusta aos dados de treinamento

8 Neste livro, não mostramos a definição da função `prepare_country_stats()`, (veja o Jupyter Notebook deste capítulo, se quiser saber dos detalhes sórdidos). É apenas um código entediante usando a biblioteca Pandas que combina os dados de satisfação de vida da OCDE com os dados do PIB per capita do FMI.

9 Não se preocupe se não entender o código ainda; apresentaremos a Scikit-Learn nos próximos capítulos.

Exemplo 1-1. Treinamento e execução de um modelo linear usando a Scikit-Learn

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model

# Carrega os dados
oecd_bli = pd.read_csv("oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv("gdp_per_capita.csv",thousands=',',delimiter='\t',
                             encoding='latin1', na_values="n/a")

# Prepara os dados
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Visualiza os dados
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

# Seleciona um modelo linear
model = sklearn.linear_model.LinearRegression()

# Treina o modelo
model.fit(X, y)

# Efetua uma predição para o Chipre
X_new = [[22587]] # PIB per capita do Chipre
print(model.predict(X_new)) # Entradas [[ 5.96242338]]
```

Caso tivesse usado um algoritmo de aprendizado baseado em instâncias, descobriria que a Eslovênia tem o PIB per capita mais próximo do Chipre (US$20.732), e, como os dados da OCDE nos informam que a satisfação de vida dos eslovenos é 5,7, você teria previsto uma satisfação de vida de 5,7 para o Chipre. Se considerar os pontos essenciais e observar os dois países mais próximos, encontrará Portugal e Espanha com satisfações de vida de 5,1 e 6,5, respectivamente. Ao calcular a média desses três valores, você obtém 5,77, bastante próximo da predição baseada em modelo. Esse simples algoritmo de regressão se chama *K-ésimo vizinho mais próximo* (neste exemplo, $k = 3$).

Para substituir o modelo de regressão linear pelo algoritmo de regressão K-ésimo vizinho mais próximo no código anterior, simplesmente troque estas suas linhas:

```
import sklearn.linear_model
model = sklearn.linear_model.LinearRegression()
```

por estas duas:

```
import sklearn.neighbors
model = sklearn.neighbors.KNeighborsRegressor(
    n_neighbors=3)
```

Se tudo correu bem, seu modelo fará boas predições. Caso contrário, talvez seja necessário usar mais atributos (índice de emprego, saúde, poluição do ar etc.), obter mais dados de treinamento ou dados com uma qualidade melhor ou talvez selecionar um modelo mais poderoso (por exemplo, um modelo de regressão polinomial).

Em resumo:

- Você estudou os dados.
- Selecionou o modelo.
- Treinou o modelo nos dados de treinamento (ou seja, o algoritmo de aprendizado procurou os valores dos parâmetros do modelo que minimizam uma função de custo).
- E, por último, aplicou o modelo para fazer predições em novos casos (isso se chama *inferência*), na expectativa de que esse modelo fizesse boas generalizações.

O que acabou de ver é um típico projeto de aprendizado de máquina. No Capítulo 2, você acompanhará de perto como é executar um projeto, do começo ao fim.

Já caminhamos muito: agora você já sabe o que é aprendizado de máquina, sua importância, quais são algumas das categorias mais comuns de sistemas de AM e como é um típico fluxo de trabalho do projeto. Agora, veremos o que pode sair de errado no aprendizado, impossibilitando-o de fazer predições certeiras.

Principais Desafios do Aprendizado de Máquina

Em síntese, uma vez que a tarefa principal é selecionar um algoritmo de aprendizado e treiná-lo em alguns dados, duas coisas podem dar errado: "algoritmos ruins" e "dados ruins". Começaremos com exemplos de dados ruins.

Quantidade insuficiente de dados de treinamento

Para que uma criança aprenda o que é uma maçã, é necessário apontar para a fruta e dizer "maçã" (repetindo inúmeras vezes esse procedimento). Agora, a criança consegue reconhecer maçãs de todos os tipos, cores e formas. Genial.

