

**Mãos à Obra:**

**Aprendizado de Máquina com Scikit-Learn & TensorFlow *Conceitos, Ferramentas e Técnicas para a Construção de Sistemas Inteligentes***

***Aurélien Géron***

Rio de Janeiro, 2019

**Mãos à Obra Aprendizado Máquina com Scikit-Learn e TensorFlow**

Copyright © 2019 da Starlin Alta Editora e Consultoria Eireli. ISBN: 978-85-508-0902-1 (PDF)

*Translated from original Hands-On Machine Learning With Scikit-Learn and TensorFlow© 2017 by Aurélien Géron. All rights re served. ISBN 978-1-491-96229-9. This translation is published and sold by permission of O’Reilly Media, Inc the owner of all rights to publish and sell the same. PORTUGUESE language edition published by Starlin Alta Editora e Consultoria Eireli, Copyright © 2019 by Starlin Alta Editora e Consultoria Eireli.*

Todos os direitos estão reservados e protegidos por Lei. Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser reproduzida ou transmitida. A violação dos Direitos Autorais é crime estabelecido na Lei nº 9.610/98 e com punição de acordo com o artigo 184 do Código Penal.

A editora não se responsabiliza pelo conteúdo da obra, formulada exclusivamente pelo(s) autor(es).

**Marcas Registradas**: Todos os termos mencionados e reconhecidos como Marca Registrada e/ou Comercial são de responsa bilidade de seus proprietários. A editora informa não estar associada a nenhum produto e/ou fornecedor apresentado no livro.

Edição revisada conforme o Acordo Ortográfico da Língua Portuguesa de 2009.

**Publique seu livro com a Alta Books. Para mais informações envie um e-mail para autoria@altabooks.com.br Obra disponível para venda corporativa e/ou personalizada. Para mais informações, fale com projetos@altabooks.com.br**

**Produção Editorial** Editora Alta Books

**Gerência Editorial** Anderson Vieira

**Equipe Editorial**

**Tradução**

Rafael Contatori

**Produtor Editorial** Juliana de Oliveira Tiê Alves

**Assistente Editorial** Adriano Barros

Bianca Teodoro Ian Verçosa

Illysabelle Trajano

**Copidesque**

Amanda Meirinho

**Marketing Editorial** marketing@altabooks.com.br

**Editor de Aquisição** José Rugeri

j.rugeri@altabooks.com.br

Kelry Oliveira

Keyciane Botelho

Maria de Lourdes Borges

**Revisão Gramatical** Vivian Sbravatti

Fernanda Lutfi

**Vendas Atacado e Varejo** Daniele Fonseca

Viviane Paiva

comercial@altabooks.com.br

Paulo Gomes

Thales Silva

Thauan Gomes

**Revisão Técnica**

Gabriel Campos

Engenheiro Eletrônico formado pelo Instituto Militar de Engenharia (IME)

**Ouvidoria**

ouvidoria@altabooks.com.br

**Diagramação**

Daniel Vargas

**Erratas e arquivos de apoio:** No site da editora relatamos, com a devida correção, qualquer erro encontrado em nossos livros, bem como disponibilizamos arquivos de apoio se aplicáveis à obra em questão.

Acesse o site www.altabooks.com.br e procure pelo título do livro desejado para ter acesso às erratas, aos arquivos de apoio e/ou a outros conteúdos aplicáveis à obra.

**Suporte Técnico:** A obra é comercializada na forma em que está, sem direito a suporte técnico ou orientação pessoal/exclusiva ao leitor.

A editora não se responsabiliza pela manutenção, atualização e idioma dos sites referidos pelos autores nesta obra.

**Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD**

| G377m Géron, Aurélien  Mãos à Obra Aprendizado de Máquina com Scikit-Learn &  TensorFlow: Conceitos, Ferramentas e Técnicas Para a Construção de Sistemas Inteligentes / Aurélien Géron ; traduzido por Rafael Contatori. - Rio de Janeiro : Alta Books, 2019.  576 p. : il. ; 42.917 Kb.  Tradução de: Hands-On Machine Learning with Scikit-Learn  &TensorFlow  Inclui índice e anexo.  ISBN: 978-85-508-0902-1 (PDF)  1. Ciência da computação. 2. Aprendizado de Máquina. I. Contatori, Rafael. II. Título.  CDD 004  2019-594  CDU 004 |
| --- |

**Elaborado por Vagner Rodolfo da Silva - CRB-8/9410**

Rua Viúva Cláudio, 291 — Bairro Industrial do Jacaré

CEP: 20.970-031 — Rio de Janeiro (RJ)

Tels.: (21) 3278-8069 / 3278-8419

www.altabooks.com.br — altabooks@altabooks.com.br

www.facebook.com/altabooks — www.instagram.com/altabooks

**Sumário**

**Parte I. Os Fundamentos do Aprendizado de Máquina 1 1. O Cenário do Aprendizado de Máquinas ......................................................3** O que é o Aprendizado de Máquina? 4 Por que Utilizar o Aprendizado de Máquina? 5

Tipos de Sistemas do Aprendizado de Máquina 8 Aprendizado Supervisionado/Não Supervisionado 8 Aprendizado Supervisionado 8 Aprendizado Não Supervisionado 10 Aprendizado Semi-supervisionado 13 Aprendizado por Reforço 14 Aprendizado Online e Em Lote 15 Aprendizado baseado em Instância Versus

Aprendizado baseado no Modelo 18

Principais Desafios do Aprendizado de Máquina 23 Quantidade Insuficiente de Dados de Treinamento 23 Dados de Treinamento Não Representativos 25 Dados de Baixa Qualidade 26 Características Irrelevantes 27 Sobreajustando os Dados de Treinamento 27 Subajustando os Dados de Treinamento 29 Voltando Atrás 30

Testando e Validando 30 Exercícios 32

**iii**

**2. Projeto de Aprendizado de Máquina de Ponta a Ponta...............................35** Trabalhando com Dados Reais 35

Um Olhar no Quadro Geral 37 Enquadre o Problema 37 Selecione uma Medida de Desempenho 39 Verifique as Hipóteses 42

Obtenha os Dados 42 Crie o Espaço de Trabalho 42 Baixe os Dados 45 Uma Rápida Olhada na Estrutura dos Dados 47 Crie um Conjunto de Testes 51

Descubra e Visualize os Dados para Obter Informações 55 Visualizando Dados Geográficos 55 Buscando Correlações 58 Experimentando com Combinações de Atributo 60

Prepare os Dados para Algoritmos do Aprendizado de Máquina 62 Limpeza dos Dados 62 Manipulando Texto e Atributos Categóricos 65 Customize Transformadores 67 Escalonamento das Características 68 Pipelines de Transformação 69

Selecione e Treine um Modelo 71 Treinando e Avaliando no Conjunto de Treinamento 72 Avaliando Melhor com a Utilização da Validação Cruzada 73

Ajuste Seu Modelo 76 Grid Search 76 Randomized Search 78 Métodos de Ensemble 78 Analise os Melhores Modelos e Seus Erros 79 Avalie Seu Sistema no Conjunto de Testes 80

Lance, Monitore e Mantenha seu Sistema 80 Experimente! 81 Exercícios 82

**3. Classificação............................................................................................83** MNIST 83 Treinando um Classificador Binário 86

**iv | Sumário**

Medições de Desempenho 86 Medição da Acurácia com a Utilização da Validação Cruzada 87 Matriz de Confusão 88 Precisão e Revocação 90 Compensação da Precisão/Revocação 91 A Curva ROC 95

Classificação Multiclasse 97 Análise de Erro 100 Classificação Multilabel 104 Classificação Multioutput 105 Exercícios 107

**4. Treinando Modelos ................................................................................ 109**

Regressão Linear 110 Método dos Mínimos Quadrados 112 Complexidade Computacional 114

Gradiente Descendente 115 Gradiente Descendente em Lote 118 Gradiente Descendente Estocástico 121 Gradiente Descendente em Minilotes 124

Regressão Polinomial 125 Curvas de Aprendizado 127

Modelos Lineares Regularizados 132 Regressão de Ridge 132 Regressão Lasso 135 Elastic Net 137 Parada Antecipada 138

Regressão Logística 139 Estimando Probabilidades 140 Treinamento e Função de Custo 141 Limites de Decisão 142 Regressão Softmax 144

Exercícios 147 **5. Máquinas de Vetores de Suporte (SVM) ................................................... 149**

Classificação Linear dos SVM 149 Classificação de Margem Suave 150

**Sumário | v**

Classificação SVM Não Linear 153 Kernel Polinomial 154 Adicionando Características de Similaridade 155 Kernel RBF Gaussiano 156 Complexidade Computacional 158

Regressão SVM 158

Nos Bastidores 160 Função de Decisão e Previsões 161 Objetivo do Treinamento 162 Programação Quadrática 163 O Problema Dual 164 SVM Kernelizado 165 SVM Online 167

Exercícios 169 **6. Árvores de Decisão................................................................................. 171** Treinando e Visualizando uma Árvore de Decisão 171 Fazendo Previsões 173 Estimando as Probabilidades de Classes 175 O Algoritmo de Treinamento CART 175 Complexidade Computacional 176 Coeficiente de Gini ou Entropia? 177 Hiperparâmetros de Regularização 177 Regressão 179 Instabilidade 181 Exercícios 182 **7. Ensemble Learning e Florestas Aleatórias ............................................... 185** Classificadores de Votação 186

Bagging e Pasting 189 Bagging e Pasting no Scikit-Learn 190 Avaliação Out-of-Bag 191

Patches e Subespaços Aleatórios 192

Florestas Aleatórias 193 Árvores-Extras 194 Importância da Característica 194

**vi | Sumário**

Boosting 196 AdaBoost 196 Gradient Boosting 199

Stacking 204 Exercícios 207 **8. Redução da Dimensionalidade................................................................209** A Maldição da Dimensionalidade 210

Principais Abordagens para a Redução da Dimensionalidade 211 Projeção 211 Manifold Learning 213

PCA 215 Preservando a Variância 215 Componentes Principais 216 Projetando para d Dimensões 217 Utilizando o Scikit-Learn 218 Taxa de Variância Explicada 218 Escolhendo o Número Certo de Dimensões 219 PCA para a Compressão 220 PCA Incremental 221 PCA Randomizado 222

Kernel PCA 222 Selecionando um Kernel e Ajustando Hiperparâmetros 223

LLE 225 Outras Técnicas de Redução da Dimensionalidade 227 Exercícios 228

**Parte II. Redes Neurais e Aprendizado Profundo 231 9. Em Pleno Funcionamento com o TensorFlow ........................................... 233** Instalação 236 Criando Seu Primeiro Grafo e o Executando em uma Sessão 237 Gerenciando Grafos 238 Ciclo de Vida de um Valor do Nó 239 Regressão Linear com o TensorFlow 240

**Sumário | vii**

Implementando o Gradiente Descendente 241 Calculando Manualmente os Gradientes 241 Utilizando o autodiff 242

Utilizando um Otimizador 244 Fornecendo Dados ao Algoritmo de Treinamento 244 Salvando e Restaurando Modelos 245

Visualização do Grafo e Curvas de Treinamento

com o TensorBoard 247 Escopos do Nome 250 Modularidade 251 Compartilhando Variáveis 253 Exercícios 256

**10. Introdução às Redes Neurais Artificiais ................................................... 259**

De Neurônios Biológicos a Neurônios Artificiais 260 Neurônios Biológicos 261 Cálculos Lógicos com Neurônios 262 O Perceptron 263 Perceptron Multicamada e Retropropagação 267

Treinando um MLP com a API de Alto Nível do TensorFlow 270

Treinando um DNN Utilizando um TensorFlow Regular 271 Fase de Construção 271 Fase de Execução 275 Utilizando a Rede Neural 276

Ajustando os Hiperparâmetros da Rede Neural 277 Número de Camadas Ocultas 277 Número de Neurônios por Camada Oculta 278 Funções de Ativação 279

Exercícios 280 **11. Treinando Redes Neurais Profundas........................................................ 283**

Problemas dos Gradientes: Vanishing/Exploding 283 Inicialização Xavier e Inicialização He 285 Funções de Ativação Não Saturadas 287 Normalização em Lote 290 Implementando a Normalização em Lote com o TensorFlow 292 Gradient Clipping 294

**viii | Sumário**

Reutilizando Camadas Pré-Treinadas 295 Reutilizando um Modelo do TensorFlow 296 Reutilizando Modelos de Outras Estruturas 298 Congelando as Camadas Inferiores 298

Armazenamento em Cache das Camadas Congeladas 299 Ajustando, Descartando ou Substituindo as

Camadas Superiores 300 Zoológicos de Modelos 301 Pré-treinamento Não Supervisionado 301 Pré-treinamento em uma Tarefa Auxiliar 302

Otimizadores Velozes 303 Otimização Momentum 303 Gradiente Acelerado de Nesterov 305 AdaGrad 306 RMSProp 307 Otimização Adam 308

Evitando o Sobreajuste Por Meio da Regularização 312 Parada Antecipada 313 Regularização ℓ1 e ℓ2 313 Dropout 314 Regularização Max-Norm 317 Data Augmentation 319

Diretrizes Práticas 320 Exercícios 321 **12. Distribuindo o TensorFlow Por Dispositivos e Servidores.......................... 325**

Múltiplos Dispositivos em uma Única Máquina 326 Instalação 326 Gerenciando a RAM do GPU 329 Colocando Operações em Dispositivos 331 Posicionamento Simples 331 Registro dos Posicionamentos 332 Função de Posicionamento Dinâmico 333 Operações e Kernels 333 Posicionamento Suave 334 Execução em Paralelo 334 Dependências de Controle 335

Vários Dispositivos em Vários Servidores 336 Abrindo uma Sessão 338 Os Serviços Master e Worker 338

**Sumário | ix**

Fixando Operações em Tarefas 339 Particionando Variáveis em Múltiplos

Servidores de Parâmetros 340 Compartilhando Estado entre Sessões com a Utilização

de Contêiner de Recursos 341 Comunicação Assíncrona com a Utilização de Filas

do TensorFlow 343 Enfileirando Dados 344 Desenfileirando os Dados 344 Filas de Tuplas 345 Fechando uma Fila 346 RandomShuffleQueue 346 PaddingFifoQueue 347 Carregando Dados Diretamente do Grafo 348 Pré-carregue os Dados em uma Variável 348 Lendo os Dados de Treinamento Diretamente do Grafo 349 Leitores Multithreaded Utilizando as Classes

Coordinator e QueueRunner 352 Outras Funções de Conveniências 354

Paralelizando Redes Neurais em um Cluster do TensorFlow 356 Uma Rede Neural por Dispositivo 356 Replicação em Grafo Versus Replicação Entre Grafos 357 Paralelismo do Modelo 359 Paralelismo de Dados 361

Exercícios 366 **13. Redes Neurais Convolucionais (CNN) ....................................................... 369** A Arquitetura do Córtex Visual 370

Camada Convolucional 371 Filtros 373 Empilhando Múltiplos Mapas de Características 374 Implementação do TensorFlow 376 Requisitos de Memória 379

Camada Pooling 380

Arquiteturas CNN 381 LeNet-5 382 AlexNet 384 GoogLeNet 385 ResNet 389

Exercícios 393

**x | Sumário**

**14. Redes Neurais Recorrentes (RNN)............................................................ 397**

Neurônios Recorrentes 398 Células de Memória 400 Sequências de Entrada e Saída 401

RNNs Básicas no TensorFlow 402 Desenrolamento Estático Através do Tempo 403 Desenrolamento Dinâmico Através do Tempo 405 Manipulando Sequências de Entrada de Comprimento Variável 406 Manipulando Sequências de Saída de Comprimento Variável 407

Treinando RNNs 407 Treinando um Classificador de Sequência 408 Treinando para Prever Séries Temporais 410 RNN Criativa 415

RNNs Profundas 415 Distribuindo uma RNN Profunda Através de Múltiplas GPUs 416 Aplicando o Dropout 418 A Dificuldade de Treinar sobre Muitos Intervalos de Tempo 419

Célula LSTM 420 Conexões Peephole 422

Célula GRU 423

Processamento de Linguagem Natural 424 Word Embeddings 424 Uma Rede Codificador-Decodificador para

Tradução de Máquina 426 Exercícios 429 **15. Autoencoders........................................................................................ 431** Representações Eficientes de Dados 432 Executando o PCA com um Autoencoder Linear Incompleto 433

Autoencoders Empilhados 435 Implementação do TensorFlow 436 Amarrando Pesos 437 Treinando um Autoencoder Por Vez 438 Visualizando as Reconstruções 441 Visualizando as Características 441

Pré-treinamento Não Supervisionado Utilizando

Autoencoders Empilhados 443

**Sumário | xi**

Autoencoders de Remoção de Ruídos 444 Implementando o TensorFlow 445

Autoencoders Esparsos 446 Implementando o TensorFlow 448

Autoencoders Variacionais 449 Gerando Dígitos 452

Outros Autoencoders 453 Exercícios 454 **16. Aprendizado por Reforço........................................................................ 457** Aprendendo a Otimizar Recompensas 458 Pesquisa de Políticas 459 Introdução ao OpenAI Gym 461 Políticas de Rede Neural 464 Avaliação das Ações: O Problema de Atribuição de Crédito 467 Gradientes de Política 468 Processos de Decisão de Markov 473

Aprendizado de Diferenças Temporais e Q-Learning 477 Políticas de Exploração 479 Q-Learning Aproximado e Deep Q-Learning 480

Aprendendo a Jogar Ms. Pac-Man com a

Utilização do Algoritmo DQN 482 Exercícios 489 Obrigado! 491

**A. Soluções dos Exercícios........................................................................... 493 B. Lista de Verificação do Projeto de Aprendizado de Máquina..................... 521 C. Problema SVM Dual ............................................................................... 527 D. Autodiff ................................................................................................ 531 E. Outras Arquiteturas Populares RNA......................................................... 539**

**Índice......................................................................................................... 549**

**Prefácio**

**O Tsunami do Aprendizado de Máquina** Em 2006, Geoffrey Hinton *et al.* publicou um artigo*1* mostrando como treinar uma rede neural profunda capaz de reconhecer dígitos manuscritos com uma precisão de última geração (> 98%). A técnica foi rotulada como “Aprendizado Profundo” [*Deep Learning*]. Treinar uma rede neural profunda era considera do uma tarefa impossível na época*2* e, desde os anos 1990, que a maioria dos pesquisadores havia abandonado essa ideia. Este artigo reavivou o interesse da comunidade científica e, em pouco tempo, muitos novos trabalhos demonstraram que o Aprendizado Profundo não só era possível, mas capaz de feitos alucinantes que nenhuma outra técnica do Aprendizado de Máquina (AM) poderia esperar alcançar (com a ajuda do tremendo poder de computação e grandes quanti dades de dados). Esse entusiasmo logo se estendeu a muitas outras áreas do Aprendizado de Máquina.

