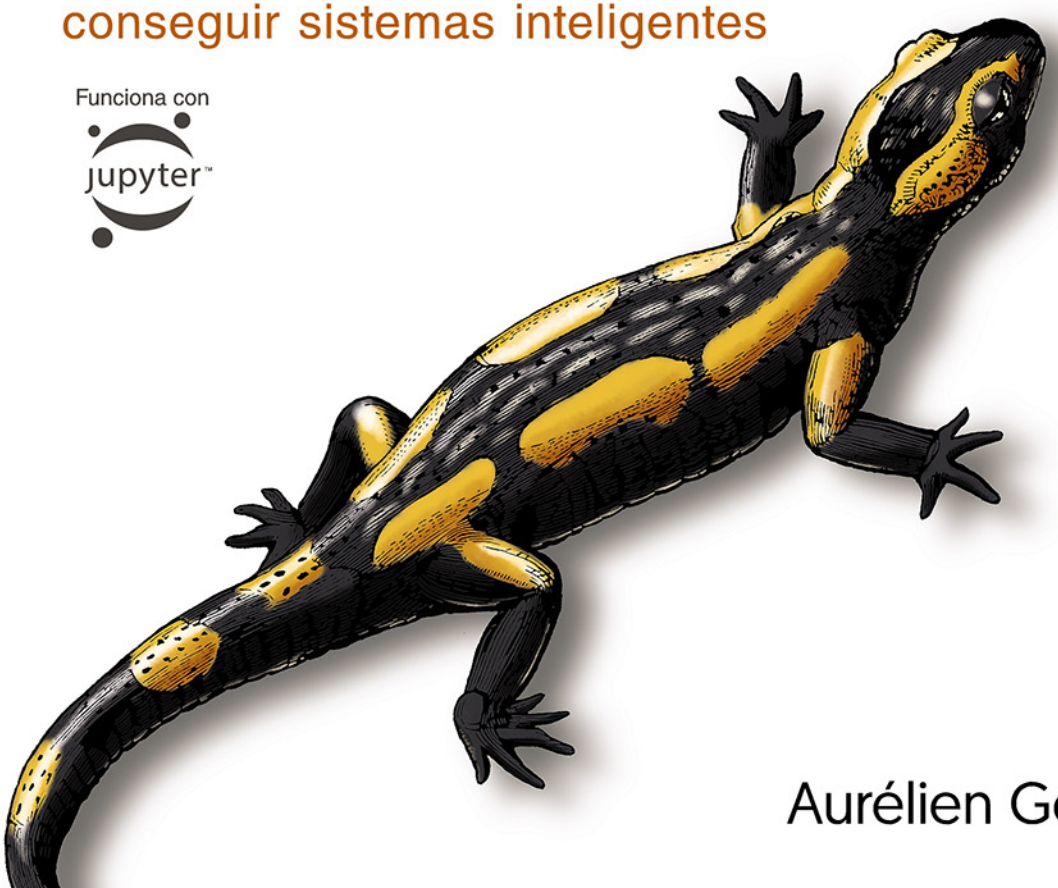
O'REILLY® ANAYA MULTIMEDIA

TERCERA EDICIÓN

Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow

Conceptos, herramientas y técnicas para conseguir sistemas inteligentes

Funciona con jupyter™



Aurélien Géron

TERCERA EDICIÓN

Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow

Conceptos, herramientas y técnicas para conseguir sistemas inteligentes

Aurélien Géron



ANAYA MULTIMEDIA

TÍTULO ESPECIAL

Título de la obra original:
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow. Third Edition

Traductor:
Beatriz Pineda González

Responsable editorial:
Víctor Manuel Ruiz Calderón

Adaptación de cubierta:
Celia Antón Santos

Primera edición digital: 2023

Todos los nombres propios de programas, sistemas operativos, equipos hardware, etc. que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Reservados todos los derechos. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multas, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujeren, plagiaren, distribuyeren o comunicaren públicamente, en todo o en parte, una obra literaria, artística o científica, o su transformación, interpretación o ejecución artística fijada en cualquier tipo de soporte o comunicada a través de cualquier medio, sin la preceptiva autorización.

Authorized translation from English language edition published by O'Reilly Media, Inc.
Copyright © 2023 Aurélien Géron. All rights reserved.

Edición española:

© EDICIONES ANAYA MULTIMEDIA (GRUPO ANAYA, S.A.), 2023
Valentín Beato, 21. 28037 Madrid
ISBN: 978-84-415-4828-2
Edición electrónica sobre la 3.ª edición impresa

Agradecimientos

Nunca, ni en mis mejores sueños, imaginé que la primera y la segunda edición de este libro tendrían un público tan grande. Recibí muchos mensajes de lectores, muchos hacían preguntas, algunos indicaban con amabilidad la presencia de erratas y la mayoría enviaban palabras de ánimo. No puedo expresar lo agradecido que estoy a todos esos lectores por todo el apoyo. ¡Muchísimas gracias a todos! Por favor, no dudes en informar de cualquier problema en GitHub (`https://homl.info/issues3`) si encuentras errores en los ejemplos de código (o en hacer preguntas sin más) o de señalar erratas (`https://homl.info/errata3`) si encuentras errores en el texto. Algunos lectores compartieron su experiencia sobre cómo este libro les ayudó a conseguir su primer trabajo o a resolver un problema concreto en el que estaban trabajando. Ese tipo de comentarios son muy motivadores para mí. Si este libro te resulta útil, me encantaría que compartieses tu historia conmigo, ya sea en privado (por ejemplo, a través de LinkedIn (`https://linkedin.com/in/aurelien-geron`)) o en público (por ejemplo, con un tuit en `@aureliengeron` o con una reseña en Amazon (`https://homl.info/amazon3`)).