O aprendizado de máquina ainda não chegou a esse nível; é preciso uma grande quantidade de dados para que a maioria dos algoritmos de aprendizado de máquina funcione corretamente. Ainda que os problemas sejam bem simples, você normalmente precisa de centenas de exemplos, e para problemas complexos, como reconhecimento de imagem ou voz, talvez sejam necessários milhões de exemplos (a menos que seja possível reutilizar partes de um modelo existente).

Dados de Treinamento Não Representativos

Com o objetivo de executar boas generalizações, é imprescindível que os dados de treinamento sejam representações dos novos casos. Isso só vale se você utilizar o aprendizado baseado em instâncias ou o aprendizado baseado em modelo.

Por exemplo, o conjunto de países que utilizamos anteriormente para treinar o modelo linear não era perfeitamente representativo; faltavam alguns países. A Figura 1-21 exemplifica como são os dados quando os países que faltam são adicionados.

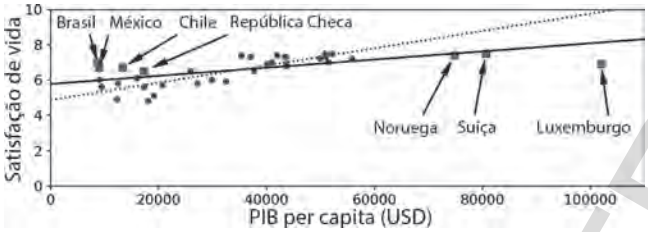

Figura 1-21. Uma amostra de treinamento mais representativa

Caso treine um modelo linear nesses dados, obterá uma linha sólida, ao passo que o modelo antigo é representado por uma linha pontilhada. Conforme se observa, acrescentar alguns países faltantes não só altera significativamente o modelo, como deixa claro que talvez um modelo linear tão simples nunca funcione bem. Aparentemente, os países muito ricos não são mais felizes do que países moderadamente ricos (na verdade, parecem mais infelizes), e vice-versa, alguns países pobres parecem ser mais felizes do que muitos países ricos.

Ao usar um conjunto de treinamento não representativo, treinamos um modelo que dificilmente fará predições exatas, sobretudo para países muito pobres e muito ricos.

É de suma importância utilizar um conjunto de treinamento representativo nos casos em que desejamos generalizar. No entanto, a coisa é mais difícil do que parece: se a amostra for muito pequena, existirá um *ruído de amostragem* (ou seja, dados não representativos como resultado do acaso), mas mesmo as amostras muito grandes podem não ser representativas se o método de amostragem for falho. Isso é chamado de *viés de amostragem*.

A Eficácia Irracional dos Dados

Em um famoso artigo (*https://homl.info/6*) publicado em 2001, os pesquisadores Michele Banko e Eric Brill, da Microsoft, demonstraram que algoritmos de aprendizado de máquina bastante distintos, inclusive os mais simples, tiveram um desempenho quase idêntico em um problema complexo de desambiguação[10] de linguagem natural após serem alimentados com dados suficientes (conforme você pode ver na Figura 1-20).

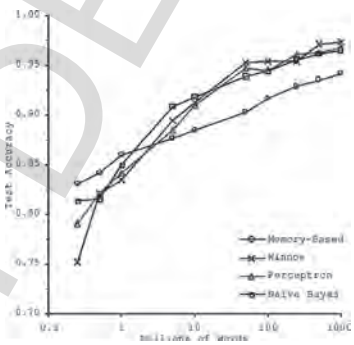

Figura 1-20. A importância dos dados versus algoritmos[11]

10 Por exemplo, em inglês, saber quando escrever "to", "two", ou "too" dependendo do contexto.

11 Figura reproduzida com permissão de Banko e Brill, "Scaling to Very Very Large Corpora for Natural Language Disambiguation", *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics* (2001): 26–33.

Como os autores alegam: "Esses resultados sugerem que talvez possamos reconsiderar o custo-benefício entre gastar tempo e dinheiro no desenvolvimento de algoritmos ou empregá-los no desenvolvimento de corpus."