Avançaremos dez anos para ver que o Aprendizado de Máquina conquistou a indús tria e, atualmente, está nos holofotes dos produtos de alta tecnologia, classificando os resultados de pesquisas na web, potencializando o reconhecimento de voz do smartphone, recomendando vídeos e superando o campeão mundial no jogo Go. Antes que você perceba, estará dirigindo seu carro.

**O Aprendizado de Máquina em Seus Projetos** Naturalmente você está eufórico com o Aprendizado de Máquina e adoraria participar da festa!

1 Disponível em *http://www.cs.toronto.edu/~hinton/* (conteúdo em inglês).

2 As redes neurais convolucionais de aprendizado profundo de Yann Lecun funcionavam bem para o reconhecimento de imagens desde a década de 1990, embora não fossem de propósito geral.

**xiii**

Talvez seu robô caseiro pudesse ganhar um cérebro próprio? Talvez ele pudesse reco nhecer rostos ou aprender a andar?

Ou talvez sua companhia tenha toneladas de dados (financeiros, de produção, de sensores de máquinas, logs de usuários, estatísticas de hotline, relatórios de RH, etc.) e, muito provavelmente, você descobriria algumas pepitas de ouro escondidas se soubesse onde procurar:

• Segmentar clientes e encontrar a melhor estratégia de marketing para cada grupo;

• Recomendar produtos para cada cliente com base no que clientes similares compraram;

• Detectar quais transações são susceptíveis de serem fraudulentas; • Prever a receita do próximo ano;

e mais (*https://www.kaggle.com/wiki/DataScienceUseCases*).*3*

Seja qual for a razão, você decidiu se familiarizar com o Aprendizado de Máquina e implementá-lo em seus projetos. O que é uma grande ideia!

**Objetivo e Abordagem**

Este livro pressupõe que você não saiba quase nada sobre Aprendizado de Máquinas. Seu objetivo é fornecer os conceitos, as intuições e as ferramentas necessárias para im plementar programas capazes de *aprender com os dados*.

Abordaremos um grande número de técnicas, desde as mais simples às mais comu mente utilizadas (como a Regressão Linear) e até algumas das técnicas do Aprendizado Profundo que ganham competições com frequência.

Em vez de implementar nossas próprias versões de cada algoritmo, utilizaremos estru turas Python prontas para produção:

• O Scikit-Learn (*http://scikit-learn.org/*) é muito fácil de usar e implementa mui tos algoritmos do AM de maneira eficiente, por isso é uma excelente porta de entrada para o Aprendizado de Máquinas.

• O TensorFlow (*http://tensorflow.org/*) é uma biblioteca complexa que utiliza grafos de fluxo de dados para o cálculo numérico distribuído. Distribuindo os

3 Todo o conteúdo dos websites citados está em inglês. A editora Alta Books não se responsabiliza pela manu tenção desse conteúdo.

**xiv | Prefácio**

cálculos entre potencialmente milhares de servidores multi GPU, essa biblioteca torna possível treinar e executar de forma eficiente enormes redes neurais. O TensorFlow foi criado no Google e suporta muitas das aplicações em larga escala do Apren dizado de Máquina. Em novembro de 2015 ele se tornou de código aberto.

O livro favorece uma abordagem prática, desenvolvendo uma compreensão intuitiva de Aprendizado de Máquina por meio de exemplos de trabalho concretos e apenas um pouco de teoria. Embora você possa ler este livro sem pegar seu notebook, é altamente recomendável que você treine com os notebooks Jupyter os exemplos de código dispo

níveis online em *https://github.com/ageron/handson-ml*.

**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, princi palmente NumPy (*http://numpy.org/*), Pandas (*http://pandas.pydata.org/*) e Matplotlib (*http://matplotlib.org/*).

Além disso, você deve ter uma compreensão razoável da matemática em nível superior, (cálculo, álgebra linear, probabilidade e estatística) caso se importe com os bastidores.

Se você ainda não conhece o Python, o *http://learnpython.org/* é um ótimo lugar para começar. O tutorial oficial em *python.org* (*https://docs.python.org/3/tutorial/*) também é muito bom.

Se você nunca usou o Jupyter, o Capítulo 2 o guiará na instalação e no básico: uma ótima ferramenta para ter sempre à mão.

Caso você não esteja familiarizado com as bibliotecas científicas do Python, os notebooks Jupyter incluem alguns tutoriais. Há também um tutorial rápido de mate mática para álgebra linear.

**Roteiro**

Este livro foi organizado em duas partes.

Parte I, *Os Fundamentos do Aprendizado de Máquina*, cobre os seguintes tópicos:

• O que é Aprendizado de Máquina? Que problemas ele tenta resolver? Quais são as principais categorias e conceitos fundamentais dos sistemas do Aprendizado de Máquina?

• Os principais passos de um típico projeto de Aprendizado de Máquina;

**Prefácio | xv**

• Aprender ajustando um modelo aos dados;

• Otimizar a função de custo;

• Manipular, limpar e preparar os dados;

• Selecionar e desenvolver recursos;

• Selecionar um modelo com a utilização da validação cruzada e ajustar hiperparâmetros;

• Os principais desafios do Aprendizado de Máquina, especialmente o subajuste e o sobreajuste (a compensação do viés/variância);

• Reduzir a dimensionalidade dos dados de treinamento para combater a maldi ção da dimensionalidade;

• Os algoritmos de aprendizado mais comuns: Regressão Linear e Polinomial, Re gressão Logística, *k-Nearest Neighbors*, Máquinas de Vetores de Suporte [SVM], Árvores de Decisão, Florestas Aleatórias e Métodos de *Ensemble*.

Parte II, *Redes Neurais e Aprendizado Profundo*, cobre os seguintes tópicos: • O que são redes neurais? Servem para quê?

• Construir e treinar redes neurais com a utilização do TensorFlow;

• As mais importantes arquiteturas de redes neurais: *feedforward*, convolucionais, recorrentes, LSTM (*long short-term memory*) e *autoencoders*;

• Técnicas para treinamento de redes neurais profundas;

• Escalonar redes neurais para grandes conjuntos de dados;

• Aprender por reforço.

A primeira parte é baseada principalmente no Scikit-Learn, enquanto a segunda utiliza o TensorFlow.

Não mergulhe de cabeça logo de primeira: embora o Aprendizado Pro

fundo seja, sem dúvida, uma das áreas mais interessantes do Aprendizado

de Máquina, primeiro você deve dominar os fundamentos. Além disso,

a maioria dos problemas pode ser resolvida com técnicas mais simples,

como os métodos Florestas Aleatórias e *Ensemble* (discutidos na Parte

I). O Aprendizado Profundo é mais adequado para problemas comple

xos, como reconhecimento de imagem e de voz, ou processamento de

linguagem natural, desde que você tenha dados, poder de computação e

paciência sufcientes.

**xvi | Prefácio**

**Outros Recursos**

Para saber mais sobre o Aprendizado de Máquina existem muitos recursos disponíveis. O curso de AM de Andrew Ng no Coursera (*https://www.coursera.org/learn/machine- -learning/*) e o curso de Geoffrey Hinton sobre redes neurais e Aprendizado Profundo (*https://www.coursera.org/course/neuralnets*) são incríveis, embora ambos demandem um investimento significativo de tempo (estamos falando de meses).

Há também muitos sites interessantes sobre o Aprendizado de Máquina, incluindo, claro, o excepcional Guia do Usuário do Scikit-Learn (*http://scikit-learn.org/stable/user\_guide. html*). Você também pode desfrutar do Dataquest (*https://www.dataquest.io/*), que oferece tutoriais interativos muito interessantes e blogs de AM, como aqueles listados no Quora (*http://goo.gl/GwtU3A*). Finalmente, o site do Aprendizado Profundo (*http://deeplearning. net/*) tem uma boa lista de recursos para seguirmos aprendendo mais.

Há também muitos outros livros introdutórios sobre o Aprendizado de Máquina:

• *Data Science do Zero,* de Joel Grus (Alta Books). Apresenta os fundamentos do Aprendizado de Máquina e implementa alguns dos principais algoritmos em Python puro (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, cobrindo 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. Aproveita as bibliotecas de código aberto do Python (Pylearn 2 e Theano);

• *Learning from Data,* de Yaser S. Abu-Mostafa, Malik Magdon-Ismail e Hsuan-Tien Lin (AMLBook). Uma abordagem bastante teórica para o AM, este livro fornece insights profundos, principalmente sobre a compensação do viés/ variância (ver Capítulo 4);

• *Inteligência Artificial, 3ª Edição,* de Stuart Russell e Peter Norvig (Campus). É um ótimo (e enorme) livro que aborda uma quantidade incrível de tópicos, in cluindo o Aprendizado de Máquina. Ajudando a colocar o AM em perspectiva.

Finalmente, uma ótima maneira de aprender é entrar em sites de competição como o Kaggle.com, que lhe permite praticar suas habilidades com problemas do mundo real com a ajuda e insights de alguns dos melhores profissionais de AM que existem por aí.

**Prefácio | xvii**

**Convenções Utilizadas neste Livro**

As seguintes convenções tipográficas são usadas neste livro:

*Itálicos*

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

Fonte monoespaçada

Usada para listagens de programas, bem como em parágrafos sobre elementos de programa, como 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 literalmente 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.

**xviii | Prefácio**

**Utilizando Exemplos de Código**

O material suplementar (exemplos de códigos, exercícios, etc.) está disponível para download em *https://github.com/ageron/handson-ml* e também no site da editora Alta Books. Entre em *www.altabooks.com.br* e procure pelo nome do livro.

Este livro está aqui para ajudá-lo a fazer o que precisa ser feito. Em geral, se um código de exemplo for oferecido, você poderá utilizá-lo em seus 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 pro

grama que utiliza vários trechos de código deste livro não requer permissão. Vender ou distribuir um CD-ROM com exemplos dos livros O’Reilly exigirá permissão. Responder uma pergunta citando este livro e citando um exemplo de código não requer permissão. Mas, a incorporação de uma quantidade significativa de exemplos de código deste livro na documentação do seu produto requer sim permissão.

Agradecemos, mas não exigimos atribuições. Uma atribuição geralmente inclui o título, o autor, o editor e o ISBN. Por exemplo: “*Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow* por Aurélien Géron (AltaBooks). Copyright 2019 de Aurélien Géron, 978-85-508-0381-4.”

**Agradecimentos**

Gostaria de agradecer aos meus colegas do Google, em especial à equipe de classificação de vídeos do YouTube, por me ensinarem muito sobre o Aprendizado de Máquina. Nunca teria iniciado este projeto sem eles. Agradecimentos especiais aos meus gurus pessoais do AM: Clément Courbet, Julien Dubois, Mathias Kende, Daniel Kitachewsky, James Pack, Alexander Pak, Anosh Raj, Vitor Sessak, Wiktor Tomczak, Ingrid von Glehn, Rich Washington e todos do YouTube de Paris.

Sou incrivelmente grato a todas as pessoas maravilhosas que tiraram um tempo de suas vidas ocupadas para revisar meu livro com tantos detalhes. Agradeço a Pete Warden por responder todas as minhas perguntas sobre o TensorFlow, revisar a Parte II, fornecer muitos insights interessantes e, claro, por fazer parte da equipe principal do TensorFlow. Aconselho você a definitivamente acompanhar o blog dele (*https://petewarden.com/*)! Muito obrigado a Lukas Biewald por sua revisão completa da Parte II: ele não deixou pedra sobre pedra, testou todo o código (e pegou alguns erros), fez ótimas sugestões, e seu entusiasmo foi contagiante. Acompanhe o blog dele (*https://lukasbiewald.com/*) e seus incríveis robôs (*https://goo.gl/Eu5u28*)! Agradecimentos a Justin Francis, que também analisou muito detalhadamente a Parte II, pegando erros e fornecendo importantes insights, principal mente no Capítulo 16. Confira suas postagens (*https://goo.gl/28ve8z*) sobre o TensorFlow!

**Prefácio | xix**

Muito obrigado também a David Andrzejewski, que revisou a Parte I identificando seções pouco claras e sugerindo como melhorá-las e forneceu um feedback incrivelmente útil. Confira o site dele (*http://www.david-andrzejewski.com/*)! Agradecimentos a Grégoire Mesnil, que revisou a Parte II e contribuiu com conselhos práticos muito interessantes sobre o treinamento de redes neurais. Agradeço também a Eddy Hung, Salim Sémaoune, Karim Matrah, Ingrid von Glehn, Iain Smears e Vincent Guilbeau por revisarem a Parte I e por fazerem muitas sugestões úteis. Também gostaria de agradecer ao meu sogro, Michel Tessier, ex-professor de matemática e agora um grande tradutor de Anton Tchekhov, por me ajudar a resolver algumas das matemáticas e anotações deste livro e rever o notebook do Jupyter de álgebra linear.

E, claro, um “obrigado” gigantesco ao meu querido irmão Sylvain, que revisou cada capítulo, testou todas as linhas de código, forneceu feedback sobre praticamente todas as seções e me encorajou desde a primeira até a última linha. Amo você, irmãozinho!

Muito obrigado também à fantástica equipe da O’Reilly, principalmente Nicole Tache, que me deu um feedback perspicaz, e foi sempre alegre, encorajadora e prestativa. Agradeço também a Marie Beaugureau, Ben Lorica, Mike Loukides e Laurel Ruma por acredita rem neste projeto e me ajudarem a definir seu escopo. Obrigado a Matt Hacker e a toda a equipe da Atlas por responder todas as minhas questões técnicas sobre formatação, *asciidoc* e LaTeX, e a Rachel Monaghan, Nick Adams e toda a equipe de produção pela revisão final e centenas de correções.

Por último, mas não menos importante, sou infinitamente grato a minha amada esposa, Emmanuelle, e aos nossos três maravilhosos filhos, Alexandre, Rémi e Gabrielle, por me encorajarem a trabalhar muito neste livro, fazerem muitas perguntas (quem disse que não é possível ensinar redes neurais para uma criança de sete anos?) e até trazerem biscoitos e café. O que mais se pode sonhar?

**Aviso**

Para melhor entendimento as figuras coloridas estão disponíveis no site da editora Alta Books. Acesse: www.altabooks.com.br e procure pelo nome do livro ou ISBN.

**xx | Prefácio**

**Parte I**

**Os Fundamentos**

**do Aprendizado de Máquina**

**Capítulo 1**

**O Cenário do Aprendizado de Máquina**

A maioria das pessoas pensa em um robô quando ouve “Aprendizado de Máquina”: um mordomo confiável ou um Exterminador do Futuro, dependendo para quem você perguntar. Mas o Aprendizado de Máquina não é apenas uma fantasia futurista; ele já está entre nós. Na verdade, há algumas décadas o AM foi introduzido como *Reco*

*nhecimento Ótico de Caracteres* (OCR) em algumas publicações especializadas, mas o primeiro aplicativo que realmente ficou famoso 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, mas pode ser tecnicamente qualificado 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, fornecem centenas de produtos e recursos, de recomendações a buscas por voz, que você utiliza regularmente.