Muchísimas gracias también a todas las personas maravillosas que me ofrecieron su tiempo y su experiencia para revisar esta tercera edición, corregir errores y hacer innumerables sugerencias. Esta edición es mucho mejor gracias a ellos: Olzhas Akpambetov, George Bonner, François Chollet, Siddha Ganju, Sam Goodman, Matt Harrison, Sasha Sobran, Lewis Tunstall, Leandro von Werra y mi querido hermano Sylvain. ¡Sois increíbles!

También estoy muy agradecido a las muchas personas que me han apoyado en este camino respondiendo a mis preguntas, sugiriendo mejoras y contribuyendo al código en GitHub: en especial, Yannick Assogba, Ian Beauregard, Ulf Bissbort, Rick Chao, Peretz Cohen, Kyle Gallatin, Hannes Hapke, Victor Khaustov, Soonson Kwon, Eric Lebigot, Jason Mayes, Laurence Moroney, Sara Robinson, Joaquín Ruales y Yuefeng Zhou.

Este libro no existiría sin el fantástico personal de O'Reilly, en especial Nicole Taché, que me ofreció unas opiniones muy perspicaces y siempre fue amable, alentadora y servicial: no podría pedir una editora mejor. Muchísimas gracias también a Michele Cronin, que me animó en los capítulos finales y consiguió hacerme cruzar la línea de meta. Gracias a todo el equipo de producción, sobre todo a Elizabeth Kelly y Kristen Brown. Gracias también a Kim Cofer por la corrección exhaustiva y a Johnny O'Toole, que gestionó la relación con Amazon y respondió a muchas de mis preguntas. Gracias a Kate Dullea por mejorar muchísimo mis ilustraciones. Gracias a Marie Beaugureau, Ben Lorica, Mike Loukides y Laurel Ruma por creer en este proyecto y ayudarme a definir su alcance. Gracias a Matt Hacker y a todo el equipo de Atlas por responder a mis preguntas técnicas acerca del formato, AsciiDoc, MathML y LaTeX, y gracias también a Nick Adams, Rebecca Demarest, Rachel Head, Judith McConville, Helen Monroe, Karen Montgomery, Rachel Roumeliotis y a todas las personas de O'Reilly que han contribuido a la publicación de este libro.

Nunca olvidaré a las personas maravillosas que me ayudaron con la primera y la segunda edición de este libro: amigos, colegas, expertos, incluidos muchos miembros del equipo de TensorFlow. La lista es larga: Olzhas Akpambetov, Karmel Allison, Martin Andrews, David

Andrzejewski, Paige Bailey, Lukas Biewald, Eugene Brevdo, William Chargin, François Chollet, Clément Courbet, Robert Crowe, Mark Daoust, Daniel "Wolff " Dobson, Julien Dubois, Mathias Kende, Daniel Kitachewsky, Nick Felt, Bruce Fontaine, Justin Francis, Goldie Gadde, Irene Giannoumis, Ingrid von Glehn, Vincent Guilbeau, Sandeep Gupta, Priya Gupta, Kevin Haas, Eddy Hung, Konstantinos Katsiapis, Viacheslav Kovalevskyi, Jon Krohn, Allen Lavoie, Karim Matrah, Grégoire Mesnil, Clemens Mewald, Dan Moldovan, Dominic Monn, Sean Morgan, Tom O'Malley, James Pack, Alexander Pak, Haesun Park, Alexandre Passos, Ankur Patel, Josh Patterson, André Susano Pinto, Anthony Platanios, Anosh Raj, Oscar Ramirez, Anna Revinskaya, Saurabh Saxena, Salim Sémaoune, Ryan Sepassi, Vitor Sessak, Jiri Simsa, Iain Smears, Xiaodan Song, Christina Sorokin, Michel Tessier, Wiktor Tomczak, Dustin Tran, Todd Wang, Pete Warden, Rich Washington, Martin Wicke, Edd Wilder-James, Sam Witteveen, Jason Zaman, Yuefeng Zhou y mi hermano Sylvain.

Por último, pero no por eso menos importante, estoy infinitamente agradecido a mi querida esposa, Emmanuelle, y a nuestros tres maravillosos hijos, Alexandre, Rémi y Gabrielle, por animarme a trabajar duro en este libro. Su curiosidad insaciable no tiene precio: explicar algunos de los conceptos más difíciles de este libro a mi mujer y mis hijos me ayudó a aclarar mis pensamientos y, directamente, mejoró muchas partes de esta obra. ¡Además, me traen galletas y café todo el rato! ¿Qué más se puede pedir?