A ideia de que os dados são mais importantes do que os algoritmos no que se refere aos problemas complexos foi popularizada por Peter Norvig et al. em um artigo intitulado "The Unreasonable Effectiveness of Data" ["A Eficácia Irracional dos Dados", em tradução livre] (*https://homl.info/7*), publicado em outubro de 2009.[12] No entanto, deve-se salientar que os conjuntos de dados pequenos e médios ainda são muito comuns, e nem sempre é fácil ou barato obter dados extras de treinamento — portanto, não abra mão dos algoritmos ainda.

Dados de baixa qualidade

Obviamente, se seus dados de treinamento estiverem cheios de erros, outliers e ruídos (por exemplo, devido a medições de baixa qualidade), o sistema terá mais dificuldade para detectar os padrões básicos; logo, é menos provável que seu sistema funcione bem. Muitas vezes vale a pena dedicar um tempo limpando os dados de treinamento. A verdade é que a maioria dos cientistas de dados gasta uma parte significativa do tempo fazendo exatamente isso. Por exemplo:

- Se algumas instâncias são claramente outliers, apenas descartá-las pode ajudar, ou você pode tentar corrigir os erros manualmente.
- Caso falte algumas características para algumas instâncias (por exemplo, 5% dos seus clientes não especificaram sua idade), você deve decidir se deseja ignorar completamente esse atributo, se deseja ignorar essas instâncias, preencher os valores ausentes (por exemplo, com a média da idade), ou treinar um modelo com a característica e um modelo sem, e assim por diante.

Características Irrelevantes

Como diz o ditado: entra lixo, sai lixo. Seu sistema só será capaz de aprender se os dados de treinamento tiverem características relevantes suficientes e poucas características irrelevantes. Uma parte imprescindível do sucesso de um projeto de aprendizado de máquina é criar um bom conjunto de características para o treinamento, processo chamado de *feature engeneering* (ou engenharia de features) que envolve os seguintes passos:

- *Seleção de características* (selecionar as características mais úteis para treinamento entre as características existentes).
- *Extração de características* (combinar características existentes a fim de obter as mais uteis — como vimos anteriormente, os algoritmos de redução de dimensionalidade podem ajudar).
- Criação de novas características ao coletar dados novos.

Agora que analisamos diversos exemplos ruins de dados, vejamos alguns exemplos ruins de algoritmos.

12 Peter Norvig et al., "The Unreasonable Effectiveness of Data", *IEEE Intelligent Systems* 24, no. 2 (2009): 8–12.

Exemplos de Viés de Amostragem

Talvez o exemplo mais famoso de viés de amostragem tenha acontecido durante as eleições presidenciais dos EUA em 1936, na disputa de Landon contra Roosevelt: a *Literary Digest* conduziu uma ampla pesquisa, enviando cartas pelo correio para cerca de 10 milhões de pessoas. Obteve um retorno de 2,4 milhões de respostas e previu com grande confiança que Landon deveria obter 57% dos votos. Mas, na verdade, Roosevelt venceu as eleições com 62% dos votos. A falha aconteceu no método de amostragem da *Literary Digest*:

- Primeiro, ao obter os endereços para o envio das pesquisas, a *Literary Digest* utilizou listas telefônicas, de assinantes, de membros de clubes, entre outras. Todas elas costumam favorecer pessoas mais ricas, mais propensas a votar em republicanos (por isso, Landon).
- Em segundo lugar, menos de 25% das pessoas responderam à enquete. Novamente, ao descartar pessoas que não se importam muito com a política, pessoas que não gostam da *Literary Digest* e outros grupos-chave, é introduzido um viés de amostragem, chamado de *viés de ausência de resposta*.