Onde começa e onde termina o Aprendizado de Máquina? O que significa exatamente que uma máquina *aprende* alguma coisa? Se eu baixar uma cópia da Wikipédia, meu computador realmente “aprenderá” algo? Será que de repente ele fica mais esperto? Neste capítulo, começaremos esclarecendo o que é o Aprendizado de Máquina e por que você vai querer utilizá-lo.

Então, antes de iniciarmos a exploração do mundo do Aprendizado de Máquina, anali saremos seu mapa e conheceremos suas principais regiões e os cenários mais notáveis: aprendizado supervisionado versus não supervisionado, aprendizado online versus aprendizado em lote, aprendizado baseado em instâncias versus aprendizado baseado em modelo. Em seguida, analisaremos o fluxo de trabalho de um típico projeto de AM, discutiremos os principais desafios que você poderá enfrentar e mostraremos como avaliar e ajustar um sistema de Aprendizado de Máquina.

Este capítulo apresenta muitos conceitos fundamentais (e jargões) que todo cientista de dados deve saber de cor. Será uma visão geral de alto nível (o único capítulo sem muito

**3**

código), tudo bastante simples, mas você deve se certificar de que entendeu tudo 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 fnal do capítulo

antes de seguir em frente.

**O que É o Aprendizado de Máquina?**

Aprendizado de Máquina é a ciência (e a arte) da programação de computadores para que eles possam *aprender com os dados*.

Veja uma definição um pouco mais abrangente:

[Aprendizado de Máquina é o] campo de estudo que dá aos computadores a habilidade de aprender sem ser explicitamente programado.

—Arthur Samuel, *1959*

E uma mais direcionada aos engenheiros:

Diz-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 assinalar e-mails como spam (por exemplo, os marcados pelos usuários) e como regulares (não spam, também chamados de “ham”). Os exemplos utilizados pelo sistema para o aprendizado são chamados de *conjuntos de treinamentos*. Cada exemplo de treinamento é chamado de *instância de treinamento* (ou *amostra*). Nesse caso, assi

nalar novos e-mails como spam é a tarefa T, a experiência E é o *dado de treinamento* e a medida de desempenho P precisa ser definida; por exemplo, você pode utilizar a média de e-mails classificados corretamente. Esta medida de desempenho particular é chamada de *acurácia* e é utilizada frequentemente em tarefas de classificação.

Se você baixou uma cópia da Wikipédia, seu computador terá muito mais dados, mas não terá repentinamente um melhor desempenho em nenhuma tarefa. Portanto, não é Aprendizado de Máquina.

**4 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Por que Utilizar o Aprendizado de Máquina?**

Considere como você escreveria um filtro de spam utilizando técnicas de programação tradicionais (Figura 1-1):

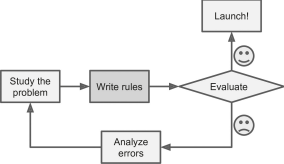
1. Primeiro, identificaria as características do spam. Você nota que algumas palavras ou frases (termos em inglês como “4U”, “credit card”, “free” e “amazing”) tendem a aparecer muito no campo do assunto. Talvez você note outros padrões no nome do remetente, no corpo do e-mail, e assim por diante.

2. Segundo, escreveria um algoritmo de detecção para cada um dos padrões observa dos, e, se fossem detectados, seu programa marcaria esses e-mails como spam.

3. Por último, você testaria seu programa, e repetiria os passos 1 e 2 até que esteja satisfatório.

Lance!

Estude o problema

Avalie Escreva regras 

Analise os

erros

*Figura 1-1. A abordagem tradicional*

Como o problema não é trivial, seu programa provavelmente se tornará uma longa lista de regras complexas — com uma manutenção muito difícil.

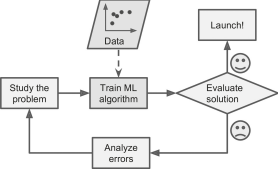
Em contrapartida, um filtro de spam baseado em técnicas de Aprendizado de Máquina aprende automaticamente quais palavras e frases são bons indicadores de spam, detectan do padrões de palavras estranhamente frequentes em exemplos de spam se comparados aos exemplos dos e-mails “não spam” (Figura 1-2). O programa é muito menor, de mais fácil manutenção e, provavelmente, mais preciso.

**Por que Utilizar o Aprendizado de Máquina? | 5**

Lance!

Dados

Estude o problema

Treine o 

algoritmo AM

Analise os erros

Avalie a solução

*Figura 1-2. Abordagem do Aprendizado de Máquina*

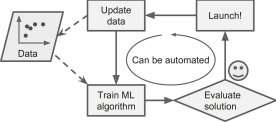
Além disso, se os *spammers* perceberem que todos os seus e-mails contendo “4U” são bloqueados, poderão começar a escrever “For U”. Um filtro de *spam* que utiliza técnicas de programação tradicionais precisaria ser atualizado para marcar os e-mails “For U”. Se os *spammers* continuam contornando seu filtro de *spam*, será preciso escrever novas regras para sempre.

Em contrapartida, um filtro de spam baseado em técnicas de Aprendizado de Máquina percebe automaticamente que “For U” tornou-se frequente no spam marcado pelos usuários e começa a marcá-los sem a sua intervenção (Figura 1-3).

Atualize os

dadosLance!

Dados

Pode ser 

automatizado

Treine o

algoritmo AM

Avalie a soluçao

*Figura 1-3. Adaptando-se automaticamente à mudança*

Outra área na qual o Aprendizado de Máquina se destaca é nos problemas muito com plexos para abordagens tradicionais ou que não possuem um algoritmo conhecido. Por exemplo, considere o reconhecimento da fala: digamos que você deseja começar com o básico e escreve um programa capaz de distinguir as palavras “one” e “two”. Você deve

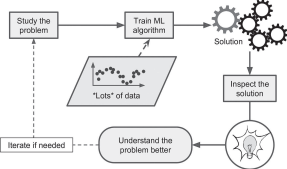
**6 | Capítulo 1: O Cenário do Aprendizado de Máquina**

perceber que a palavra “Two” começa com um som agudo (“T”), então você codifica um algoritmo que mede a intensidade do som agudo e o utiliza para distinguir “One” e “Two”. Obviamente, esta técnica não se estenderá a milhares de palavras faladas por milhões de pessoas muito diferentes em ambientes confusos e em dezenas de idiomas. A melhor solução (pelo menos hoje) seria escrever um algoritmo que aprenda por si só por meio de muitas gravações de exemplos para cada palavra.

Finalmente, 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 (embora para alguns algoritmos isso possa ser complicado). Por exemplo, uma vez que o filtro foi treinado para o spam, ele pode facilmente ser inspecionado e revelar uma lista de palavras e combinações previstas que ele acredita serem as mais prováveis. Às vezes, isso revelará correlações não esperadas ou novas tendências, levando a 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 aparentes. Isto é chamado de mineração de dados.

Estude o problema

Treine o 

algoritmo AM

Solução

Inspecione

Itere se necessário

\*Muitos\* dados

Entenda melhor

o problema

a solução

*Figura 1-4. O Aprendizado de Máquina pode ajudar no ensino dos humanos* Resumindo, o Aprendizado de Máquina é ótimo para:

• Problemas para os quais as soluções existentes exigem muita configuração manual ou longas listas de regras: um algoritmo de Aprendizado de Máquina geralmente simplifica e melhora o código;

• Problemas complexos para os quais não existe uma boa solução quando utilizamos uma abordagem tradicional: as melhores técnicas de Aprendizado de Máquina podem encontrar uma solução;

**Por que Utilizar o Aprendizado de Máquina? | 7**

• Ambientes flutuantes: um sistema de Aprendizado de Máquina pode se adaptar a novos dados;

• Compreensão de problemas complexos e grandes quantidades de dados.

**Tipos de Sistemas do Aprendizado de Máquina**

Existem tantos tipos diferentes de sistemas de Aprendizado de Máquina que será útil classificá-los em extensas categorias com base em:

• Serem ou não treinados com supervisão humana (supervisionado, não supervi sionado, semissupervisionado e aprendizado por reforço);

• Se podem ou não aprender rapidamente, de forma incremental (aprendizado online versus aprendizado por lotes);

• 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 versus aprendizado baseado em modelo).

Esses critérios não são exclusivos; você pode combiná-los da maneira que quiser. Por exemplo, um filtro de spam de última geração pode aprender rapidamente com a utiliza ção de um modelo de rede neural profundo treinado com exemplos de spam e não spam, fazendo deste um sistema de aprendizado supervisionado online, baseado em modelos.

Vejamos cada um desses critérios um pouco mais de perto.

**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 por reforço.

**Aprendizado Supervisionado**

No *aprendizado supervisionado*, os dados de treinamento que você fornece ao algoritmo incluem as soluções desejadas, chamadas de *rótulos* (Figura 1-5).

**8 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Conjunto de treinamento** 

Rótulo

Instância

Nova instância

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

A *classificação* é uma tarefa típica do aprendizado supervisionado. O filtro de spam é um bom exemplo disso: ele é treinado com muitos exemplos de e-mails junto às *classes* (spam ou não spam) e deve aprender a classificar novos e-mails.

Prever um *alvo* de valor numérico é outra tarefa típica, como o preço de um carro a partir de um conjunto de *características* (quilometragem, idade, marca, etc.) denomi nadas *previsores*. Esse tipo de tarefa é chamada de *regressão* (Figura 1-6)*1*. Para treinar o sistema, você precisa fornecer muitos exemplos de carros incluindo seus previsores e seus *labels* (ou seja, seus preços).

No Aprendizado de Máquina, um *atributo* é um tipo de dado (por

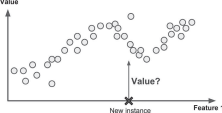
exemplo, “Quilometragem”), enquanto uma *característica* possui

vários signifcados, dependendo do contexto, geralmente signif

cando um atributo mais o seu valor (por exemplo, “Quilometragem

= 15000”). Embora muitas pessoas utilizem as palavras *atributo* e

*característica* intercambiavelmente.

**Valor** 

**Valor?**

**Característica1**

*Figura 1-6. Regressão*

Nova instância

1 Curiosidade: este nome estranho é um termo de estatística introduzido por Francis Galton, enquanto estudava o fato de que os filhos de pessoas altas tendem a ser mais baixos do que os pais. Como as crianças eram mais baixas, ele chamou essa alteração de *regressão à média*. Este nome foi aplicado aos métodos utilizados por ele para analisar as correlações entre variáveis.

**Tipos de Sistemas do Aprendizado de Máquina | 9**

Observe que alguns algoritmos de regressão também podem ser utilizados para clas sificação, e vice-versa. Por exemplo, a *Regressão Logística* é comumente utilizada para classificação pois pode produzir um valor que corresponde à probabilidade de pertencer a uma determinada classe (por exemplo, 20% de chances de ser spam).

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

• k-Nearest Neighbours

• Regressão Linear

• Regressão Logística

• Máquinas de Vetores de Suporte (SVM)

• Árvores de Decisão e Florestas Aleatórias

• Redes Neurais*2*

**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.

**Conjunto de treinamento**

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

Eis alguns dos mais importantes algoritmos de aprendizado não supervisionado (fala remos sobre a redução da dimensionalidade no Capítulo 8):

2 Algumas arquiteturas de redes neurais podem ser não supervisionadas, como *autoencoders* e máquinas de Boltzmann restritas. Elas também podem ser semi-supervisionadas, como redes de crenças profundas e pré-treino sem supervisão.

**10 | Capítulo 1: O Cenário do Aprendizado de Máquina**

• Clustering

— k-Means

— Clustering Hierárquico [HCA, do inglês]

— Maximização da Expectativa

• Visualização e redução da dimensionalidade

— Análise de Componentes Principais [PCA, do inglês]

— Kernel PCA

— Locally-Linear Embedding (LLE)

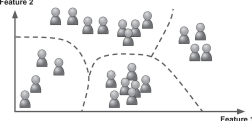
— t-distributed Stochastic Neighbor Embedding (t-SNE)

• Aprendizado da regra da associação

— Apriori

— Eclat

Por exemplo, digamos que você tenha muitos dados sobre os visitantes do seu blog. Você quer executar um algoritmo de *clustering* para tentar detectar grupos de visitantes semelhantes (Figura 1-8). Em nenhum momento você diz ao algoritmo a qual grupo o visitante pertence: ele encontrará essas conexões sem sua ajuda. Por exemplo, ele pode notar que 40% dos seus visitantes são homens que adoram histórias em quadrinhos e geralmente leem seu blog à noite, enquanto 20% são jovens amantes de ficção científica e o visitam durante os finais de semana, e assim por diante. Se você utilizar um algoritmo de *clustering hierárquico,* ele também poderá subdividir cada grupo em grupos menores. O que pode ajudá-lo a segmentar suas postagens para cada um deles.

**Característica 2** 

**Característica1**

*Figura 1-8. Clustering*

Os algoritmos de *visualização* também são bons exemplos de algoritmos de aprendizado não supervisionado: você os alimenta com muitos dados complexos e não rotulados e eles

**Tipos de Sistemas do Aprendizado de Máquina | 11**

exibem 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 fazer com que os clusters no espaço de entrada que estão separados não se sobre ponham na visualização) para que você possa entender como os dados estão organizados e talvez identificar padrões ignorados.

sapo

gato

pássaro

gato

automóvel caminhão sapo

navio

avião

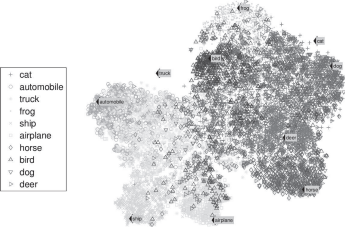
cavalo

pássaro

cachorro cervo

automóvel

avião

cachorro 

cervo

cavalo

navio

caminhão

*Figura 1-9. Exemplo de uma visualização t-SNE destacando grupos semânticos3*

A *redução da dimensionalidade* é uma tarefa relacionada na qual o objetivo é simplificar os dados sem perder muita informação. Uma maneira de fazer isso é mesclar várias ca racterísticas correlacionadas em uma. Por exemplo, a quilometragem de um carro pode estar muito correlacionada com seu tempo de uso, de modo que o algoritmo da redução de dimensionalidade irá mesclá-los em uma característica que representa o desgaste do carro. Isso é chamado de *extração de características*.

3 Observe como os animais estão muito bem separados dos veículos, como os cavalos estão próximos aos cervos, mas longe dos pássaros, e assim por diante. Imagem reproduzida com permissão de Socher, Ganjoo, Manning e Ng (2013), “Visualização T-SNE do espaço da palavra semântica”.

**12 | Capítulo 1: O Cenário do Aprendizado de Máquina**

É uma boa ideia tentar reduzir a dimensão dos dados de treinamento

com a utilização de um algoritmo de redução de dimensionalidade antes

de fornecê-lo a outro algoritmo do Aprendizado de Máquina (como um

algoritmo de aprendizado supervisionado). Este algoritmo será executa

do muito mais rapidamente, os dados ocuparão menos espaço em disco

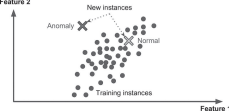
e na memória e, em alguns casos, podem rodar melhor também.

Outra importante tarefa não supervisionada é a *detecção de anomalias* — por exemplo, a detecção de transações incomuns em cartões de crédito para evitar fraudes, detectar defeitos de fabricação ou remover automaticamente outliers de um conjunto de dados antes de fornecê-lo a outro algoritmo de aprendizado. O sistema é treinado com instâncias normais e, quando vê uma nova instância, pode dizer se ela parece normal ou se é uma provável anomalia (veja a Figura 1-10).

**Característica 2** Novas instâncias

Anomalia

*Figura 1-10. Detecção da anomalia*

Normal 

Treinando instâncias

**Característica 1**

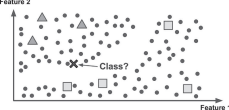
Finalmente, outra tarefa comum não supervisionada é o *aprendizado de regras de associa ção*, cujo objetivo é se aprofundar em grandes quantidades de dados e descobrir relações interessantes entre atributos. Por exemplo, suponha que você possua um supermercado. Executar uma regra de associação em seus registros de vendas pode revelar que as pessoas que compram molho de churrasco e batatas fritas também tendem a comprar carnes. Desta forma, você vai querer colocar esses itens próximos uns dos outros.