Sobre el autor

Aurélien Géron es asesor y profesor de *machine learning*. Antiguo *googler*, dirigió el equipo de clasificación de imágenes de YouTube entre 2013 y 2016. Ha sido fundador y CTO de varias empresas diferentes: Wifirst, proveedor de servicios de Internet inalámbrico líder en Francia; Polyconseil, una asesoría centrada en las telecomunicaciones, los medios y la estrategia; y Kiwisoft, una asesoría centrada en el *machine learning* y la privacidad de datos.

Antes de todo eso, Aurélien trabajó como ingeniero en diversos campos: finanzas (JP Morgan y Société Générale), defensa (Departamento de Defensa de Canadá) y sanidad (transfusiones de sangre). También publicó algunos libros técnicos (sobre C++, WiFi y arquitecturas de Internet) y dio clases de informática en una escuela de ingeniería francesa.

Curiosidades: enseñó a sus tres hijos a contar en binario con los dedos (hasta 1.023), estudió microbiología y genética evolutiva antes de pasarse a la ingeniería de software y su paracaídas no se abrió la segunda vez que saltó.



Índice de contenidos

Agradecimientos 5
Sobre el autor 6

Introducción **21**

El *tsunami* del *machine learning* 21
Machine learning en tus proyectos 21
Objetivo y enfoque 22
Ejemplos de código 23
Prerrequisitos 23
Hoja de ruta 24
Cambios entre la primera y la segunda edición 25
Cambios entre la segunda y la tercera edición 25
Otros recursos 26
Convenciones 27
Sobre la imagen de cubierta 28

Parte I. Fundamentos del *machine learning* **29**

Capítulo 1. El paisaje del *machine learning* **31**

¿Qué es el *machine learning*? 32
¿Por qué utilizar *machine learning*? 33
Ejemplos de aplicaciones 35
Tipos de sistemas de *machine learning* 37
Supervisión del entrenamiento 37
Aprendizaje por lotes frente a aprendizaje *online* 44

Aprendizaje basado en instancias frente a aprendizaje basado en modelos ... 48
Principales retos del *machine learning* ... 53
Cantidad insuficiente de datos de entrenamiento ... 54
Datos de entrenamiento no representativos ... 55
Datos de mala calidad ... 56
Características irrelevantes ... 57
Sobreajustar los datos de entrenamiento ... 57
Subajustar los datos de entrenamiento ... 59
Retroceder ... 60
Probar y validar ... 60
Ajuste de hiperparámetros y selección del modelo ... 61
Discrepancia de datos ... 62
Ejercicios ... 64

Capítulo 2. Proyecto de *machine learning* de principio a fin 65

Trabajar con datos reales ... 65
Tener una visión amplia ... 67
Enmarcar el problema ... 67
Seleccionar una medida del rendimiento ... 69
Comprobar las suposiciones ... 71
Obtener los datos ... 72
Ejecutar los ejemplos de código usando Google Colab ... 72
Guardar los cambios en tu código y tus datos ... 74
El poder y el peligro de la interactividad ... 75
Código en el libro frente a código en los cuadernos ... 75
Descargar los datos ... 76
Echar un vistazo rápido a la estructura de los datos ... 77
Crear un conjunto de prueba ... 81
Explorar y visualizar los datos para tener un mayor entendimiento ... 86
Visualizar datos geográficos ... 86
Buscar correlaciones ... 88
Experimentar con combinaciones de atributos ... 91
Preparar los datos para algoritmos de *machine learning* ... 92
Limpiar datos ... 93
Manejar atributos de texto y categóricos ... 96
Escalado de características y transformación ... 99
Transformadores personalizados ... 103
Pipelines de transformación ... 106

Seleccionar un modelo y entrenarlo 111
Entrenar y evaluar con el modelo de entrenamiento 111
Una evaluación mejor utilizando la validación cruzada 113
Perfeccionar el modelo 114
Búsqueda exhaustiva 115
Búsqueda aleatorizada 116
Métodos de ensamblaje 118
Analizar los mejores modelos y sus errores 118
Evaluar el sistema en el conjunto de prueba 119
Lanzar, monitorizar y mantener el sistema 120
¡Pruébalo! 123
Ejercicios 124

Capítulo 3. Clasificación 125

MNIST 125
Entrenar un clasificador binario 127
Medidas de rendimiento 128
Medir la precisión con validación cruzada 129
Matrices de confusión 130
Precisión y sensibilidad 131
Compensación precisión/sensibilidad 133
La curva ROC 136
Clasificación multiclase 140
Análisis de errores 143
Clasificación multietiqueta 146
Clasificación multisalida 148
Ejercicios 149

Capítulo 4. Entrenar modelos 151

Regresión lineal 152
La ecuación normal 154
Complejidad computacional 157
Descenso de gradiente 158
Descenso de gradiente por lotes 161
Descenso de gradiente estocástico 164
Descenso de gradiente por minilotes 167
Regresión polinomial 168
Curvas de aprendizaje 170