Vejamos outro exemplo: digamos que você deseja criar um sistema para o reconhecimento de vídeos de música funk. Uma forma de desenvolvê-lo seria pesquisar "música funk" no YouTube e utilizar os vídeos resultantes. Mas isso pressupõe que o mecanismo de pesquisa do YouTube retornará um conjunto de vídeos que representa todos os vídeos de música no YouTube. Provavelmente, os artistas populares terão resultados tendenciosos na pesquisa (e, se você mora no Brasil, receberá muitos vídeos de "funk carioca" que não se parecem nada com James Brown). Por outro lado, de que outra forma você consegue obter um grande conjunto de treinamento?

Sobreajuste dos Dados de Treinamento

Digamos que você está visitando um país estrangeiro e o taxista lhe cobra os olhos da cara por uma corrida. Talvez você caia na tentação de dizer que todos os motoristas de táxi do país são ladrões. Generalizar as coisas exageradamente é algo que nós, humanos, fazemos com muita frequência, e infelizmente as máquinas podem cair na mesma armadilha se não tomarmos cuidado. No aprendizado de máquina, isso é chamado de sobreajuste: significa que o modelo funciona bem nos dados de treinamento, mas não generaliza tão bem.

A Figura 1-22 exemplifica um modelo polinomial de alto nível de satisfação de vida que se sobreajusta acentuadamente nos dados de treinamento. Ainda que apresente um desempenho bem melhor nos dados de treinamento do que no modelo linear simples, você confiaria em suas predições?

Modelos complexos como redes neurais profundas podem detectar padrões sutis nos dados, mas se o conjunto de treinamento é ruidoso ou se é muito pequeno (e introduz ruído na amostragem), o modelo provavelmente detectará padrões no próprio ruído. É óbvio que esses padrões não serão generalizados para novas instâncias. Por exemplo, digamos que você alimenta o modelo de satisfação de vida com muitos outros atributos, incluindo alguns não informativos, como o nome do país. Nesse caso, um modelo complexo poderia detectar padrões como o fato de que todos os países com um *W* em seu nome em inglês têm uma satisfação de vida superior a 7: New Zealand [Nova Zelândia] (7,3), Norway [Noruega] (7,4), Sweden [Suécia] (7,2) e Switzerland [Suíça] (7,5). Você confiaria nessa regra que também generalizaria países como Rwanda [Ruanda] ou Zimbabwe [Zimbábue]? É evidente que esse padrão nos dados ocorreu por mero acaso, contudo o modelo não tem como saber se um padrão é real ou simplesmente resultado de ruído nos dados.

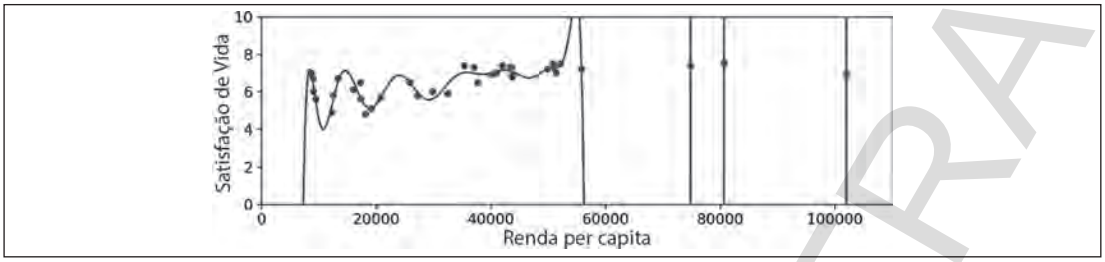

Figura 1-22. Sobreajuste nos dados de treinamento

O sobreajuste acontece quando o modelo é muito complexo em relação à quantidade e ao ruído dos dados de treinamento. As possíveis soluções são:

- Simplificar o modelo ao selecionar um com menos parâmetros (por exemplo, um modelo linear em vez de um modelo polinomial de alto nível), reduzindo o número de atributos nos dados de treinamento ou restringindo o modelo.
- Coletar mais dados de treinamento.
- Reduzir o ruído nos dados de treinamento (por exemplo, corrigir erros de dados e remover outliers).