**Aprendizado Semi-supervisionado**

Alguns algoritmos podem lidar com dados de treinamento parcialmente rotulados, uma grande quantidade de dados não rotulados e um pouco de dados rotulados. Isso é chamado de *aprendizado semi-supervisionado* (Figura 1-11).

**Tipos de Sistemas do Aprendizado de Máquina | 13**

Alguns serviços de hospedagem de fotos, como o Google Fotos, são bons exemplos disso. Ao carregar todas as suas 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. Esta é a parte não supervisionada do algoritmo (agrupamento). Agora, o sistema apenas precisa que você diga quem são essas pessoas. Com apenas um *rótulo*

por pessoa*4* ele será capaz de nomear todos, o que é útil para pesquisar fotos. **Característica 2** 

**Classe?**

**Característica 1**

*Figura 1-11. Aprendizado Semissupervisionado*

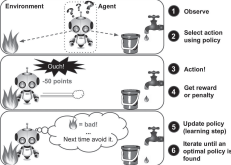
A maior parte dos algoritmos de aprendizado semissupervisionado é de combinações de algoritmos supervisionados e não supervisionados. Por exemplo, as *redes neurais de crenças profundas* [DBNs, do inglês] são baseadas em componentes não supervisionados, chamados *máquinas restritas de Boltzmann* [RBMs, do inglês], empilhados uns em cima dos outros. As RBMs são treinadas sequencialmente de forma não supervisionada, e então todo o sistema é ajustado utilizando-se técnicas de aprendizado supervisionado.

**Aprendizado por Reforço**

O *aprendizado por eforço* é um bicho muito diferente. O sistema de aprendizado, chamado de *agente* nesse contexto, pode observar o ambiente, selecionar e executar ações e obter *recompensas* em troca — ou *penalidades* na forma de recompensas negativas (Figura 1-12). Ele deve aprender por si só 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.

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

**14 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Ambiente** 

Opa!

-50 pontos

**Agente**

**Ruim!**

**Observe**

**Selecione a ação**

**utilizando a política**

**Ação!**

**Ganhe a recompensa ou a penalidade**

**Atualize a política (etapa de aprendizado)**

Da próxima vez, evite.

*Figura 1-12. Aprendizado por reforço*

**Itere até encontrar uma boa política**

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 desenvolveu sua política vencedora analisando milhões de jogos e depois praticando muito contra si mesmo. Note que o aprendizado foi desativado durante os jogos contra o campeão; o AlphaGo apenas aplicou a política que aprendeu.

**Aprendizado Online e em Lote**

Outro critério utilizado para classificar os sistemas de Aprendizado de Máquina é se o sistema pode ou não aprender de forma incremental a partir de um fluxo de dados recebido.

**Aprendizado em lote**

No *aprendizado em lote,* o sistema é incapaz de aprender de forma incremental: ele deve ser treinado com a utilização de todos os dados disponíveis. Isso geralmente demandará muito tempo e recursos de computação, portanto, normalmente é feito offline. Primeiro, o sistema é treinado, em seguida, é lançado em produção, e roda sem aprender mais nada; apenas aplicando o que aprendeu. Isso é chamado de *aprendizado offline*.

Se você quiser que um sistema de aprendizado em lote conheça novos dados (como um novo tipo de spam), é preciso treinar uma nova versão do sistema a partir do zero no conjunto completo de dados (não apenas os novos dados, mas também os antigos), então parar o sistema antigo e substituí-lo pelo novo.

**Tipos de Sistemas do Aprendizado de Máquina | 15**

Felizmente, todo o processo de treinamento, avaliação e lançamento de um sistema de Aprendizado de Máquina pode ser automatizado facilmente (como mostrado na Figura 1-3), então mesmo um sistema de aprendizado em lote pode se adaptar às mudanças. Basta atualizar os dados e treinar a partir do zero uma nova versão do sistema sempre que necessário.

Esta solução é simples e geralmente funciona bem, mas, com a utilização do conjunto completo de dados, o treinamento pode demorar muitas horas, então você normalmente treinará um novo sistema apenas a cada 24 horas ou mesmo semanalmente. Se seu sistema precisa se adaptar a dados que mudam rapidamente (por exemplo, prever os preços das ações), você precisa de uma solução mais reativa.

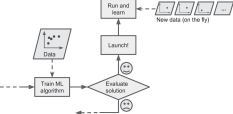
Além disso, o treinamento no conjunto completo de dados requer muitos recursos de computação (CPU, espaço de memória, espaço em disco, E/S do disco, E/S de rede, etc.). Se você tem muitos dados e seu sistema é automatizado para treinar todos os dias a partir do zero, esse processo ficará muito caro. Se a quantidade de dados for enorme, talvez seja impossível utilizar um algoritmo de aprendizado em lote.

Finalmente, se o seu sistema precisa ser capaz de aprender de forma autônoma e tem recursos limitados (por exemplo, um aplicativo de smartphone ou um rover em Marte), então, seria um grave erro carregar grandes quantidades de dados de treinamento e usar inúmeros recursos para treinar por horas todos os dias.

Felizmente, uma opção melhor em todos esses casos seria utilizar algoritmos capazes de aprender de forma incremental.

**Aprendizado online**

No *aprendizado online,* você treina o sistema de forma incremental, alimentando sequencialmente as instâncias de dados individualmente ou em pequenos grupos, cha mados de *minilotes*. Cada etapa do aprendizado é rápida e barata, então o sistema pode aprender rapidamente sobre os novos dados assim que eles chegam (veja Figura 1-13).

Execute e 

aprendaNovos dados (durante a ação)

Lance!

Dados

Treine o

algoritmo AM

*Figura 1-13. Aprendizado online*

Avalie a solução

**16 | Capítulo 1: O Cenário do Aprendizado de Máquina**

O aprendizado online é excelente para sistemas que recebem dados como um fluxo contínuo (por exemplo, preços das ações) e precisam se adaptar às mudanças rápida ou autonoma mente. Também é uma boa opção se seus recursos de computação são limitados: uma vez que um sistema de aprendizado online aprendeu sobre as novas instâncias de dados, ele não precisa mais delas, então você pode descartá-las (a menos que queira reverter para um estágio anterior e “reproduzir” os dados). O que pode economizar muito 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 (isto é chamado de *out-of-core learning*). O algoritmo carrega parte dos dados, executa uma etapa do treinamento nesses dados e repete o processo até que ele tenha sido exe

cutado em todos os dados (veja Figura 1-14).

Todo esse processo geralmente é feito ofine (ou seja, não no sistema

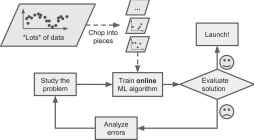
ao vivo), então *aprendizado online* pode ser um nome confuso. Pense

nisso como *aprendizado incremental*.

\*Muitos\* dados

Estude o

problema

Corte em 

pedaços

Treine **online** o

algoritmo AM

Analise

erros

Lance!

Avalie a solução

*Figura 1-14. Utilizando aprendizado online para lidar com grandes conjuntos de dados*

Um parâmetro importante dos sistemas de aprendizado online é a rapidez com que eles devem se adaptar às mudanças dos dados: isto é chamado de *taxa de aprendizado*. Se você definir uma alta taxa de aprendizado, seu sistema se adaptará rapidamente aos novos dados, mas também tenderá a se esquecer rapidamente os 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; isto é, aprenderá mais devagar, mas também será menos sensível ao apontar novos dados ou sequências de pontos de dados não representativos.

**Tipos de Sistemas do Aprendizado de Máquina | 17**

Um grande desafio no aprendizado online é que, se incluirmos dados ruins no sistema, seu desempenho diminuirá gradualmente. Se estamos falando de um sistema ao vivo, seus clientes perceberão. Por exemplo, dados ruins podem vir de um sensor com mau funcionamento em um robô, ou de alguém que envia um spam a um mecanismo de pesquisa para tentar se posicionar no topo. Para reduzir esse risco, você precisa moni

torar de perto o sistema e desligar o aprendizado rapidamente se detectar uma queda no desempenho (e possivelmente reverter para um estágio anterior de trabalho). Você também poderá monitorar os dados de entrada e reagir a dados anormais (por exemplo, com a utilização de um algoritmo de detecção de anomalias).

**Aprendizado Baseado em Instância Versus**

**Aprendizado Baseado em Modelo**

Mais uma forma de categorizar os sistemas de Aprendizado de Máquina é por meio da *generalização*. A maioria das tarefas de Aprendizado de Máquina faz previsões. Isso significa que, dada uma série de exemplos de treinamento, o sistema precisa ser capaz de generalizar em exemplos que nunca viu antes. Ter uma boa medida do desempenho nos dados de treinamento é bom, mas insuficiente; o verdadeiro objetivo é ter um bom desempenho em novas instâncias.

Existem duas abordagens principais para a generalização: aprendizado baseado em instâncias e aprendizado baseado em modelo.

**Aprendizado baseado em instância**

Possivelmente, a forma mais trivial de aprendizado é simplesmente decorar. Se você fosse criar um filtro de *spam* desta forma, ele apenas marcaria todos os e-mails que são idênticos em relação aos e-mails que já foram marcados pelos usuários — não seria a pior solução, mas certamente não é a melhor.

Em vez de marcar apenas e-mails que são idênticos aos e-mails de spam conhecidos, seu filtro de spam pode ser programado para marcar também e-mails que são muito seme lhantes aos e-mails conhecidos de spam. Isso requer uma *medida de similaridade* 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 é chamado de *aprendizado baseado em instância*: o sistema aprende os exemplos por meio da memorização e, em seguida, generaliza para novos casos utilizando uma medida de similaridade (Figura 1-15).

**18 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Característica 2** 

Nova instância

Treinando instâncias

**Característica 1**

*Figura 1-15. Aprendizado baseado em instância*

**Aprendizado baseado em modelo**

Outra maneira de generalizar a partir de um conjunto de exemplos seria construir um modelo desses exemplos e utilizar esse modelo para fazer *previsões*. Isso é chamado de *aprendizado baseado em modelo* (Figura 1-16).

**Característica 2** 

**Característica 1**

**Modelo**

Nova instância

*Figura 1-16. Aprendizado baseado em modelo*

Por exemplo, suponha que você queira saber se o dinheiro faz as pessoas felizes; então você baixa os dados do *Better Life Index* no site da OCDE (*https://goo.gl/0Eht9W*), bem como as estatísticas sobre o PIB per capita no site do FMI (*http://goo.gl/j1MSKe*). Logo, você junta as tabelas e classifica pelo PIB per capita. A Tabela 1-1 exemplifica uma amostra do que você obtém.

*Tabela 1-1. O dinheiro torna as pessoas mais felizes?*

**País PIB 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

**Tipos de Sistemas do Aprendizado de Máquina | 19**

Vamos plotar os dados para alguns países aleatórios (Figura 1-17).

a

d

i

v

e

d

o

ã

ç

a

f

s

i

t

a

S

HungriaCoreiaFrançaAustráliaEUA PIB per capita 

*Figura 1-17. Você vê uma tendência aqui?*

Parece haver uma tendência aqui! Embora os dados estejam *ruidosos* (ou seja, parcialmente aleatórios), parece que a satisfação de vida aumenta de forma mais ou menos linear à medida que aumenta o PIB per capita do país. Então você decide modelar a satisfação de vida [*life\_satisfaction*] como uma *função linear* do PIB per capita [*GDP\_per\_capita*]. Esta etapa é chamada de *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*

*life\_satisfaction* = *θ*0 + *θ*1 × *GDP\_per\_capita*

Este modelo tem dois *parâmetros do modelo*, *θ*₀ e *θ*.*5* Ao ajustar esses parâmetros, você pode fazer com que seu modelo represente qualquer função linear, como mostrado na Figura 1-18.

a 

d

i

v

e

d

o

ã

ç

a

f

s

i

t

a

S

PIB per capita

*Figura 1-18. Alguns modelos lineares possíveis*

5 Por convenção, a letra grega *θ (theta)* é frequentemente utilizada para representar os parâmetros do modelo.

**20 | Capítulo 1: O Cenário do Aprendizado de Máquina**

Antes de poder utilizar seu modelo, você precisa definir os valores dos parâmetros *θ*0 e *θ*1. Como você pode saber quais valores farão seu modelo funcionar melhor? Para responder a esta pergunta, você precisa especificar uma medida de desempenho. Você pode definir uma *função de utilidade* (ou *função fitness*) que mede o quão *bom* seu modelo é, ou uma *função de custo*, que mede o quão *ruim* ele é. Para problemas de regressão linear, as pes

soas geralmente utilizam uma função de custo que mede a distância entre as previsões do modelo linear e os exemplos de treinamento; o objetivo é minimizar essa distância.

É aqui que o algoritmo de regressão linear entra: você o alimenta com seus exemplos de treinamento e ele encontra os parâmetros que tornam o modelo linear mais adequado aos seus dados. Isso é chamado de *treinar* o modelo. No nosso caso, o algoritmo descobre que os valores dos parâmetros otimizados são *θ*₀ = 4,85 e *θ* = 4,91 × 10-5.

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

a 

d

i

v

e

d

o

ã

ç

a

f

s

i

t

a

S

PIB per capita

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

Você finalmente está pronto para executar o modelo e fazer previsões. Por exemplo, digamos que 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 fazer uma boa previsão: você olha para o PIB per capita do Chipre, encontra US$22.587 e, em seguida, aplica seu modelo e verifica que a satisfação de vida estará provavelmente ao redor de 4,85 + 22.587 × 4,91 × 10-5 = 5,96.

Para estimular o seu apetite, o Exemplo 1-1 mostra o código Python que carrega os dados, os prepara,*6* cria um diagrama de dispersão para visualização e, em seguida, treina um modelo linear e faz uma previsão.*7*

6 O código assume que prepare\_country\_stats() já está definido: ele mescla os dados de PIB e de satisfação de vida em um único *dataframe* do Pandas.

7 Tudo bem se você não entender todo o código ainda; apresentaremos o Scikit-Learn nos próximos capítulos.

**Tipos de Sistemas do Aprendizado de Máquina | 21**

*Exemplo 1-1. Treinando e executando um modelo linear utilizando o Scikit-Learn*

import matplotlib

import matplotlib

import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

import numpy as np

import numpy as np

import pandas as pd

import pandas as pd

import sklearn

import sklearn

# Carregue os dados

# Carregue os dados

oecd\_bli = pd.read\_csv("oecd\_bli\_2015.csv", thousands=',')

oecd\_bli = pd.read\_csv("oecd\_bli\_2015.csv", thousands=',')

gdp\_per\_capita = pd.read\_csv("gdp\_per\_capita.csv",thousands=',',delimiter='\t', gdp\_per\_capita = pd.read\_csv("gdp\_per\_capita.csv",thousands=',',delimiter='\t', encoding='latin1', na\_values="n/a") encoding='latin1', na\_values="n/a")

# Prepare os dados

# Prepare os dados

country\_stats = prepare\_country\_stats(oecd\_bli, gdp\_per\_capita) country\_stats = prepare\_country\_stats(oecd\_bli, gdp\_per\_capita) X = np.c\_[country\_stats["GDP per capita"]]

X = np.c\_[country\_stats["GDP per capita"]]

y = np.c\_[country\_stats["Life satisfaction"]]

y = np.c\_[country\_stats["Life satisfaction"]]

# Visualize os dados

# Visualize os dados

country\_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction') country\_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction') plt.show()

plt.show()

# Selecione um modelo linear

# Selecione um modelo linear

model = sklearn.linear\_model.LinearRegression()

model = sklearn.linear\_model.LinearRegression()

# Treine o modelo

# Treine o modelo

model.fit(X, y)

model.fit(X, y)

# Faça uma predição para o Cyprus

# Faça uma predição para o Cyprus

X\_new = [[22587]] # Cyprus' GDP per capita

X\_new = [[22587]] # Cyprus' GDP per capita

print(model.predict(X\_new)) # outputs [[ 5.96242338]]

print(model.predict(X\_new)) # outputs [[ 5.96242338]]

Se você tivesse optado por utilizar um algoritmo de aprendizado ba

seado em instâncias, teria visto que a Eslovênia tem o PIB per capita mais próximo do Chipre (US$20.732), e, como os dados da OCDE nos dizem que a satisfação da vida dos eslovenos é 5,7, você teria pre

visto uma satisfação de vida de 5,7 para o Chipre. Se você reduzir um pouco e observar os dois países mais próximos, encontrará Portugal e Espanha com satisfações de vida de 5,1 e 6,5, respectivamente. Fa

zendo a média desses três valores, obtém-se 5,77, o que é bastante próximo da sua previsão baseada em modelo. Este algoritmo simples é chamado de regressão *k-nearest neighbors* (neste exemplo, k = 3).