Modelos lineales regularizados 174
Regresión de arista 174
Regresión Lasso 176
Regresión de red elástica 179
Detención temprana 180
Regresión logística 181
Calcular probabilidades 182
Entrenamiento y función de pérdida 183
Límites de decisión 184
Regresión *softmax* 187
Ejercicios 191

Capítulo 5. Máquinas de vectores soporte 193

Clasificación SVM lineal 193
Clasificación de margen blando 194
Clasificación SVM no lineal 196
Kernel polinomial 197
Características de similitud 198
Kernel de función de base radial gaussiana 199
Clases SVM y complejidad computacional 201
Regresión SVM 202
Entre bambalinas de los clasificadores SVM lineales 204
El problema dual 206
SVM kernelizadas 207
Ejercicios 210

Capítulo 6. Árboles de decisión 211

Entrenar y visualizar árboles de decisión 211
Hacer predicciones 212
Estimación de probabilidades de clase 215
El algoritmo de entrenamiento CART 215
Complejidad computacional 216
¿Impureza de Gini o entropía? 216
Hiperparámetros de regularización 217
Regresión 219
Sensibilidad a la orientación del eje 221
Los árboles de decisión tiene una varianza alta 222
Ejercicios 223

Capítulo 7. Ensamblaje y *random forests* 225

Clasificadores de votación ... 225
Bagging y *pasting* ... 229
Bagging y *pasting* en Scikit-Learn ... 230
Evaluación fuera de la bolsa ... 232
Parches aleatorios y subespacios aleatorios ... 233
Random forests ... 233
Extra-Trees ... 234
Importancia de las características ... 235
Boosting ... 235
AdaBoost ... 236
Gradient boosting ... 239
Gradient boosting basado en histogramas ... 243
Stacking ... 244
Ejercicios ... 247

Capítulo 8. Reducción de dimensionalidad 249

La maldición de la dimensionalidad ... 250
Enfoques principales para la reducción de dimensionalidad ... 251
Proyección ... 251
Aprendizaje de variedades ... 253
PCA ... 255
Preservar la varianza ... 255
Componentes principales ... 256
Proyección para bajar a *d* dimensiones ... 257
Utilizar Scikit-Learn ... 258
Ratio de varianza explicada ... 258
Elegir el número adecuado de dimensiones ... 258
PCA para compresión ... 260
PCA aleatorizado ... 261
PCA gradual ... 262
Proyección aleatoria ... 263
LLE ... 265
Otras técnicas de reducción de dimensionalidad ... 267
Ejercicios ... 268

Capítulo 9. Técnicas de aprendizaje no supervisado 271

Algoritmos de agrupamiento: k-medias y DBSCAN ... 272
K-medias ... 274
Límites de k-medias ... 284

Utilizar agrupamiento para la segmentación de imágenes 284
Utilizar agrupamiento para aprendizaje semisupervisado 286
DBSCAN 290
Otros algoritmos de agrupamiento 292
Mezclas gaussianas 294
Utilizar mezclas gaussianas para la detección de anomalías 298
Seleccionar el número de grupos 299
Modelos bayesianos de mezcla gaussiana 302
Otros algoritmos para detección de anomalías y de novedades 303
Ejercicios 304

Parte II. Redes neuronales y *deep learning* 307

Capítulo 10. Introducción a las redes neuronales artificiales con Keras 309

De las neuronas biológicas a las artificiales 310
Neuronas biológicas 311
Cálculos lógicos con neuronas 312
El perceptrón 313
El perceptrón multicapa y la retropropagación 317
PMC de regresión 321
PMC de clasificación 323
Implementación de PMC con Keras 325
Creación de un clasificador de imágenes utilizando la API secuencial 325
Creación de un PMC de regresión con la API secuencial 335
Creación de modelos complejos con la API funcional 336
Utilización de la API de subclasificación para crear modelos dinámicos 342
Guardar y restaurar un modelo 343
Utilización de retrollamadas 344
Uso de TensorBoard para visualización 346
Ajuste de los hiperparámetros de una red neuronal 349
Número de capas ocultas 354
Número de neuronas por capa oculta 355
Tasa de aprendizaje, tamaño de lote y otros hiperparámetros 356
Ejercicios 358

Capítulo 11. Entrenar redes neuronales profundas 361

Los problemas de desvanecimiento/explosión de gradientes 362
Inicialización de Glorot y He 363
Funciones de activación mejores 365
Normalización de lotes 371
Recorte de gradiente 376
Reutilizar redes preentrenadas 377
Aprendizaje por transferencia con Keras 378
Preentrenamiento no supervisado 380
Preentrenar con una tarea auxiliar 381
Optimizadores más rápidos 382
Momentum 382
Gradiente acelerado de Nesterov 384
AdaGrad 385
RMSProp 386
Adam 387
AdaMax 388
Nadam 388
AdamW 389
Programación de la tasa de aprendizaje 390
Evitar el sobreajuste mediante la regularización 395
Regularizaciones ℓ_1 y ℓ_2 395
Dropout 396
Monte Carlo (MC) Dropout 399
Regularización *max-norm* 402
Resumen y directrices prácticas 403
Ejercicios 404