Chamamos de regularização quando restringimos um modelo para simplificar e reduzir o risco de sobreajuste. Por exemplo, o modelo linear que definimos anteriormente tem dois parâmetros, θ_0 e θ_1. Isso fornece ao algoritmo de aprendizado dois graus de liberdade para adaptar o modelo aos dados de treinamento: ele pode ajustar tanto a altura (θ_0) quanto a inclinação (θ_1) da linha. Se forçássemos o $\theta_1 = 0$, o algoritmo teria apenas um grau de liberdade e teria muito mais dificuldade em ajustar os dados corretamente: tudo o que ele poderia fazer seria mover a linha para cima ou para baixo, de modo a se aproximar o máximo possível das instâncias de treinamento; assim, você teria uma média. Na verdade, seria um modelo bem simples! Mas, caso permitíssemos que o algoritmo modificasse θ_1 e forçássemos sua redução, o algoritmo apresentaria algo como um e dois graus de liberdade. Ele geraria um modelo mais simples do que aquele com dois graus de liberdade, porém mais complexo do que o modelo com apenas um grau. Queremos encontrar o equilíbrio adequado entre o ajuste perfeito dos dados e a necessidade de manter o modelo simples o bastante para garantir que ele generalize bem.

A Figura 1-23 exemplifica três modelos: a linha pontilhada representa o modelo original que foi treinado nos países representados como círculos (sem os países representados como quadrados); a linha tracejada é o nosso segundo modelo treinado com todos os países (círculos e quadrados); e a linha contínua é um modelo treinado com os mesmos dados do primeiro modelo, mas com uma restrição de regularização. Você pode observar que a regularização forçou o modelo a ter uma inclinação menor: esse modelo não se adéqua aos dados de treinamento (círculos) e ao primeiro modelo, no entanto generaliza melhor em exemplos novos que não foram vistos durante o treinamento (quadrados).

A quantidade de regularização aplicada durante o aprendizado pode ser controlada por meio de um hiperparâmetro. Um hiperparâmetro é um parâmetro de um algoritmo de aprendizado (não do modelo). Como tal, não é afetado pelo próprio algoritmo de aprendizado; deve ser definido antes do treinamento e permanecer constante ao longo dele. Se você definir o hiperparâmetro de regularização com um valor muito alto, obterá um modelo quase plano (uma inclinação próxima a zero); o algoritmo de aprendizado certamente não se sobreajustará nos dados de treinamento e terá menos chances de encontrar uma boa solução. O ajuste dos hiperparâmetros é uma parte importante da construção de um sistema de aprendizado de máquina (você verá um exemplo detalhado no próximo capítulo).

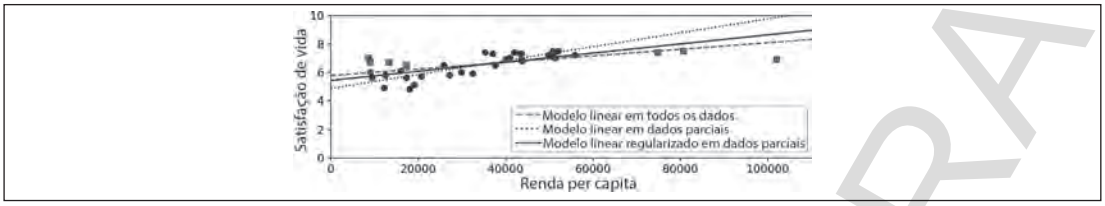

Figura 1-23. A regularização minimiza o risco de sobreajustes

Subajuste dos dados de treinamento

Como você pode imaginar, o subajuste é o oposto de sobreajuste: ocorre quando o modelo é muito simples para o aprendizado da estrutura fundamental dos dados. Por exemplo, um modelo linear de satisfação de vida está propenso a ser subajustado; a realidade é mais complexa do que o modelo, por isso as predições tendem a ser imprecisas mesmo nos exemplos de treinamento.

Veja as principais opções para solucionar esses problemas:

- Selecionar um modelo mais poderoso, com mais parâmetros.
- Alimentar o algoritmo de aprendizado com melhores características (feature engineering).