Para substituir o modelo de regressão linear com a regressão *k-nearest neighbors* no código anterior, simplesmente substitua esta linha:

model = sklearn.linear\_model.LinearRegression()

por esta:

model = sklearn.neighbors.KNeighborsRegressor(n\_neighbors=3)

**22 | Capítulo 1: O Cenário do Aprendizado de Máquina**

Se tudo correu bem, seu modelo fará boas previsões. Caso contrário, talvez seja necessário utilizar mais atributos (taxa de emprego, saúde, poluição do ar, etc.), obter mais ou melhor qualidade dos dados de treinamento, ou talvez selecionar um modelo mais poderoso (por exemplo, um modelo de Regressão Polinomial).

Em resumo:

• Você estudou os dados;

• Selecionou um modelo;

• Treinou-o com os dados de treinamento (ou seja, o algoritmo de aprendizado pro curou os valores dos parâmetros do modelo que minimizam uma função de custo);

• Finalmente, você aplicou o modelo para fazer previsões em novos casos (isto é chamado de *Inferência*), na expectativa de que esse modelo generalize bem.

Um projeto típico de Aprendizado de Máquina é assim. No Capítulo 2, você experimen tará isso em primeira mão, de ponta a ponta em um projeto.

Já caminhamos muito até aqui: agora você já sabe o que é realmente o Aprendizado de Máquina, por que ele é útil, quais são algumas das categorias mais comuns de sistemas do AM e como é um fluxo de trabalho típico do projeto. Agora, veremos o que pode dar errado no aprendizado e impedir que você faça previsões precisas.

**Principais Desafios do Aprendizado de Máquina**

Em suma, uma vez que a sua tarefa principal é selecionar um algoritmo de aprendizado e treiná-lo em alguns dados, as duas coisas que podem dar errado são: “algoritmos ruins” e “dados ruins”. Comecemos com exemplos de dados ruins.

**Quantidade Insuficiente de Dados de Treinamento**

Para que uma criança aprenda o que é uma maçã, é preciso que você aponte para uma maçã e diga “maçã” (possivelmente repetindo algumas vezes esse procedimento). Agora, a criança consegue reconhecer maçãs em todos os tipos de cores e formas. Genial.

O Aprendizado de Máquina ainda não está lá; é preciso uma grande quantidade de dados para que a maioria dos algoritmos de Aprendizado de Máquina funcione corretamente. Você precisará de milhares de exemplos mesmo para problemas muito simples, e para problemas complexos, como reconhecimento de imagem ou da fala, precisará de milhões de exemplos (a menos que possa reutilizar partes de um modelo existente).

**Principais Desafios do Aprendizado de Máquina | 23**

| **A Eficácia Não Razoável dos Dados**  Em um artigo famoso (*http://goo.gl/R5enIE*) publicado em 2001, os pesquisadores da Microsoft Michele Banko e Eric Brill mostraram que, uma vez que dados suficientes foram fornecidos, algoritmos do aprendizado de máquina muito diferentes, mesmo os mais simples, tiveram um desempenho quase idêntico em um problema complexo de desambiguação*8* (como você pode ver na Figura 1-20).  o  ã  s  i  c  e  r  P    e  d    e  t  s  e  T  Baseado em Memória  Winnow  Perceptron  Naive Bayes  Milhões de Palavras  *Figura 1-20. A importância dos dados versus algoritmos9*  Como os autores afrmam: “esses resultados sugerem que talvez possamos reconsiderar o cus to-benefício entre gastar tempo e dinheiro no desenvolvimento de algoritmos ou gastá-los no desenvolvimento de *corpus.*”  A ideia de que, para problemas complexos, os dados são mais importantes do que algorit mos foi popularizada por Peter Norvig *et al.* em um artigo intitulado “Te Unreasonable Efectiveness of Data” [“A Efcácia Não Razoável dos Dados”, em tradução livre] (*http:// goo.gl/q6LaZ8*) publicado em outubro de 2009.*10* No entanto, vale notar que os conjuntos de dados de pequenas e médias dimensões ainda são muito comuns e nem sempre é fácil ou barato obter dados extras de treinamento, então não abandone os algoritmos ainda. |
| --- |

8 Por exemplo, em inglês, saber quando escrever "to", "two", ou "too" dependendo do contexto. 9 Figura reproduzida com permissão de Banko e Brill (2001), “Learning Curves for Confusion Set Disambiguation”. 10 “The Unreasonable Effectiveness of Data” Peter Norvig *et al.* (2009).

**24 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Dados de Treinamento Não Representativos**

A fim de generalizar bem, é crucial que seus dados de treinamento sejam representati vos dos novos casos para os quais você deseja generalizar, não importa se você utiliza 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 mostra como ficam os dados quando os países que faltam são adicionados.

a

di

v

e

d

o

ã

ç

a

f

s

i

t

a

S

Brasil Brasil Méxic Méxicoo Chile Ce República Checa República Checa PIB per capita 

Noruega Suíça Luxemburgo

*Figura 1-21. Uma amostra de treinamento mais representativa*

Se você treinar um modelo linear nesses dados, obterá a linha sólida, enquanto o modelo antigo está representado pela linha pontilhada. Como pode ser observado, adicionar alguns países faltantes não só altera significativamente o modelo, como deixa claro que um modelo linear tão simples provavelmente nunca funcionará bem. Parece que países muito ricos não são mais felizes do que países moderadamente ricos (na verdade, parecem mais infelizes) e, inversamente, alguns países pobres parecem ser mais felizes do que muitos países ricos.

Ao utilizar um conjunto de treinamento não representativo, treinamos um modelo que dificilmente fará previsões precisas, especialmente para países muito pobres e muito ricos.

É crucial utilizar um conjunto de treinamento representativo nos casos em que desejamos generalizar. Isso pode ser 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.

**Um exemplo famoso 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* con duziu uma grande 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 confança que Landon deveria obter 57% dos votos.

**Principais Desafios do Aprendizado de Máquina | 25**

Mas, na verdade, Roosevelt venceu as eleições com 62% dos votos. A falha aconteceu no mé todo de amostragem da *Literary Digest*:

• Em primeiro lugar, ao obter os endereços para o envio das pesquisas, a *Literary Digest* utilizou listas telefônicas, de assinantes, de membros de clubes, dentre outras. Todas essas listas tendem a favorecer pessoas mais ricas, mais propensas a votar em republi canos (daí 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 falta de resposta*.

Aqui temos outro exemplo: digamos que você deseja criar um sistema para o reconhe cimento de vídeos de música funk. Uma forma de construí-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 tenden ciosos 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?

**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 subjacentes, portanto, menos propício a um bom funciona mento. Muitas vezes vale a pena perder um tempo limpando seus dados de treinamento. A verdade é que a maioria dos cientistas de dados gasta uma parte significativa de seu tempo fazendo exatamente isso. Por exemplo:

• Se algumas instâncias são claramente outliers*,* isso pode ajudar a descartá-las ou tentar manualmente a correção dos erros;

• Se faltam algumas características para algumas instâncias (por exemplo, 5% dos seus clientes não especificaram sua idade), você deve decidir se deseja ignorar completa mente 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 ela, e assim por diante.

**26 | Capítulo 1: O Cenário do Aprendizado de Máquina**

**Características Irrelevantes**

Como diz o ditado: entra lixo, sai lixo. Seu sistema só será capaz de aprender se os dados de treinamento contiverem características relevantes suficientes e poucas características irrelevantes. Uma parte crítica do sucesso de um projeto de Aprendizado de Máquina é criar um bom conjunto de características para o treinamento, processo chamado de *feature engeneering* que envolve:

• *Seleção das características*: selecionar as mais úteis para treinar entre as existentes;

• *Extração das características*: combinar características existentes para produzir uma mais útil (como vimos anteriormente, os algoritmos de redução de dimen sionalidade podem ajudar);

• Criação de novas características ao coletar novos dados.

Agora que examinamos vários exemplos ruins de dados, vejamos alguns exemplos ruins de algoritmos.

**Sobreajustando os dados de treinamento**

Digamos que você está visitando um país estrangeiro e o taxista o rouba. Você é tentado a dizer que *todos* os motoristas de táxi nesse país são ladrões. Generalizar é algo que nós humanos fazemos com muita frequência e, infelizmente, as máquinas podem cair na mesma armadilha se não tivermos 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 mostra um exemplo de um modelo polinomial de satisfação de vida de alto grau que se sobreajusta fortemente nos dados de treinamento. Você realmente confiaria em suas previsões, se o seu desempenho é muito melhor em dados de treinamento do que no modelo linear simples?

a 

d

i

v

e

d

o

ã

ç

a

f

s

i

t

a

S

PIB per capita

*Figura 1-22. Sobreajustando nos dados de treinamento*

**Principais Desafios do Aprendizado de Máquina | 27**

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 (o que intro duz ruído na amostragem), então 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 seu 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 (7,3), Norway (7,4), Sweden (7,2) e Switzerland (7,5). Você confiaria nesta regra que também generalizará para países como Rwanda ou Zimbabwe? É evidente que esse padrão nos dados ocorreu por acaso, mas o modelo não tem como saber se um padrão é real ou simplesmente resultado de ruído nos dados.

O sobreajuste acontece quando o modelo é muito complexo em rela

ção 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 polino

mial de alto grau), 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 possui dois parâmetros, *θ*0 e *θ*1. Isso dá 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 *θ*1 = 0 o algoritmo teria apenas um grau de liberdade e teria muito mais dificuldade para ajustar os dados corretamente: ele apenas poderia mover a linha para cima ou para baixo para chegar o mais próximo possível das instâncias de treinamento e finalizar em torno da média. Este é um modelo muito simples! Mas, se permitirmos que o algoritmo modifique *θ*1 e forçarmos para que ele se mantenha reduzido, então o algoritmo efetivamente terá algo entre um e dois graus de liberdade. Ele produzirá um modelo mais simples do que aquele com dois graus de liberdade, mas mais complexo do que o modelo com apenas um. Queremos encontrar o equilíbrio certo entre o ajuste perfeito dos dados e a necessidade de manter o modelo simples o suficiente para garantir que ele generalize bem.

**28 | Capítulo 1: O Cenário do Aprendizado de Máquina**

A Figura 1-23 mostra três modelos: a linha pontilhada representa o modelo original que foi treinado com poucos países, a linha tracejada é o nosso segundo modelo trei nado com todos os países, e a linha sólida é um modelo linear treinado com os mes mos dados do primeiro modelo, mas com uma restrição de regularização. Veja que a regularização forçou o modelo a ter uma inclinação menor, o que faz com que tenha um ajuste menor dos dados no modelo treinado mas, na verdade, permite generalizar melhor em novos exemplos.

a 

d

i

v

e

d

o

ã

ç

a

f

s

i

t

a

S

Modelo linear em todos os dados

Modelo linear em dados parciais

Modelo linear regularizado em dados parciais PIB per capita

*Figura 1-23. A regularização reduz o risco de sobreajustes*

A quantidade de regularização a ser aplicada por um *hiperparâmetro* durante o apren dizado pode ser controlada. Um hiperparâmetro é um parâmetro de um algoritmo de aprendizado (não do modelo). Como tal, não é afetado pelo próprio algoritmo de apren dizado; deve ser definido antes do treinamento e permanecer constante durante ele. Se você configurar o hiperparâmetro de regularização para 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, mas 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).

**Subajustando os dados de Treinamento**

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

As principais opções para resolver esses problemas são:

• Selecionar um modelo mais poderoso, com mais parâmetros;

**Principais Desafios do Aprendizado de Máquina | 29**

• Alimentar o algoritmo de aprendizado com melhores características (*feature engineering*);

• Reduzir as restrições no modelo (por exemplo, reduzindo o hiperparâmetro de regularização).

**Voltando Atrás**

Agora você já sabe muito sobre o Aprendizado de Máquina. No entanto, passamos por tantos conceitos que você pode estar se sentindo um pouco perdido, então vamos dar um passo atrás e olhar para o quadro geral:

• Aprendizado de Máquina é fazer com que as máquinas evoluam em algu mas tarefas aprendendo com os dados, em vez de ter que programar as regras explicitamente;

• Existem muitos tipos diferentes de sistemas do AM: supervisionados ou não, em lote ou online, baseados em instâncias ou em modelos, e assim por diante;

• Em um projeto do AM, você coleta dados em um conjunto de treinamento e os for nece para um algoritmo de aprendizado. Se o algoritmo estiver baseado em modelo, ele ajusta alguns parâmetros para adequar o modelo ao conjunto de treinamento (ou seja, para fazer boas previsões no próprio conjunto de treinamento), e então, se tudo der certo, também poderá fazer boas previsões em novos casos. Se o algoritmo for baseado em instância, ele simplesmente decora os exemplos e utiliza uma medi da de similaridade generalizando para novas instâncias;

• O sistema não terá um bom funcionamento se o seu 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 seu modelo não precisa ser nem 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 para novos casos. Você vai querer avaliar e ajustar se necessário. Vejamos como.

**Testando e Validando**

A única maneira de saber o quão bem um modelo generalizará em novos casos é de fato testá-lo em novos casos. Uma forma de pôr isso em prática é colocar seu modelo em produção e monitorar a qualidade do seu desempenho. Isso funciona bem, mas, se seu modelo for muito ruim, seus usuários se queixarão — e esta não é a melhor ideia.

**30 | Capítulo 1: O Cenário do Aprendizado de Máquina**

Uma opção melhor seria dividir seus dados em dois conjuntos: o *conjunto de treinamento* e o *conjunto de teste*. Como esses nomes implicam, você treina seu modelo utilizando o conjunto de treinamento e o testa utilizando o conjunto de teste. A taxa de erro em novos casos é chamada de *erro de generalização* (ou *erro fora da amostra*) e, ao avaliar seu modelo no conjunto de teste, você obtém uma estimativa desse erro. O valor indica se o seu modelo terá um bom funcionamento em instâncias inéditas.

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

É comum utilizar 80% dos dados para treinamento e *reservar* 20%

para o teste.

É muito simples avaliar um modelo: basta utilizar um conjunto de teste. Agora, suponha que você hesite entre dois modelos (digamos um modelo linear e um modelo polinomial): como decidir? Uma opção será treinar ambos utilizando o conjunto de teste e comparar se eles generalizam bem.

Agora, suponha que o modelo linear generalize melhor, mas você deseja aplicar algu ma regularização para evitar o sobreajuste. A questão é: como você escolhe o valor do hiperparâmetro de regularização? Uma opção seria treinar 100 modelos diferentes utilizando 100 valores diferentes para este hiperparâmetro. Suponha que você encontre o melhor valor de hiperparâmetro que produza um modelo com o menor erro de gene ralização, digamos, apenas 5% de erro.

Então, você coloca esse modelo em produção, mas infelizmente ele não funciona tão bem quanto o esperado e produz 15% de erros. O que acabou de acontecer?

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

Uma solução comum para este problema é ter um segundo conjunto de retenção chamado *conjunto de validação*. Você treina vários modelos com vários hiperparâmetros utilizando o conjunto de treinamento, seleciona o modelo e os hiperparâmetros que atuam melhor no conjunto de validação, e, quando estiver contente com seu modelo, executa um único teste final no conjunto de teste para obter uma estimativa do erro de generalização.

**Testando e Validando | 31**

Para evitar “desperdiçar” muitos dados de treinamento em conjuntos de validação, uma técnica comum é utilizar a *validação cruzada*: o conjunto de treinamento é dividido em subconjuntos complementares e cada modelo é treinado com uma combinação diferente desses subconjuntos e validado em relação às partes restantes. Uma vez selecionados o tipo de modelo e os hiperparâmetros, um modelo final é treinado com a utilização desses hiperparâmetros no conjunto completo de treinamento e o erro generalizado é medido no conjunto de testes.

| **Teorema do Não Existe Almoço Grátis**  Um modelo é uma versão simplifcada das observações. As simplifcações destinam-se a descartar os detalhes supérfuos que provavelmente não serão generalizados em novas instâncias. 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 pode ser ignorado com segurança.  Em um famoso artigo de 1996 (*https://goo.gl/dzp946*),*11* David Wolpert demonstrou que, se você não fzer suposição alguma sobre os dados, então não há motivo para preferir um ou outro modelo. Isso é chamado de Teorema do *Não existe almoço grátis* (NFL, do inglês "no free lunch"). Para alguns conjuntos de dados, o melhor modelo é um modelo linear, enquanto para outros conjuntos será uma rede neural. Não existe um modelo que tenha a garantia de fun  cionar melhor a priori (daí o nome do teorema). A única maneira de saber com certeza qual seria o melhor modelo é a avaliação de todos. Como isso não é possível, na prática você parte de alguns pressupostos razoáveis sobre os dados e avalia apenas alguns modelos razoáveis. Por exemplo, para tarefas simples, você pode avaliar modelos lineares com vários níveis de regula  rização e, para um problema complexo, você pode avaliar várias redes neurais. |
| --- |

**Exercícios**

Neste capítulo, abordamos alguns dos conceitos mais importantes do Aprendizado de Máquina. Nos próximos, vamos mergulhar mais fundo e escrever mais código, mas, antes disso, certifique-se de responder as seguintes perguntas:

1. Como você definiria o Aprendizado de Máquina?

2. Você saberia nomear quatro tipos de problemas nos quais o AM se destaca? 3. O que é um conjunto de treinamento rotulado?