Capítulo 12. Modelos personalizados y entrenamiento con TensorFlow 407

Un *tour* rápido por TensorFlow 407
Utilizar TensorFlow como NumPy 411
Tensores y operaciones 411
Tensores y NumPy 413
Conversiones de tipos 413
Variables 414
Otras estructuras de datos 414
Personalizar modelos y algoritmos de entrenamiento 415
Funciones de pérdida personalizadas 415

Guardar y cargar modelos que contengan componentes personalizados 416
Funciones de activación, inicializadores, regularizadores y restricciones personalizados 418
Métricas personalizadas 420
Capas personalizadas 422
Modelos personalizados 425
Pérdidas y métricas basadas en componentes internos del modelo 427
Calcular gradientes utilizando diferenciación automática 429
Bucles de entrenamiento personalizados 432
Grafos y funciones de TensorFlow 435
AutoGraph y trazado 437
Reglas de las funciones TF 439
Ejercicios 440

Capítulo 13. Cargar y preprocesar datos con TensorFlow 443

La API tf.data 444
Encadenar transformaciones 445
Mezclar los datos 447
Preprocesar los datos 450
Juntarlo todo 451
Precarga 451
Utilizar el conjunto de datos con Keras 454
El formato TFRecord 455
Archivos TFRecord comprimidos 456
Breve introducción a los búferes de protocolo 456
Protobufs de TensorFlow 458
Cargar y analizar ejemplos 459
Manejar listas de listas usando el protobuf SequenceExample 460
Capas de preprocesamiento de Keras 461
La capa Normalization 462
La capa Discretization 464
La capa CategoryEncoding 465
La capa StringLookup 466
La capa Hashing 467
Codificar características categóricas utilizando *embeddings* 468
Preprocesamiento de texto 472
Utilizar componentes de modelos de lenguaje preentrenados 474
Capas de preprocesamiento de imágenes 475
El proyecto TensorFlow Datasets 476
Ejercicios 477

Capítulo 14. *Deep learning* para visión por ordenador usando redes neuronales convolucionales 479

La arquitectura de la corteza visual 480
Capas convolucionales 481
Filtros 483
Apilar múltiples mapas de características 485
Implementar capas convolucionales con Keras 487
Requisitos de memoria 490
Capas de *pooling* 491
Implementación de capas de *pooling* con Keras 493
Arquitecturas de RNC 495
LeNet-5 498
AlexNet 499
GoogLeNet 502
VGGNet 505
ResNet 505
Xception 509
SENet 510
Otras arquitecturas destacables 512
Elegir la arquitectura de RNC adecuada 514
Implementar una RNC ResNet-34 usando Keras 515
Utilizar modelos preentrenados desde Keras 516
Modelos preentrenados para aprendizaje por transferencia 518
Clasificación y localización 521
Detección de objetos 522
Redes completamente convolucionales 525
You Only Look Once 527
Seguimiento de objetos 530
Segmentación semántica 531
Ejercicios 535

Capítulo 15. Procesar secuencias utilizando RNR y RNC 537

Capas y neuronas recurrentes 538
Celdas de memoria 540
Secuencias de entrada y salida 541
Entrenamiento de RNR 542
Predicción de una serie temporal 543
La familia de modelos ARMA 549
Preparar los datos para modelos de *machine learning* 552
Realizar un pronóstico utilizando un modelo lineal 555

Realizar un pronóstico utilizando una RNR simple 556
Realizar un pronóstico utilizando una RNR profunda 557
Predecir series temporales multivariables 559
Predecir varios pasos de tiempo más adelante 560
Realizar un pronóstico utilizando un modelo secuencia a secuencia 562
Manejar secuencias largas 565
Enfrentarse al problema de los gradientes inestables 565
Enfrentarse al problema de la memoria a corto plazo 568
Ejercicios 576

Capítulo 16. Procesamiento de lenguaje natural con RNR y atención 577

Generar texto shakespeariano utilizando una RNR a nivel de carácter 578
Crear el conjunto de datos de entrenamiento 579
Crear y entrenar el modelo de RNR a nivel de carácter 581
Generar texto shakespeariano falso 582
RNR con estado 584
Análisis de sentimiento 587
Enmascaramiento 590
Reutilización de *embeddings* y modelos de lenguaje preentrenados 593
Una red codificador-descodificador para la traducción automática neuronal 595
RNR bidireccionales 601
Haz local (*beam search*) 603
Mecanismos de atención 605
Solo necesitas atención: la arquitectura Transformer original 609
Una avalancha de modelos de transformador 620
Transformadores de visión 624
Biblioteca Transformers de Hugging Face 629
Ejercicios 633

Capítulo 17. Autocodificadores, GAN y modelos de difusión 635

Representaciones de datos eficientes 637
Realizar PCA con un autocodificador lineal incompleto 639
Autocodificadores apilados 640
Implementación de un autocodificador apilado usando Keras 641
Visualizar las reconstrucciones 642
Visualizar el conjunto de datos Fashion MNIST 643