- Minimizar as restrições no modelo (por exemplo, reduzindo o hiperparâmetro de regularização).

Um passo atrás

Por ora, você já sabe muito sobre o aprendizado de máquina. No entanto, vimos tantos conceitos que você pode estar se sentindo um pouco perdido, então vamos dar um passo atrás e rever o panorama geral:

- Aprendizado de máquina é garantir que as máquinas evoluam em algumas tarefas aprendendo com os dados, em vez de ter que programar explicitamente as regras.
- Existem muitos tipos diferentes de sistemas AM: supervisionados ou não, em batch ou online, baseados em instâncias ou em modelos.
- Em um projeto AM, você coleta dados em um conjunto de treinamento e os fornece para um algoritmo de aprendizado. Se o algoritmo for baseado em modelo, ele ajusta alguns parâmetros para adequar o modelo ao conjunto de treinamento (ou seja, para fazer boas predições no próprio conjunto de treinamento); em seguida, se tudo der certo, também poderá fazer boas predições em novos casos. Se o algoritmo for baseado em instância, ele simplesmente aprende os exemplos de cor e salteado e utiliza uma medida de similaridade para generalizar em instâncias novas.
- O sistema não terá um bom desempenho se o conjunto de treinamento for muito pequeno ou se os dados não forem representativos, ruidosos ou poluídos com características irrelevantes (entra lixo, sai lixo). Por último, o modelo não precisa ser simples demais (subajustado) nem muito complexo (superajustado).

Abordaremos um último tópico importante: uma vez treinado um modelo, você não vai querer apenas "torcer" para que ele generalize casos novos. Você vai querer avaliar e ajustar se necessário. Vejamos como fazer isso.

Teste e Validação

A única forma de mensurar até que ponto um modelo generalizará bem em casos novos é testá-lo na prática. Você pode disponibilizar seu modelo em produção e monitorar a qualidade do desempenho. Isso funciona bem, mas, se o modelo for muito ruim, os usuários reclamarão — e isso não é nada bom.

Uma das melhores opções seria dividir seus dados em dois conjuntos: o conjunto de treinamento e o conjunto de teste. Como os nomes indicam, você treina o modelo utilizando o conjunto de treinamento e o testa usando o conjunto de teste. A taxa de erro nos casos novos se chama erro de generalização (ou erro fora da amostra) e, ao avaliar o modelo no conjunto de teste, você obtém uma estimativa desse erro. Esse valor lhe informa o desempenho do modelo em instâncias que ele nunca trabalhou antes.

Se o erro de treinamento for baixo (ou seja, seu modelo comete alguns erros no conjunto de treinamento), e o erro de generalização for alto, isso significa que o modelo está se sobreajustando aos dados de treinamento.

É comum usar 80% dos dados para treinamento e *separar* 20% para testes. No entanto, isso depende do tamanho do conjunto de dados: se ele tem 10 milhões de instâncias, separar 1% significa que seu conjunto de testes conterá 100 mil instâncias, provavelmente mais que o suficiente para gerar uma boa estimativa do erro de generalização.

Ajuste de hiperparâmetro e seleção de modelo

Avaliar um modelo é bem simples: basta utilizar um conjunto de teste. Agora, suponha que você esteja em dúvida entre dois modelos (um modelo linear e um modelo polinomial): como decidir? Uma opção é treiná-los usando o conjunto de teste e comparar a generalização de ambos.

Agora, suponha que o modelo linear generalize melhor, mas você quer implementar um pouco de regularização para evitar o sobreajuste. A questão é: como escolher o valor do hiperparâmetro de regularização? Uma opção seria treinar 100 modelos diferentes utilizando 100 valores distintos para este hiperparâmetro. Suponha que você identifique o melhor valor de hiperparâmetro que produza um modelo com o menor erro de generalização — digamos apenas 5% de erro. Então, você implementa esse modelo em produção, mas infelizmente ele não funciona tão bem quanto o esperado e gera 15% de erros. O que acabou de acontecer?