4. Quais são as duas tarefas supervisionadas mais comuns?

5. Você consegue nomear quatro tarefas comuns sem supervisão? 11 “The Lack of A Priori Distinctions Between Learning Algorithms”, D. Wolperts (1996).

**32 | Capítulo 1: O Cenário do Aprendizado de Máquina**

6. Que tipo de algoritmo de Aprendizado de Máquina você utilizaria para permitir que um robô ande em vários terrenos desconhecidos?

7. Que tipo de algoritmo você utilizaria para segmentar seus clientes em vários grupos?

8. Você enquadraria o problema da detecção de spam como um problema de aprendizado supervisionado ou sem supervisão?

9. O que é um sistema de aprendizado online?

10. O que é o *out-of-core learning*?

11. Que tipo de algoritmo de aprendizado depende de uma medida de similaridade para fazer previsões?

12.Qual a diferença entre um parâmetro de modelo e o hiperparâmetro do algoritmo de aprendizado?

13. O que os algoritmos de aprendizado baseados em modelos procuram? Qual é a estratégia mais comum que eles utilizam para ter sucesso? Como fazem previsões?

14. Você pode citar quatro dos principais desafios no Aprendizado de Máquina?

15. Se o seu modelo é ótimo nos dados de treinamento, mas generaliza mal para no vas instâncias, o que está acontecendo? Você pode nomear três soluções possíveis?

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 pode dar errado se você ajustar os hiperparâmetros utilizando o conjunto de teste?

19. O que é validação cruzada e por que você preferiria usá-la ao invés de um conjunto de validação?

As soluções para estes exercícios estão disponíveis no Apêndice A.

**Exercícios | 33**

**Capítulo 2**

**Projeto de Aprendizado de Máquina de Ponta a Ponta**

Neste capítulo você verá um exemplo de projeto de ponta a ponta na pele de um cientista de dados recentemente contratado por uma empresa do mercado imobiliário.*1* Você seguirá estes principais passos:

1. Olhar para o quadro geral;

2. Obter os dados;

3. Descobrir e visualizar os dados para obter informações;

4. Preparar os dados para os algoritmos do Aprendizado de Máquina; 5. Selecionar e treinar um modelo;

6. Ajustar o seu modelo;

7. Apresentar sua solução;

8. Lançar, monitorar e manter seu sistema.

**Trabalhando com Dados Reais**

É melhor experimentar com dados do mundo real e não apenas conjuntos de dados arti ficiais ao estudar Aprendizado de Máquina. Felizmente, existem milhares de conjuntos de dados disponíveis a sua escolha, variando em todos os tipos de domínios. Você pode procurar em muitos lugares, tais quais:

• Repositórios Populares de *open data*:

— UC Irvine Machine Learning Repository (*http://archive.ics.uci.edu/ml/*)

1 Este exemplo de projeto é completamente fictício. O objetivo é apenas ilustrar os níveis principais de um projeto de Aprendizado de Máquina, não aprender sobre o mercado imobiliário.

**35**

— Conjunto de dados no Kaggle (*https://www.kaggle.com/datasets*) — Conjunto de Dados no AWS da Amazon (*http://aws.amazon.com/fr/datasets/*)

• Meta portais (eles listam repositórios *open data*):

— *http://dataportals.org/*

— *http://opendatamonitor.eu/*

— *http://quandl.com/*

• Outras páginas que listam muitos repositórios populares de *open data*:

— Lista de conjuntos de dados de Aprendizado de Máquina do Wikipedia (*https:// goo.gl/SJHN2k*)

— Pergunta no Quora.com (*http://goo.gl/zDR78y*)

— Conjuntos de dados no Reddit (*https://www.reddit.com/r/datasets*)

Neste capítulo, escolhemos o conjunto de dados do repositório StatLib referente a pre ços do setor imobiliário na Califórnia*2* (veja a Figura 2-1). Este conjunto de dados foi baseado no censo de 1990 na Califórnia. Não são dados recentes (naquela época, ainda era possível comprar uma casa agradável perto da Baía de São Francisco), mas possuem muitas qualidades para o aprendizado, então vamos fingir que são dados recentes. Também adicionamos um atributo categórico e removemos algumas características para fins de ensino.

População 

s

a

s

a

C

s

a

e

d

d

u

o

t

i

i

t

d

a

é

L

M

r

o

l

a

V

Longitude

*Figura 2-1. Preços de casas na Califórnia*

2 O conjunto de dados original figurou em R. Kelley Pace Ronald Barry, “Sparse Spatial Autoregressions”, Statistics & Probability Letters 33, nº 3 (1997): 291–297.

**36 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

**Um Olhar no Quadro Geral**

Bem-vindo à Machine Learning Housing Corporation! A primeira tarefa que você executará será construir um modelo de preços do setor imobiliário utilizando os dados do censo da Califórnia. Esses dados têm métricas como população, renda média, preço médio do setor imobiliário e assim por diante para cada grupo de bairros. Os grupos de bairros são a menor unidade geográfica para a qual o *US Census Bureau* publica dados de amostra (um grupo de bairros geralmente tem uma população de 600 a 3 mil pessoas). Para abreviar, os chamaremos de “bairros”.

Seu modelo deve aprender com esses dados e ser capaz de prever o preço médio em qualquer bairro, considerando todas as outras métricas.

Como um cientista de dados bem organizado, a primeira coisa a fa

zer é obter sua lista de verifcação do projeto. Você pode começar

com a lista do Apêndice B, que deve funcionar razoavelmente bem

para a maioria dos projetos de Aprendizado de Máquina, mas asse

gure-se de adaptá-la às suas necessidades. Neste capítulo, passaremos

por muitos itens da lista de verifcação, mas também ignoraremos

alguns porque eles são autoexplicativos ou porque serão discutidos

em capítulos posteriores.

**Enquadre o Problema**

A primeira pergunta a ser feita ao seu chefe é: qual é exatamente o objetivo comercial? Provavelmente construir um modelo não será o objetivo final. Como a empresa espera usar e se beneficiar deste modelo? Isso é importante porque determinará como você en quadrará o problema, quais algoritmos selecionará, que medida de desempenho utilizará para avaliar seu modelo e quanto esforço você deve colocar nos ajustes.

Seu chefe responde que o resultado do seu modelo (uma previsão do preço médio do setor imobiliário no bairro) será enviado para outro sistema de Aprendizado de Máquina (veja a Figura 2-2), juntamente com muitos outros *sinais.3* Esse sistema de downstream deter minará se vale a pena investir em uma determinada área ou não. É fundamental acertar nessa etapa, uma vez que afetará diretamente a receita.

3 Dados fornecidos para um sistema de Aprendizado de Máquina são chamados de sinais em referência à teoria da informação de Shannon: você quer que a proporção sinal/ruído seja alta.

**Um Olhar no Quadro Geral | 37**

Seu componente Outros sinais

Componentes Preços do

Análise de

ascendentes

Dados do Bairro

Bairro

Investimento 

Preços do Bairro Investimentos

*Figura 2-2. Um pipeline de Aprendizado de Máquina para investimentos imobiliários*

| **Pipelines**  Uma sequência de *componentes* de processamento de dados é chamada de *pipeline* de dados. Os pipelines são muito comuns em sistemas do Aprendizado de Máquina, uma vez que exis tem muitos dados para manipular e muitas transformações para aplicar neles.  Os componentes normalmente rodam de forma assíncrona. Cada componente puxa uma grande quantidade de dados, os processa e envia o resultado para outro armazenador dados, e então, algum tempo depois, o próximo componente no pipeline puxa esses dados e envia sua própria saída, e assim por diante. Cada componente é bastante autônomo: a interface entre os componentes é simplesmente o armazenamento de dados. Isso faz com que entender o siste  ma seja bastante simples (com a ajuda de um grafo do fuxo de dados), e diferentes equipes podem se concentrar em diferentes componentes. Além disso, se um componente se romper, os componentes de downstream geralmente continuam a funcionar normalmente (pelo me nos por um tempo), utilizando apenas a última saída do componente rompido. Isso torna a arquitetura bastante robusta.  Por outro lado, se um monitoramento adequado não for implementado, um componente rompido pode passar despercebido por algum tempo. Os dados fcarão obsoletos e o desem penho geral do sistema decairá. |
| --- |

A próxima pergunta a ser feita é: qual é a solução no momento (se houver)? Muitas vezes a resposta lhe dará um desempenho de referência, bem como informações sobre como resolver o problema. Seu chefe responde que os preços das casas nos bairros são estimados manualmente por especialistas: uma equipe reúne informações atualizadas sobre um bairro e, quando não conseguem obter o preço médio, o estimam utilizando regras complexas.

Este processo é caro e demorado, e suas estimativas não são boas; nos casos em que conseguem descobrir o preço médio real do setor imobiliário, geralmen te percebem que suas estimativas ficaram mais de 10% abaixo do valor. É por isso que a empresa acha útil treinar um modelo a partir de outros dados sobre esse bairro para prever o preço médio do setor imobiliário. Os dados do recenseamento

**38 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

parecem um ótimo conjunto de dados para serem explorados com este propósito, pois incluem os preços médios de milhares de bairros, bem como outros dados.

Bem, com toda essa informação você já está pronto para começar a projetar seu sistema. Primeiro, você precisa enquadrar o problema: será supervisionado, sem supervisão ou um Aprendizado por Reforço? Será uma tarefa de classificação, de regressão ou outra coisa? Você deve utilizar técnicas de aprendizado em lote ou online? Antes de ler, pause e tente responder para si estas perguntas.

Você encontrou as respostas? Vamos ver: claramente temos uma tarefa típica de aprendi zado supervisionado, uma vez que você recebe exemplos *rotulados* de treinamento (cada instância vem com o resultado esperado, ou seja, o preço médio do setor imobiliário do bairro). Além disso, também é uma tarefa típica de regressão, já que você é solicitado a prever um valor. Mais especificamente, trata-se de um problema de *regressão multiva riada*, uma vez que o sistema utilizará múltiplas características para fazer uma previsão (ele usará a população do bairro, a renda média, etc.). No primeiro capítulo, você previu níveis de satisfação de vida com base em apenas uma característica, o PIB per capita, de modo que era um problema de *regressão univariante*. Finalmente, não há um fluxo con tínuo de dados entrando no sistema, não há uma necessidade em especial de se ajustar rapidamente à mudança de dados, e os dados são pequenos o suficiente para caber na memória, de modo que o aprendizado simples em lote deve funcionar bem.

Se os dados fossem enormes, você poderia dividir seu trabalho de

aprendizado em lotes em vários servidores (utilizando a técnica

*MapReduce*, como veremos mais adiante), ou poderia utilizar uma

técnica de *aprendizado online*.

**Selecione uma Medida de Desempenho**

Seu próximo passo é selecionar uma medida de desempenho. Uma medida típica de desempenho para problemas de regressão é a *Raiz do Erro Quadrático Médio* (sigla RMSE, em inglês). Ela dá uma ideia da quantidade de erros gerados pelo sistema em suas previsões, com um peso maior para grandes erros. A Equação 2-1 mostra a fórmula matemática para calcular a RMSE.

*Equação 2-1. Raiz do Erro Quadrático Médio (RMSE)*

*m*

RMSE **X**, *h* = 1*m* ∑*i* = 1

*h* **x** *i* − *y i* 2

**Um Olhar no Quadro Geral | 39**

4z

**Notações**

Esta equação apresenta vários apontamentos muito comuns do Aprendizado de Máquina que usaremos ao longo deste livro:

• *m* é o número de instâncias no conjunto de dados no qual você está medindo o RMSE.

— Por exemplo, se estiver avaliando o RMSE em um conjunto de validação de 2 mil bairros, então *m* = 2 mil.

• x*(i)* é um vetor de todos os valores da característica (excluindo o *rótulo*) da *i-ésima* instância conjunto de dados, e *y(i)* é seu *rótulo* (o valor desejado da saída para aquela instância).

— Por exemplo, se o primeiro bairro no conjunto de dados estiver localizado na longitude –118,29°, latitude 33,91°, e ele tem 1.416 habitantes com uma renda média de US$ 38.372, e o valor médio da casa é de US$ 156.400 (ignorando as outras características por ora), então,

−118 , 29

33 , 91

1416

38372

e

**x** 1 =

*y* (1) = 156.400

• **X** é uma matriz que contém todos os valores da característica (excluindo rótulos) de todas as instâncias no conjunto de dados. Existe uma linha por instância e a *i-ésima* linha é igual a transposição de **x***(i)*, notado por (**x***(i)*)*T*.*4*

— Por exemplo, se o primeiro bairro for conforme descrito, então a matriz **X** fica assim: **x** 1 *T*

**X** =

**x** 2 *T*

**x** 1999 *T* **x** 2000 *T*

= −118,29 33 , 91 1416 38372

• *h* é a função de previsão do seu sistema, também chamada de *hipótese*. Quando seu sis tema recebe o vetor **x***(i)* da característica de uma instância, ele exibe um valor previsto *ŷ(i)* = *h*(**x***(i)*) para aquela instância (*ŷ* se pronuncia “y-chapéu”).

4 Lembre-se de que o operador de transposição vira um vetor de coluna em um vetor de linha (e vice-versa).

**40 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

— Por exemplo, se o seu sistema prevê que o preço médio no primeiro bairro é de US$ 158.400, então *ŷ*(1) = *h*(**x**(1)) = 158.400. A previsão de erro para este bairro é *ŷ*(1) – *y*(1) = 2.000.

• RMSE(**X**,*h*) é a função de custo medida no conjunto de exemplos utilizando sua hipótese *h*.

Usamos a fonte em itálico e caixa baixa para valores escalares (como *m* ou *y(i)*) e nomes de funções (como *h*), fonte em negrito e caixa baixa para vetores (como **x***(i)*) e em negrito e caixa alta para matrizes (como **X**).

Embora a RMSE seja geralmente a medida de desempenho preferencial para tarefas de regressão, em alguns contextos você pode preferir utilizar outra função. Por exemplo, suponha que existam vários bairros outliers. Nesse caso, você pode considerar usar o *Erro Médio Absoluto* (sigla MAE, em inglês) (também chamado de Desvio Médio Ab

soluto; ver Equação 2-2):

*Equação 2-2. Erro Médio Absoluto*

*m*

MAE **X**, *h* = 1*m* ∑*i* = 1

*h* **x** *i* − *y i*

Tanto a RMSE quanto o MAE são formas de medir a distância entre dois vetores: o vetor das previsões e o vetor dos valores alvo. Várias medidas de distância, ou *normas*, são possíveis:

• Calcular a raiz de uma soma de quadrados (RMSE) corresponde à *norma eucli diana*: é a noção de distância que você conhece. Também chamado de *norma* ℓ2, notado por ∥ · ∥2 (ou apenas ∥ · ∥).

• Calcular a soma de absolutos (MAE) corresponde à *norma* ℓ1, notado por ∥ · ∥1. Às vezes é chamado de *norma Manhattan* porque mede a distância entre dois pontos em uma cidade, se você só puder viajar ao longo de bairros ortogonais da cidade.

• Mais genericamente, a norma ℓ*k* de um vetor **v** que contém *n* elementos é defi nida como *k* = *v*0*k* + *v*1*k* + . . . + *vnk*1

∥ v *k*∥· ℓ0 apenas dá o número de elementos diferentes de zero no vetor, e ℓ∞ dá o valor máximo absoluto no vetor.

• Quanto maior o índice da norma, mais ela se concentra em grandes valores e ne gligencia os pequenos. É por isso que a RMSE é mais sensível a outliers do que o MAE. Mas, quando os outliers são exponencialmente raros (como em uma curva em forma de sino), a RMSE funciona muito bem, e geralmente é a preferida.

**Um Olhar no Quadro Geral | 41**

**Verifique as Hipóteses**

Por último, uma boa prática é listar e verificar as hipóteses que foram feitas até agora (por você ou outros); fica mais fácil pegar problemas sérios logo no início. Por exemplo, os preços dos bairros mostrados pelo seu sistema serão alimentados em um sistema downstream do Aprendizado de Máquina, e assumiremos que esses preços serão usados como tal. Mas, e se o sistema downstream converter os preços em categorias (por exemplo, “barato”, “médio” ou “caro”) e utilizar essas categorias em vez dos próprios preços? Neste caso, não é importante saber o preço exato se o seu sistema só precisa obter a categoria certa. Se for assim, então o problema deveria ter sido enquadrado como uma tarefa de classificação, não uma tarefa de regressão. Você não vai querer descobrir isso após trabalhar por meses em um sistema de regressão.