Preentrenamiento no supervisado utilizando autocodificadores apilados 644
Atar pesos 645
Entrenar autocodificadores de uno en uno 646
Autocodificadores convolucionales 648
Autocodificador con eliminación de ruido 649
Autocodificadores dispersos 651
Autocodificadores variacionales 654
Generar imágenes de Fashion MNIST 658
Redes generativas antagónicas 659
Las dificultades de entrenar GAN 663
GAN convolucionales profundas 665
Crecimiento progresivo de GAN 668
StyleGAN 671
Modelos de difusión 673
Ejercicios 681

Capítulo 18. Aprendizaje por refuerzo 683

Aprender a optimizar recompensas 684
Búsqueda de políticas 685
Introducción a OpenAI Gym 687
Políticas de redes neuronales 691
Evaluar acciones: el problema de la asignación de crédito 693
Gradientes de política 694
Procesos de decisión de Markov 699
Aprendizaje por diferencia temporal 703
Aprendizaje Q-Learning 704
Políticas de exploración 706
Q-Learning aproximado y Q-Learning profundo 707
Implementación de Q-Learning profundo 708
Variantes del Q-Learning profundo 713
Objetivos de valores Q fijos 713
Double DQN 714
Repetición de experiencia priorizada 714
Dueling DQN 715
Visión general de algunos algoritmos de aprendizaje por refuerzo populares 716
Ejercicios 720

Capítulo 19. Entrenar y desplegar modelos de TensorFlow a escala 721

Servir un modelo de TensorFlow ... 722
Utilizar TensorFlow Serving ... 722
Crear un servicio de predicción en Vertex AI ... 731
Realizar tareas de predicción por lotes en Vertex AI ... 738
Desplegar un modelo en un dispositivo móvil o integrado ... 740
Ejecutar un modelo en una página web ... 743
Uso de GPU para acelerar la computación ... 745
Conseguir tu propia GPU ... 746
Gestionar la RAM de la GPU ... 748
Colocar operaciones y variables en dispositivos ... 751
Ejecución paralela en múltiples dispositivos ... 752
Entrenar modelos en múltiples dispositivos ... 755
Paralelismo del modelo ... 755
Paralelismo de datos ... 758
Entrenar y escalar con la API de estrategia de distribución ... 764
Entrenar un modelo en un clúster de TensorFlow ... 765
Ejecutar trabajos de entrenamiento grandes en Vertex AI ... 769
Ajuste de hiperparámetros en Vertex AI ... 771
Ejercicios ... 775
¡Gracias! ... 776

Parte III. Apéndices 777

Apéndice A. Lista de comprobación de proyectos de *machine learning* 779

Enmarcar el problema y tener una visión amplia ... 779
Obtener los datos ... 780
Explorar los datos ... 780
Preparar los datos ... 781
Seleccionar modelos prometedores ... 782
Perfeccionar el sistema ... 782
Presentar la solución ... 783
¡Lanzar! ... 783

Apéndice B. Diferenciación automática 785

Diferenciación manual 785
Aproximación mediante diferencias finitas 786
Diferencia automática hacia delante 787
Diferenciación automática inversa 790

Apéndice C. Estructuras de datos especiales 793

Cadenas 793
Tensores irregulares 794
Tensores dispersos 795
Matrices tensoriales 796
Conjuntos 797
Colas 797

Apéndice D. Grafos de TensorFlow 799

Funciones TF y funciones concretas 799
Explorar definiciones y grafos de función 801
Atención al trazado 802
Utilizar AutoGraph para capturar estructuras de control 804
Manejar variables y otros recursos en funciones TF 805
Utilizar funciones TF con Keras (o no) 806

Índice alfabético 809



Introducción

El *tsunami* del *machine learning*

En 2006, Geoffrey Hinton *et al.* publicaron un artículo (`https://homl.info/136`)[1] que mostraba cómo entrenar una red neuronal profunda capaz de reconocer dígitos escritos a mano con una precisión de vanguardia (> 98 %). Denominaron a esta técnica "*deep learning*". Una red neuronal profunda es un modelo (muy) simplificado de nuestro córtex cerebral, compuesto por una pila de capas de neuronas artificiales. A nivel general, en aquel momento se consideraba que entrenar una red neuronal profunda era imposible[2] y la mayoría de los investigadores habían abandonado la idea a finales de los 90. Ese artículo reavivó el interés de la comunidad científica y enseguida aparecieron muchos artículos nuevos que demostraban que el *deep learning* no solo era posible, sino que era capaz de alcanzar logros asombrosos que ninguna otra técnica de *machine learning* (ML) podía igualar (con la ayuda de una capacidad de procesamiento tremenda y una cantidad de datos enorme). Este entusiasmo se extendió con rapidez a muchas otras áreas del *machine learning*.