O problema é que você calculou o erro de generalização diversas vezes no conjunto de teste e adaptou o modelo e os hiperparâmetros para gerar o melhor modelo para esse conjunto. Isso significa que o modelo provavelmente não funcionará tão bem com os dados novos.

Uma solução comum para esse problema se chama *método holdout* de validação cruzada: basta dividir parte do conjunto de treinamento a fim de avaliar diversos modelos concorrentes e selecionar o melhor. O novo conjunto separado se chama *conjunto de validação* (ou, às vezes, *conjunto de desenvolvimento* ou *dev set*). Em termos concretos, você treina diversos modelos de dados com vários hiperparâmetros no conjunto de treinamento limitado (ou seja, o conjunto de treinamento completo menos o conjunto de validação) e seleciona o modelo com melhor desempenho no conjunto de validação. Após esse processo holdout de validação cruzada, você treina o melhor modelo em todo o conjunto de treinamento (incluindo o conjunto de validação), e isso fornece o modelo final. Por fim, você avalia esse modelo final no conjunto de testes para obter uma estimativa do erro de generalização.

Essa solução geralmente funciona muito bem. No entanto, se o conjunto de validação for muito pequeno, as avaliações do modelo serão imprecisas: você pode acabar selecionando um modelo abaixo do ideal por engano. Por outro lado, se o conjunto de validação for muito grande, o conjunto de treinamento

restante será muito menor que o conjunto de treinamento completo. Por que isso é ruim? Veja bem, como o modelo final será treinado em todo o conjunto de treinamento, o ideal não é comparar modelos concorrentes treinados em um conjunto de treinamento muito pequeno. Seria como selecionar o velocista mais rápido para participar de uma maratona. Uma forma de solucionar esse problema é usar um outro método de validação, o *método k-fold* de validação cruzada, utilizando muitos conjuntos de validação pequenos. Cada modelo é avaliado uma vez por conjunto de validação, após ser treinado no restante dos dados. Ao calcular a média de todas as avaliações de um modelo, você obtém uma medida mais precisa de seu desempenho. Contudo, existe um inconveniente: o tempo de treinamento é multiplicado pelo número de conjuntos de validação.

Incompatibilidade de Dados

Em algumas situações, é fácil obter uma quantidade massiva de dados para treinamento, mas esses dados não representarão perfeitamente os dados que serão usados na produção. Por exemplo, suponha que você queira desenvolver um aplicativo móvel para tirar fotos de flores e determinar automaticamente as espécies. Você pode fazer o download de milhões de fotos de flores na internet, no entanto elas não representarão perfeitamente as fotos que serão tiradas usando o aplicativo em um dispositivo móvel. Talvez você tenha apenas 10 mil fotos representativas (ou seja, realmente tiradas com o aplicativo). Nesse caso, a regra mais importante a ser lembrada é que o conjunto de validação e o conjunto de teste devem ser o mais representativos possível dos dados que você pretende usar em produção; portanto, eles devem ser constituídos exclusivamente de imagens representativas: você pode embaralhá-las e colocar metade no conjunto de validação e a outra metade no conjunto de teste (assegurando que nenhuma, ou quase nenhuma, duplicação esteja em ambos os conjuntos). Mas, após treinar seu modelo nas imagens da internet, caso o desempenho do modelo no conjunto de validação seja decepcionante, você não saberá se isso ocorreu porque o modelo sobreajustou o conjunto de treinamento ou se isso se deve somente à incompatibilidade entre as imagens da internet e as imagens de aplicativos para dispositivos móveis. Uma solução é separar algumas imagens de treinamento (da internet) em outro conjunto, que Andrew Ng chama de *train-dev set [conjunto de desenvolvimento de treinamento]*. Após o modelo ser treinado (no conjunto de treinamento, não no train-dev set), você pode avaliá-lo no the train-dev set. Se apresentar um bom desempenho, o modelo não se sobreajusta no conjunto de treinamento. Se apresentar um desempenho ruim no conjunto de validação, ele deve ser oriundo da incompatibilidade de dados. Você pode tentar resolver esse problema pré-processando as imagens da internet para que fiquem mais parecidas com as fotos que serão tiradas pelo aplicativo móvel e, em seguida, treinar novamente o modelo. Em contrapartida, se o modelo tiver um desempenho duvidoso no train-dev set, ele deve ser sobreajustado no conjunto de treinamento; logo, você deve tentar simplificar ou regularizar o modelo, obter mais dados de treinamento e limpar os dados de treinamento.