Felizmente, depois de conversar com a equipe responsável pelo sistema downstream, você está confiante de que realmente precisa dos preços reais e não apenas das categorias. Ótimo! Você está pronto, as luzes estão verdes e você pode começar a programar agora!

**Obtenha os Dados**

É hora de colocar as mãos na massa. Não hesite em pegar seu laptop e acompanhar os seguintes exemplos de código em um notebook do Jupyter, que está disponível na íntegra em *https://github.com/ageron/handson-ml*.

**Crie o Espaço de Trabalho**

Primeiro, você precisará ter o Python instalado. Provavelmente ele já está instalado em seu sistema. Caso contrário, você pode baixá-lo em *https://www.python.org/*.*5*

Em seguida, crie um diretório de trabalho para seu código e conjuntos de dados do Aprendizado de Máquina. Abra um terminal e digite os seguintes comandos (após o prompts $):

$ export ML\_PATH="$HOME/ml" # Mude o caminho se preferir $ mkdir -p $ML\_PATH

Você precisará de vários módulos Python: Jupyter, NumPy, Pandas, Matplotlib e Scikit-Learn. Se você já tem o Jupyter rodando com todos esses módulos instalados, pode pular com segurança para “Baixe os dados” na página 45. Se você ainda não tem, há várias

5 A versão mais recente do Python 3 é recomendada. O Python 2.7+ também funcionará bem, mas está obsoleto. Se usar o Python 2, você deve adicionar from\_future\_import division, print\_function, unicode\_literals no início do seu código.

**42 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

maneiras de instalá-los (e suas dependências). Você pode utilizar o sistema de empacota mento do seu sistema (por exemplo, apt-get no Ubuntu, ou MacPorts ou HomeBrew no macOS), instale uma distribuição científica do Python, como o Anaconda, e utilize seu sistema de empacotamento, ou utilize o sistema de empacotamento do Python, pip, que está incluído por padrão com os instaladores binários Python (desde a versão Python 2.7.9).*6* Você pode verificar se o pip está instalado digitando o seguinte comando:

$ pip3 --version

pip 9.0.1 from [...]/lib/python3.5/site-packages (python 3.5)

Você deve se certificar de que tenha uma versão recente do pip instalada, pelo menos > 1.4 para suportar a instalação do módulo binário (também conhecida como wheels). Para atualizar o módulo do pip, digite:*7*

$ pip3 install --upgrade pip

Collecting pip

[...]

Successfully installed pip-9.0.1

**Criando um Ambiente Isolado**

Se você quiser trabalhar em um ambiente isolado (o que é fortemente recomendado para que se possa trabalhar em projetos diferentes sem ter versões confitantes de biblioteca), instale o virtualenv executando o seguinte comando pip:

$ pip3 install --user --upgrade virtualenv

Collecting virtualenv

[...]

Successfully installed virtualenv

Agora você pode criar um ambiente Python isolado digitando:

$ cd $ML\_PATH

$ virtualenv env

Using base prefix '[...]'

New python executable in [...]/ml/env/bin/python3.5

Also creating executable in [...]/ml/env/bin/python

Installing setuptools, pip, wheel...done.

Agora, sempre que quiser ativar esse ambiente, basta abrir um terminal e digitar:

$ cd $ML\_PATH

$ source env/bin/activate

6 Mostraremos a seguir os passos para a instalação usando pip em uma *bash shell* no Linux ou no macOS. Talvez seja preciso adaptar estes comandos ao seu próprio sistema. Para o Windows, recomendamos a instalação do Anaconda.

7 Talvez seja necessário ter direitos de administrador para usar este comando; se for o caso, tente prefixá-lo com *sudo.*

**Obtenha os Dados | 43**

Qualquer pacote que você instalar utilizando pip será instalado neste ambiente isolado enquanto o ambiente estiver ativo, e o Python somente terá acesso a esses pacotes (se você também quiser acesso aos pacotes do sistema no site, deverá criar o ambiente uti lizando a opção --system-site-packages do virtualenv). Confra a documentação do virtualenv para obter mais informações.

Agora você pode instalar todos os módulos necessários e suas dependências utilizando este simples comando pip (se você não estiver utilizando um virtualenv, precisará de direitos de administrador ou adicionar a opção --user):

$ pip3 install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn Collecting jupyter

Downloading jupyter-1.0.0-py2.py3-none-any.whl

Collecting matplotlib

[...]

Para verificar sua instalação, tente importar todos os módulos assim: $ python3 -c "import jupyter, matplotlib, numpy, pandas, scipy, sklearn" Não deve haver nenhuma saída e nenhum erro. Agora você pode iniciar o Jupyter digitando:

$ jupyter notebook

[I 15:24 NotebookApp] Serving notebooks from local directory: [...]/ml [I 15:24 NotebookApp] 0 active kernels

[I 15:24 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/ [I 15:24 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

Um servidor Jupyter está sendo executado agora pela porta 8888 em seu terminal. Você pode visitar este servidor abrindo seu navegador em *http://localhost: 8888/* (isso geralmente acontece automaticamente quando o servidor é iniciado). Você deve ver seu diretório de trabalho vazio (contendo apenas o diretório *env* se você seguiu as instruções anteriores do virtualenv).

Agora, crie um novo notebook Python clicando no botão New e selecionando a versão adequada do Python*8* (veja a Figura 2-3).

Isto faz três coisas: primeiro, cria um novo arquivo notebook chamado *Untitled.ipynb* em seu espaço de trabalho; segundo, inicia um *Jupyter Python kernel* para rodar este *notebook*; e, terceiro, abre este notebook em uma nova guia. Você deve começar clicando em Untitled e digitando o novo nome, renomeando este notebook para “Housing” (isso mudará o arquivo automaticamente para *Housing.ipynb*).

8 Note que o Jupyter pode lidar com várias versões do Python, e até muitas outras linguagens, como R ou Octave.

**44 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

**Files Running Clusters** 

Select items to perform actions on them.

**env**

*Figura 2-3. Seu espaço de trabalho no Jupyter*

**Logout**

**Upload New**

**Text File**

**Folder**

**Terminal**

**Notebooks**

**Octave**

**Python 2**

**Python 3**

Um notebook contém uma lista de células. Cada célula pode conter código executável ou texto formatado. No momento, o notebook contém apenas uma célula de código vazia, rotulada “In [1]:”. Tente digitar **print(“Hello world!”)** na célula, e clique no botão Reproduzir (veja a Figura 2-4) ou pressione Shift-Enter. Isso envia a célula atual para o *kernel* Python deste notebook, que o executa e retorna a saída. O resultado é exibido abaixo da célula, e, já que chegamos ao final do notebook, uma nova célula é automaticamente criada. Confira o User Interface Tour no menu de Ajuda do Jupyter para aprender o básico.

**Housing**

*Figura 2-4. Notebook Python Hello world*

**Baixe os Dados**

Em ambientes típicos, seus dados estariam disponíveis em um banco de dados relacional (ou algum outro armazenamento comum de dados) e espalhados por várias tabelas/ documentos/arquivos. Para acessá-lo, primeiro você precisaria obter suas credenciais e autorizações*9* de acesso e familiarizar-se com o esquema de dados. Neste projeto, no en tanto, as coisas são muito mais simples: você só vai baixar um único arquivo compactado,

9 Você também pode precisar verificar restrições legais, como campos privados que nunca devem ser copiados para armazenamentos inseguros de dados.

**Obtenha os Dados | 45**

*housing.tgz*, que contém um arquivo com valores separados por vírgulas (CSV) chamado *housing.csv* com todos os dados.

Você poderia usar o navegador da Web para baixá-lo e executar o tar xzf housing.tgz para descompactá-lo e extrair o arquivo CSV, mas é preferível criar uma pequena função para fazer isso. Ela será útil, principalmente, se os dados mudarem regularmente, pois permite que você escreva um pequeno script que poderá ser executado sempre que pre cisar para buscar os dados mais recentes (ou poderá configurar um trabalho agendado para fazer isso automaticamente em intervalos regulares). Também será útil automatizar o processo de busca caso você precise instalar o conjunto de dados em várias máquinas.

Esta é a função para buscar os dados:*10*

import os

import os

import tarfile

import tarfile

from six.moves import urllib

from six.moves import urllib

DOWNLOAD\_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/" DOWNLOAD\_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"

HOUSING\_PATH = os.path.join("datasets", "housing")

HOUSING\_PATH = os.path.join("datasets", "housing")

HOUSING\_URL = DOWNLOAD\_ROOT + "datasets/housing/housing.tgz"

HOUSING\_URL = DOWNLOAD\_ROOT + "datasets/housing/housing.tgz"

def fetch\_housing\_data(housing\_url=HOUSING\_URL, housing\_path=HOUSING\_PATH): def fetch\_housing\_data(housing\_url=HOUSING\_URL, housing\_path=HOUSING\_PATH):

if not os.path.isdir(housing\_path):

if not os.path.isdir(housing\_path):

os.makedirs(housing\_path)

os.makedirs(housing\_path)

tgz\_path = os.path.join(housing\_path, "housing.tgz")

tgz\_path = os.path.join(housing\_path, "housing.tgz")

urllib.request.urlretrieve(housing\_url, tgz\_path)

urllib.request.urlretrieve(housing\_url, tgz\_path)

housing\_tgz = tarfile.open(tgz\_path)

housing\_tgz = tarfile.open(tgz\_path)

housing\_tgz.extractall(path=housing\_path)

housing\_tgz.extractall(path=housing\_path)

housing\_tgz.close()

housing\_tgz.close()

Agora, quando chamamos fetch\_housing\_data(), ele cria um diretório *datasets/housing* no seu espaço de trabalho, baixa o arquivo *housing.tgz* e extrai o *housing.csv* neste diretório.

Agora, vamos carregar os dados com o Pandas. Mais uma vez você deve escrever uma pequena função para carregá-los:

import pandas as pd

def load\_housing\_data(housing\_path=HOUSING\_PATH):

csv\_path = os.path.join(housing\_path, "housing.csv")

return pd.read\_csv(csv\_path)

Esta função retorna um objeto *DataFrame Pandas* contendo todos os dados.

10 Em um projeto real, você salvaria este código em um arquivo Python, mas, por enquanto, você pode apenas escrevê-lo no seu notebook do Jupyter.

**46 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

**Uma Rápida Olhada na Estrutura dos Dados**

Utilizando o método head() do DataFrame, vejamos as cinco linhas superiores (veja Figura 2-5).

*Figura 2-5. As cinco linhas superiores no conjunto de dados*

Cada linha representa um bairro. Existem 10 atributos (você pode ver os primeiros seis na captura de tela): longitude, latitude, housing\_median\_age, total\_rooms, total\_bed rooms, population, households, median\_income, median\_house\_value e ocean\_proximity.

O método info() é útil para a obtenção de uma rápida descrição dos dados, em especial o número total de linhas, o tipo de cada atributo e o número de valores não nulos (veja a Figura 2-6).

*Figura 2-6. Informações das Casas*

Existem 20.640 instâncias no conjunto de dados, o que significa que é muito pequeno para os padrões de Aprendizado de Máquina, mas é perfeito para começar. Repare que o atributo total\_bedrooms tem apenas 20.433 valores não nulos, significando que 207 bairros não possuem esta característica. Cuidaremos disso mais tarde.

**Obtenha os Dados | 47**

Todos os atributos são numéricos, exceto o campo ocean\_proximity. Seu tipo é object, então ele poderia conter qualquer tipo de objeto Python, mas, como você carregou esses dados de um arquivo CSV, você sabe que ele deve ser um atributo de texto. Quando você olhou para as cinco primeiras linhas, provavelmente percebeu que os valores na coluna ocean\_proximity eram repetitivos, o que significa que é provavelmente um atributo categórico. Você pode descobrir quais categorias existem e quantos bairros pertencem a cada categoria utilizando o método value\_counts():

>>> housing["ocean\_proximity"].value\_counts()

<1H OCEAN 9136

INLAND 6551

NEAR OCEAN 2658

NEAR BAY 2290

ISLAND 5

Name: ocean\_proximity, dtype: int64

Vamos olhar para outros campos. O método describe() mostra um resumo dos atri butos numéricos (Figura 2-7).



*Figura 2-7. Resumo de cada atributo numérico*

As linhas count, mean, min e max são autoexplicativas. Observe que os valores nulos são ignorados (então, por exemplo, count do total\_bedrooms é 20.433, não 20.640). A linha std mostra o *desvio padrão*, que mede a dispersão dos valores.*11*As linhas 25%, 50% e 75% mostram os *percentis* correspondentes: um percentil indica o valor abaixo no qual uma dada porcentagem cai em um grupo de observações. Por exemplo, 25% dos bairros tem uma housing\_median\_age menor que 18, enquanto 50% é menor que 29, e 75% é menor que 37. Estes são frequentemente chamados de 25º percentil (ou 1º*quartil*), a média e o 75º percentil (ou 3º quartil).

11 O desvio padrão é geralmente denotado σ (da letra grega sigma) e é a raiz quadrada da *variância*, que é a média do desvio quadrado da média. Quando uma característica tem uma *distribuição normal* em forma de sino (também chamada de *distribuição Gaussiana*), que é muito comum, aplica-se a regra “68-95-99,7”: cerca de 68% dos valores se enquadram em 1σ da média, 95% dentro de 2σ e 99,7% dentro de 3σ.

**48 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

Outro método rápido de perceber o tipo de dados com o qual você está lidando é traçar um histograma para cada atributo numérico. Um histograma mostra o número de ins tâncias (no eixo vertical) que possuem um determinado intervalo de valores (no eixo horizontal). Você pode traçar esse único atributo por vez ou pode chamar o método hist() em todo o conjunto de dados e traçar um histograma para cada atributo numérico (veja a Figura 2-8). Por exemplo, você pode ver que pouco mais de 800 bairros possuem um median\_house\_value equivalente a mais ou menos US$ 100 mil.

%matplotlib inline # somente no notebookJupyter

import matplotlib.pyplot as plt

housing.hist(bins=50, figsize=(20,15))

plt.show()

O método hist() depende do Matplotlib, que por sua vez depen

de de um backend gráfco especifcado pelo usuário para fgurar em

sua tela. Então, antes que você consiga plotar qualquer coisa, é pre

ciso especifcar qual backend o Matplotlib deve usar. A opção mais

simples é usar o comando mágico do Jupyter %matplotlib inline.

Isso leva o Jupyter a confgurar o Matplotlib para que ele utilize seu

próprio backend. As plotagens são então processadas dentro do

próprio notebook. Observe que é opcional em um notebook do

Jupyter chamar o show(), pois ele exibirá gráfcos automaticamente

sempre que uma célula for executada.

households housing\_median\_age latitude 

longitude median\_house\_value median\_income

population total\_bedrooms total\_rooms

*Figura 2-8. Um histograma para cada atributo numérico*

**Obtenha os Dados | 49**

Preste atenção em alguns pontos destes histogramas:

1. Primeiro, o atributo da renda média não parece estar expresso em dólares ame ricanos (USD). Depois de verificar com a equipe de coleta de dados, você é infor mado que os dados foram dimensionados e limitados em 15 (na verdade 15,0001) para a média dos maiores rendimentos, e em 0,5 (na verdade 0,4999) para a média dos rendimentos mais baixos. É comum trabalhar com atributos pré-processados no Aprendizado de Máquina e isto não é necessariamente um problema, mas você deve tentar entender como os dados foram calculados.

2. A idade média e o valor médio da casa também foram limitados. Este último pode ser um problema sério, pois é seu atributo alvo (seus rótulos). Os algoritmos de Aprendizado de Máquina podem aprender que os preços nunca ultrapassam esse limite. Você precisa verificar com a equipe do seu cliente (a equipe que usará a saída do seu sistema) para ver se isso é ou não um problema. Se eles disserem que precisam de previsões precisas mesmo acima de US$ 500 mil, então você terá duas opções:

a. Coletar rótulos adequados para os bairros cujos rótulos foram limitados.

b. Remover esses bairros do conjunto de treinamento (e também do conjunto de testes, já que seu sistema não deve ser responsabilizado por prever valores além de US$ 500 mil).

3. Esses atributos têm escalas muito diferentes. Discutiremos isso mais adiante neste capítulo, quando explorarmos o escalonamento das características.

4. Finalmente, muitos histogramas têm um *rastro alongado*: eles se estendem muito mais à direita da média do que à esquerda. Isso pode dificultar a detecção de pa drões em alguns algoritmos do Aprendizado de Máquina. Vamos tentar transfor mar esses atributos mais tarde para conseguir mais distribuições na forma de sino.

Espero que você tenha agora uma melhor compreensão do tipo de dados com os quais está lidando.