Una década después, el *machine learning* ha conquistado la industria y hoy está en el centro de gran parte de la magia de los productos de alta tecnología, ordena nuestros resultados de búsqueda en la web, permite el reconocimiento de discurso de nuestros *smartphones* y, quizá incluso, conduciendo tu coche.

Machine learning en tus proyectos

¡Es natural que estés entusiasmado por el *machine learning* y quieras unirte a la fiesta!

¿Te gustaría, quizá, dotar de un cerebro propio a tu robot casero? ¿Hacer que reconociese caras? ¿Que aprendiese a andar?

---

1. Geoffrey E. Hinton *et al.*, "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation* 18 (2006): 1527-1554.
2. A pesar de que las redes neuronales convolucionales profundas de Yann LeCun habían funcionado bien para el reconocimiento de imágenes desde los 90, no eran para fines generales.

Tal vez tu empresa tiene montones de datos (registros de usuario, datos financieros, datos de producción, datos de sensores de máquinas, estadísticas de llamadas a la línea directa, informes de recursos humanos, etc.) y es muy probable que pudieses encontrar algún tesoro oculto si supieses donde buscar. Con el *machine learning*, podrías conseguir lo siguiente, entre otras muchas cosas (`https://homl.info/usecases`):

- Segmentar clientes y encontrar la mejor estrategia de marketing para cada grupo.
- Recomendar productos para cada cliente en función de lo que han comprado clientes similares.
- Detectar qué transacciones tienen probabilidades de ser fraudulentas.
- Predecir los ingresos del año siguiente.

Sea cual sea la razón, has decidido aprender *machine learning* e implementarlo en tus proyectos. ¡Buena idea!

Objetivo y enfoque

Este libro da por hecho que no sabes prácticamente nada de *machine learning*. Su objetivo es proporcionarte los conceptos, las herramientas y la intuición que necesitas para implementar programas capaces de aprender a partir de datos.

Hablaremos de una gran cantidad de técnicas, desde las más sencillas y de uso más común (como la regresión lineal) a algunas de las técnicas de *deep learning* que ganan competiciones con regularidad. Para eso, utilizaremos *frameworks* de Python listos para la producción:

- Scikit-Learn (`http://scikit-learn.org/`) es muy fácil de utilizar y, aun así, implementa muchos algoritmos de *machine learning* de forma eficiente, así que es un punto de partida estupendo para aprender *machine learning*. Lo creó David Cournapeau en 2007 y ahora lo dirige un equipo de investigadores del INRIA (Institut National de Recherche en Informatique et en Automatique, instituto francés para la investigación en informática y automatización).
- TensorFlow (`https://tensorflow.org/`) es una biblioteca más compleja para computación numérica distribuida. Hace que sea posible entrenar y ejecutar redes neuronales muy grandes de manera eficiente al distribuir las computaciones a través de cientos de servidores con múltiples GPU (unidades de procesamiento gráfico) potenciales. TensorFlow (TF) se creó en Google y soporta muchas de sus aplicaciones de *machine learning* a gran escala. Se lanzó como código abierto en noviembre de 2015 y la versión 2.0 se lanzó en septiembre de 2019.
- Keras (`https://keras.io/`) es una API de *deep learning* de alto nivel que hace que sea muy fácil entrenar y ejecutar redes neuronales. Keras viene con TensorFlow y depende de él para todos los cálculos intensivos.

Este libro adopta un enfoque práctico, desarrollando un entendimiento intuitivo del *machine learning* a través de ejemplos concretos en funcionamiento y solo un poco de teoría.

> **Truco:** Aunque puedes leer este libro sin coger el ordenador, te recomiendo encarecidamente que experimentes con los ejemplos de código.

Ejemplos de código

Todos los ejemplos de código de este libro son de código abierto y están disponibles en línea en `https://github.com/ageron/handson-ml3`, como cuadernos de Jupyter. Se trata de documentos interactivos que contienen texto, imágenes y fragmentos de código ejecutable (en nuestro caso, Python). La manera más sencilla y rápida de empezar es ejecutar estos cuadernos usando Google Colab: se trata de un servicio gratuito que te permite ejecutar cualquier cuaderno de Jupyter directamente en línea, sin tener que instalar nada en la máquina. Solo necesitas un navegador web y una cuenta de Google.

> **Nota:** En este libro, voy a asumir que utilizas Google Colab, pero también he probado los cuadernos en otras plataformas en línea, como Kaggle y Binder, así que puedes usar esas si prefieres. Como alternativa, puedes instalar las bibliotecas y herramientas (o la imagen de Docker para este libro) y ejecutar los cuadernos de forma directa en tu propio ordenador. Consulta las instrucciones `https://homl.info/install`.

Prerrequisitos

Este libro da por hecho que tienes algo de experiencia en programación con Python. Si todavía no conoces Python, `https://learnpython.org` es un punto de partida estupendo. El tutorial oficial en Python.org (`https://docs.python.org/3/tutorial`) también está bastante bien.