Teorema Não Existe Almoço Grátis

Um modelo é uma versão simplificada das observações. As simplificações têm como objetivo deixar de lado os detalhes supérfluos que provavelmente não serão generalizados em instâncias novas. No entanto, para decidir quais dados descartar e quais manter, você deve fazer *suposições*. Por exemplo, um modelo linear supõe que os dados são fundamentalmente lineares e que a distância entre as instâncias e a linha reta é apenas o ruído, que certamente pode ser ignorado.

Em um famoso artigo de 1996 (*https://homl.info/8*),[13] David Wolpert demonstrou que, se você não fizer suposição alguma a respeito dos dados, então não há motivo para preferir um modelo a

13 David Wolpert, "The Lack of A Priori Distinctions Between Learning Algorithms", *Neural Computation* 8, no. 7 (1996): 1341–1390.

outro. Isso se chama *Teorema Não Existe Almoço Grátis* [*No Free Lunch*]. Para alguns conjuntos de dados, o melhor modelo é um modelo linear, ao passo que, para outros conjuntos, será uma rede neural. Não existe um modelo que *a priori* funcione melhor (por isso, o nome do teorema). A única maneira de saber com certeza qual seria o melhor modelo é avaliar todos. Como isso é impossível, na prática você parte de alguns pressupostos sobre os dados e avalia somente alguns modelos razoáveis. Por exemplo, para tarefas simples, você pode avaliar modelos lineares com vários níveis de regularização e, para um problema complexo, pode avaliar diversas redes neurais.

Exercícios

Neste capítulo, abordamos alguns dos conceitos mais importantes do aprendizado de máquina. Nos próximos, investigaremos as coisas mais a fundo e escreveremos mais código. Mas, antes, faça questão de responder às seguintes perguntas:

1. Como você definiria o aprendizado de máquina?
2. Você consegue apontar quatro tipos de problemas que se destacam?
3. O que é um conjunto de treinamento rotulado?
4. Quais são as duas tarefas supervisionadas mais comuns?
5. Você consegue citar quatro tarefas comuns não supervisionadas?
6. Qual tipo de algoritmo de aprendizado de máquina você usaria para possibilitar que um robô andasse por aí em lugares inexplorados?
7. Que tipo de algoritmo você utilizaria a fim de segmentar seus clientes em diversos grupos?
8. Você acha que o problema da detecção de spam tem a ver com o aprendizado supervisionado ou não supervisionado?
9. O que é um sistema de aprendizado online?
10. O que é o aprendizado out-of-core?
11. Qual tipo de algoritmo de aprendizado depende de uma medida de similaridade para efetuar predições?
12. Qual a diferença entre um parâmetro de modelo e o hiperparâmetro do algoritmo de aprendizado?
13. Para que servem os algoritmos de aprendizado baseados em modelos? Qual é a estratégia mais comum que eles utilizam para serem bem-sucedidos? Como eles fazem predições?
14. Você pode mencionar quatro dos principais desafios do aprendizado de máquina?
15. Caso o modelo tenha um bom desempenho nos dados de treinamento, mas a generalização deixa a desejar em instâncias novas, o que está acontecendo? Você pode exemplificar três possíveis soluções?
16. O que é um conjunto de testes e por que você o utilizaria?
17. Qual é o propósito de um conjunto de validação?
18. O que é um train-dev set, quando é necessário e como usá-lo?
19. O que pode sair de errado se você ajustar os hiperparâmetros utilizando o conjunto de teste?

As soluções desses exercícios estão disponíveis no Apêndice A.