Espere! Antes de analisar ainda mais os dados, você precisa criar um

conjunto de teste, colocá-lo de lado e nunca checá-lo.

**50 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

**Crie um Conjunto de Testes**

Pode parecer estranho colocar de lado, voluntariamente, uma parte dos dados nesta fase. Afinal, você só deu uma rápida olhada nos dados e certamente precisa aprender muito mais a respeito antes de decidir quais algoritmos usar, certo? Isso é verdade, mas o seu cérebro é um incrível sistema de detecção de padrões, o que significa que é altamente propenso ao *sobreajuste*: se você olhar para o conjunto de teste, pode tropeçar em algum padrão aparentemente interessante nos dados de teste que o levará a selecionar um tipo particular de modelo do Aprendizado de Máquina. Quando você estima o erro de generalização utilizando o conjunto de teste, sua estimativa será muito otimista e você lançará um sistema que não funcionará tão bem quanto o esperado. Isso é chamado de *data snooping bias*.

Criar um conjunto de testes é, teoricamente, bastante simples: basta escolher aleatoria mente algumas instâncias, geralmente 20% do conjunto de dados, e colocá-las de lado:

import numpy as np

def split\_train\_test(data, test\_ratio):

shuffled\_indices = np.random.permutation(len(data))

test\_set\_size = int(len(data) \* test\_ratio)

test\_indices = shuffled\_indices[:test\_set\_size]

train\_indices = shuffled\_indices[test\_set\_size:]

return data.iloc[train\_indices], data.iloc[test\_indices]

Você pode usar essa função assim:

>>> train\_set, test\_set = split\_train\_test(housing, 0.2)

>>> print(len(train\_set), "train +", len(test\_set), "test")

16512 train + 4128 test

Bem, isso funciona, mas não é perfeito: ao executar o programa novamente, será gerado um conjunto diferente de testes! Ao longo do tempo, você (ou seus algoritmos do Apren dizado de Máquina) verá todo o conjunto de dados, o que deve ser evitado.

Uma solução seria salvar o conjunto de testes na primeira execução e depois carregá-lo em execuções subsequentes. Outra opção é definir a semente do gerador de números alea tórios (por exemplo, np.random.seed(42))*12* antes de chamar a np.random.permutation(), de modo que ele sempre gere os mesmos índices embaralhados.

Mas ambas as soluções serão interrompidas na próxima vez que você buscar um conjunto de dados atualizado. Uma solução comum é utilizar o identificador de cada instância para decidir se ela deve ou não ir no conjunto de teste (supondo que as instâncias tenham um identificador único e imutável). Por exemplo, você pode calcular um hash do identi

12 Você verá com frequência pessoas colocarem a *random seed* em 42. Esse número não possui propriedade especial alguma além de ser A Resposta para a Vida, para o Universo e Tudo Mais.

**Obtenha os Dados | 51**

ficador de cada instância, manter apenas o último byte do hash e colocar a instância no conjunto de teste se esse valor for menor ou igual a 51 (~20% de 256). Isso garante que o conjunto de teste permanecerá consistente em várias execuções, mesmo ao atualizar o conjunto de dados. O novo conjunto de testes conterá 20% das novas instâncias, mas não conterá nenhuma instância que já estivesse no conjunto de treinamento. Veja uma possível implementação:

import hashlib

def test\_set\_check(identifier, test\_ratio, hash):

return hash(np.int64(identifier)).digest()[-1] < 256 \* test\_ratio

def split\_train\_test\_by\_id(data, test\_ratio, id\_column, hash=hashlib.md5): ids = data[id\_column]

in\_test\_set = ids.apply(lambda id\_: test\_set\_check(id\_, test\_ratio, hash)) return data.loc[~in\_test\_set], data.loc[in\_test\_set]

Infelizmente, o conjunto de dados do setor imobiliário não possui uma coluna de iden tificação. A solução mais simples é utilizar o índice da linha como ID:

housing\_with\_id = housing.reset\_index() # adiciona uma coluna ‘index` train\_set, test\_set = split\_train\_test\_by\_id(housing\_with\_id, 0.2, "index")

Se você utilizar o índice de linha como um identificador exclusivo, precisa se certificar de que novos dados sejam anexados ao final do conjunto de dados e que nenhuma linha seja excluída. Se isso não for possível, tente utilizar características mais estáveis para criar um identificador exclusivo. Por exemplo, a latitude e a longitude de um bairro serão certamente estáveis por alguns milhões de anos, então você pode combiná-las em um ID dessa forma:*13*

housing\_with\_id["id"] = housing["longitude"] \* 1000 + housing["latitude"] train\_set, test\_set = split\_train\_test\_by\_id(housing\_with\_id, 0.2, "id")

O Scikit-Learn fornece algumas funções para dividir conjuntos de dados em vários subconjuntos de diversas maneiras. A função mais simples é train\_test\_split, que faz praticamente o mesmo que a função split\_train\_test definida anteriormente, mas com alguns recursos adicionais. Primeiro, temos um parâmetro random\_state que permite que você defina a semente do gerador de números aleatórios como explicado anteriormente e, em segundo lugar, você pode passar múltiplos conjuntos de dados com um número idêntico de linhas, e ele os dividirá nos mesmos índices (isso é muito útil se, por exemplo, você tiver um *DataFrame* separado para os rótulos):

from sklearn.model\_selection import train\_test\_split

train\_set, test\_set = train\_test\_split(housing, test\_size=0.2, random\_state=42)

13 A informação de localização é realmente bastante grosseira e, como resultado, muitos bairros terão exatamente a mesma identificação, acabando no mesmo conjunto (*test* ou *train*). Infelizmente, isso introduz algum viés de amostragem.

**52 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

Até agora consideramos somente métodos de amostragem puramente aleatórios. Isso geralmente é bom se o seu conjunto de dados for suficientemente grande (especialmente em relação ao número de atributos), mas, se não for, corre o risco de apresentar um viés significativo de amostragem. Quando uma empresa de pesquisa decide ligar para mil pessoas e lhes fazer algumas perguntas, eles não escolhem aleatoriamente mil pessoas em uma lista telefônica. Eles tentam garantir que essas mil pessoas representem toda a população. Por exemplo, a população dos EUA é composta por 51,3% de pessoas do sexo feminino e 48,7% do sexo masculino, de modo que uma pesquisa bem conduzida tentaria manter essa proporção na amostragem: 513 mulheres e 487 homens. Isso é chamado de *amostragem estratificada*: a população é dividida em subgrupos homogêneos, chamados de *estratos*, e o número certo de instâncias de cada estrato é amostrado para garantir que o conjunto de testes seja representativo da população em geral. Se eles utilizassem amostragem puramente aleatória, haveria cerca de 12% de chance de amostrar um con junto de teste distorcido, tanto com menos de 49% feminino quanto com mais de 54%. De qualquer forma, os resultados da pesquisa seriam significativamente tendenciosos.

Suponha que você tenha conversado com especialistas que lhe disseram que a renda média é um atributo muito importante para estimar os preços médios. Você quer garantir que o conjunto de testes seja representativo das várias categorias de rendimentos em todo o conjunto de dados. Uma vez que a renda média é um atributo numérico contínuo, primeiro você precisa criar um atributo na categoria da renda. Vejamos mais de perto o histograma da renda média (de volta à Figura 2-8): a maioria dos valores médios da renda está agrupada em torno de US$ 20 mil–US$ 50 mil mas alguns rendimentos mé dios ultrapassam os US$ 60 mil. É importante ter um número suficiente de instâncias para cada estrato em seu conjunto de dados, ou então a estimativa da importância do estrato poderá ser tendenciosa. Isso significa que você não deve ter muitos estratos, e cada estrato deve ser grande o suficiente. O código a seguir cria um atributo da categoria da renda dividindo a renda média por 1,5 (para limitar o número de categorias da renda) e arredondando com a utilização do ceil (para ter categorias discretas) e, em seguida, mesclando todas as categorias maiores que 5, na categoria 5:

housing["income\_cat"] = np.ceil(housing["median\_income"] / 1.5)

housing["income\_cat"].where(housing["income\_cat"] < 5, 5.0, inplace=True) Essas categorias de renda estão representadas na (Figura 2–9):

**Obtenha os Dados | 53**

****

*Figura 2-9. Histograma das categorias de renda*

Agora você está pronto para fazer uma amostragem estratificada com base na categoria da renda. Para isso você pode utilizar a classe StratifiedShuffleSplit do Scikit-Learn:

from sklearn.model\_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42) for train\_index, test\_index in split.split(housing, housing["income\_cat"]): strat\_train\_set = housing.loc[train\_index]

strat\_test\_set = housing.loc[test\_index]

Vamos ver se isso funcionou como esperado. Você pode começar pela análise das pro porções da categoria de renda no conjunto de testes:

>>> strat\_test\_set["income\_cat"].value\_counts() / len(strat\_test\_set) 3.0 0.350533

2.0 0.318798

4.0 0.176357

5.0 0.114583

1.0 0.039729

Name: income\_cat, dtype: float64

Com um código similar, você pode medir as proporções da categoria de renda no con junto completo de dados. A Figura 2-10 compara as proporções da categoria de renda no conjunto geral de dados, no conjunto de teste gerado com a amostragem estratificada e em um conjunto de testes gerado a partir da amostragem puramente aleatória. Como você pode ver, o conjunto de testes gerado com a utilização da amostragem estratifica da tem proporções da categoria de renda quase idênticas às do conjunto completo de dados, enquanto o conjunto de testes gerado com amostragem puramente aleatória é bastante distorcido.

**54 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

****

*Figura 2-10. Comparação de viés de amostragem estratificada versus amostragem aleatória*

Agora, você deve remover o atributo income\_cat para que os dados voltem ao seu estado original:

for set\_ in (strat\_train\_set, strat\_test\_set):

set\_.drop("income\_cat", axis=1, inplace=True)

Ficamos um tempo na geração de conjuntos de testes por uma boa razão: esta parte crítica é muitas vezes negligenciada em um projeto de Aprendizado de Máquina. Além disso, muitas dessas ideias serão úteis mais tarde quando discutirmos a validação cruzada. Agora, é hora de avançar para o próximo estágio: explorar os dados.

**Descubra e Visualize os Dados para Obter Informações**

Até aqui você deu apenas uma rápida olhada para ter uma compreensão geral do tipo de dados que está manipulando. Agora, o objetivo é aprofundar-se um pouco mais.

Primeiro, certifique-se de colocar o teste de lado e apenas explorar o conjunto de treina mento. Além disso, se o conjunto de treinamento for muito grande, talvez seja melhor experimentar um conjunto de exploração para fazer manipulações fácil e rapidamente. No nosso caso, o conjunto é muito pequeno, então você pode trabalhar diretamente no conjunto completo. Criaremos uma cópia para que você possa treinar com ela sem prejudicar o conjunto de treinamento:

housing = strat\_train\_set.copy()

**Visualizando Dados Geográficos**

Como existem informações geográficas (latitude e longitude), é uma boa ideia criar um diagrama de dispersão para visualizar os dados de todos os bairros (Figura 2-11):

housing.plot(kind="scatter", x="longitude", y="latitude")

**Descubra e Visualize os Dados para Obter Informações | 55**

e 

d

u

t

i

t

a

L

Longitude

*Figura 2-11. Um diagrama de dispersão geográfica dos dados*

Isso se parece com a Califórnia, mas, além disso, é difícil ver qualquer padrão específico. Definir a opção alpha em 0,1 facilita a visualização dos locais onde existe uma alta densidade de pontos de dados (Figura 2-12):

housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)

e 

d

u

t

i

t

a

L

Longitude

*Figura 2-12. Uma melhor visualização destacando áreas de alta densidade*

Agora já está melhor: é possível ver claramente as áreas de alta densidade, especificamente a Área da Baía de Los Angeles e San Diego, além de uma longa linha de alta densidade no Vale Central, principalmente ao redor de Sacramento e Fresno.

**56 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

No geral, nosso cérebro é muito bom em detectar padrões em imagens, mas talvez seja necessário brincar com parâmetros de visualização para que os padrões se destaquem.

Agora, vejamos os preços do setor imobiliário (Figura 2-13). O raio de cada círculo re presenta a população do bairro (opção s) e a cor representa o preço (opção c). Usaremos um mapa de cores pré-definido (opção cmap) chamado jet, que varia do azul (valores baixos) para o vermelho (preços altos):*14*

housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4, s=housing["population"]/100, label="population", figsize=(10,7), c="median\_house\_value", cmap=plt.get\_cmap("jet"), colorbar=True, )

plt.legend()

Population

e

u

l

a

v

\_

e 

e

s

d

u

u

o

t

i

h

t

\_

a

L

n

a

i

d

e

m

Longitude

*Figura 2-13. Preços das casas na Califórnia*

Esta imagem informa que os preços do setor imobiliário estão muito relacionados à localização (por exemplo, perto do oceano) e à densidade populacional, como prova velmente você já sabia. Será útil utilizar um algoritmo de agrupamento para detectar os grupos principais e adicionar novas características que medem a proximidade com os centros de agrupamento. O atributo de proximidade do oceano também pode ser útil, embora na costa norte da Califórnia os preços não sejam muito altos, então essa não é uma regra simples.

14 Se você estiver lendo em escala de cinza, pegue uma caneta vermelha e rabisque na maior parte do litoral da Área da Baía até San Diego (como esperado). Você também pode acrescentar um trecho amarelo ao redor de Sacramento.

**Descubra e Visualize os Dados para Obter Informações | 57**

**Buscando Correlações**

Uma vez que o conjunto de dados não é muito grande, você pode calcular facilmente o *coeficiente de correlação padrão* (também chamado *r de Pearson*) entre cada par de atributos utilizando o método corr():

corr\_matrix = housing.corr()

Agora, vejamos o quanto cada atributo se correlaciona com o valor médio da habitação:

>>> corr\_matrix["median\_house\_value"].sort\_values(ascending=False) median\_house\_value 1.000000

median\_income 0.687170

total\_rooms 0.135231

housing\_median\_age 0.114220

households 0.064702

total\_bedrooms 0.047865

population -0.026699

longitude -0.047279

latitude -0.142826

Name: median\_house\_value, dtype: float64

O coeficiente de correlação varia de -1 a 1. Quando está próximo de 1, significa que existe uma forte correlação positiva; por exemplo, o valor médio da habitação tende a aumentar quando a renda média aumenta. Quando o coeficiente está próximo de -1, significa que existe uma forte correlação negativa; é possível ver uma pequena correlação negativa entre a latitude e o valor médio da habitação (ou seja, os preços tendem a diminuir quando você vai para o norte). Finalmente, coeficientes próximos de zero significam que não há correlação linear. A Figura 2-14 mostra várias plotagens juntamente com o coeficiente de correlação entre seus eixos horizontal e vertical.



*Figura 2-14. Coeficiente de correlação padrão de vários conjuntos de dados (fonte: Wikipedia; imagem de domínio público)*

**58 | Capítulo 2: Projeto de Aprendizado de Máquina de Ponta a Ponta**

O coefciente de correlação apenas mede correlações lineares (“se *x*

sobe, então *y* geralmente sobe/desce”). Ele pode perder completa

mente as relações não lineares (por exemplo, “se *x* é próximo a zero,

então, *y,* geralmente, sobe”). Observe como todas as plotagens da li

nha inferior têm um coefciente de correlação igual a zero, apesar do

fato de seus eixos claramente não serem independentes: são exem

plos de relações não lineares. Além disso, a segunda linha mostra

exemplos em que o coefciente de correlação é igual a 1 ou -1; obser

ve que isso não tem nada a ver com a inclinação. Por exemplo, sua

altura em polegadas tem um coefciente de correlação de 1 com sua

altura em pés ou em nanômetros.

Outra maneira de verificar a correlação entre atributos é utilizar a função scatter\_matrix, do Pandas, que plota cada atributo numérico em relação a qualquer outro atributo numérico. Uma vez que existem 11 atributos numéricos, você obteria 112 = 121 plotagens, o que não caberia em uma página, então focaremos apenas em alguns atributos promissores que parecem mais correlacionados com o valor médio do setor imobiliário (Figura 2-15):

from pandas.tools.plotting import scatter\_matrix

attributes = ["median\_house\_value", "median\_income", "total\_rooms", "housing\_median\_age"]

scatter\_matrix(housing[attributes], figsize=(12, 8))

e 

u

l

a

v

\_

e

s

u

o

h

\_

n

a

i

d

e

m

e

m

o

c

n

i

\_

n

a

i

d

e

m

s

m

o

o

r

\_

l

a

t

o

t

e

g

a

\_

n

a

i

d

e

m

\_

g

n

i

s

u

o

h

median\_house\_value

median\_incometotal\_rooms housing\_median\_age

*Figura 2-15. Matriz de dispersão*

**Descubra e Visualize os Dados para Obter Informações | 59**