El libro también da por hecho que estás familiarizado con las bibliotecas científicas principales de Python, en concreto, NumPy (`http://numpy.org/`), Pandas (`http://pandas.pydata.org/`) y Matplotlib (`http://matplotlib.org/`). Si nunca has utilizado estas bibliotecas, no te preocupes; son fáciles de aprender y he creado un tutorial para cada una. Puedes acceder a ellos en línea en `https://homl.info/tutorials`.

Además, si quieres entender por completo cómo funcionan los mecanismos de *machine learning* (y no solo utilizarlos), deberías tener al menos una comprensión básica de algunos conceptos matemáticos, en especial el álgebra lineal. Específicamente, deberías saber qué son los vectores y las matrices, y cómo realizar operaciones simples, como sumar vectores o transponer y multiplicar matrices. Si necesitas una introducción rápida al álgebra lineal (¡en realidad no es muy difícil!), te ofrezco un tutorial en `https://homl.info/tutorials`. También encontrarás un tutorial sobre cálculo diferencial, que puede resultarte útil para entender cómo se entrenan las redes neuronales, pero no es imprescindible para asimilar los conceptos importantes. Este libro también usa de vez en cuando otros conceptos matemáticos, como

exponenciales y logaritmos, algo de teoría de la probabilidad y algunos conceptos básicos de estadística, pero nada demasiado avanzado. Si necesitas ayuda con algo de esto, puedes consultar `https://khanacademy.org`, que ofrece muchos cursos de matemáticas en línea excelentes y gratuitos.

Hoja de ruta

Este libro está organizado en dos partes. La parte I, Fundamentos del *machine learning*, trata los siguientes temas:

- Qué es el *machine learning*, qué problemas intenta resolver y cuáles son las principales categorías y los conceptos fundamentales en sus sistemas.
- Los pasos de un proyecto de *machine learning* típico.
- Aprender ajustando un modelo a unos datos.
- Optimizar una función de pérdida.
- Manejar, limpiar y preparar datos.
- Seleccionar y utilizar ingeniería de características.
- Seleccionar un modelo y ajustar hiperparámetros utilizando validación cruzada.
- Los retos del *machine learning*, en concreto el subajuste y el sobreajuste (el compromiso sesgo-varianza).
- Los algoritmos de aprendizaje más comunes: regresión lineal y polinomial, regresión logística, *k* vecinos más cercanos, máquinas de vectores soporte, árboles de decisiones, *random forests* y métodos de ensamblaje.
- Reducir la dimensionalidad de los datos de entrenamiento para combatir la "maldición de la dimensionalidad".
- Otras técnicas de aprendizaje no supervisado, incluyendo el agrupamiento, la estimación de densidad y la detección de anomalías.

La parte II, Redes neuronales y *deep learning*, trata los siguientes temas:

- Qué son las redes neuronales y para qué sirven.
- Crear y entrenar redes neuronales utilizando TensorFlow y Keras.
- Las arquitecturas de redes neuronales más importantes: redes neuronales prealimentadas para datos tabulares, redes convolucionales para visión por ordenador, redes recurrentes y redes LSTM (*long short-term memory*) para el procesamiento de secuencias, codificadores/descodificadores y transformadores para el procesamiento de lenguaje natural (¡y más!), autocodificadores, redes generativas antagónicas (GAN) y modelos de difusión para aprendizaje generativo.
- Técnicas para entrenar redes neuronales profundas.
- Cómo crear un agente (por ejemplo, un bot en un juego) que pueda aprender buenas estrategias mediante un enfoque de prueba y error, utilizando el aprendizaje por refuerzo.
- Cargar y preprocesar cantidades grandes de datos con eficiencia.
- Entrenar y desplegar modelos de TensorFlow a escala.

La primera parte se basa sobre todo en Scikit-Learn, mientras que la segunda parte utiliza TensorFlow y Keras.

> **Advertencia:** No te sumerjas a lo loco donde no hagas pie: aunque el *deep learning* es, sin duda, una de las áreas más apasionantes del *machine learning*, deberías dominar los conceptos básicos primero. Además, la mayoría de los problemas pueden resolverse bastante bien mediante técnicas más sencillas, como los *random forests* y los métodos de ensamblaje (que se explican en la parte I). El *deep learning* es más adecuado para problemas complejos, como el reconocimiento de imágenes, el reconocimiento de discurso o el procesamiento de lenguaje natural, y requiere muchos datos, capacidad de procesamiento y paciencia (a menos que puedas hacer uso de una red neuronal preentrenada, como veremos).

Cambios entre la primera y la segunda edición

Si ya has leído la primera edición, estos son los cambios principales entre la primera y la segunda edición:

- Todo el código se migró de TensorFlow 1.x a TensorFlow 2.x, y sustituí la mayoría del código de TensorFlow de nivel bajo (grafos, sesiones, columnas de características, estimadores, etc.) por código de Keras mucho más simple.
- La segunda edición introdujo la API Data para cargar y preprocesar conjuntos de datos grandes, la API de estrategias de distribución para entrenar y desplegar modelos de TF a escala, TF Serving y AI Platform de Google Cloud para pasar modelos a producción y (brevemente) TF Transform, TFLite, TF Addons/Seq2Seq, TensorFlow.js y TF Agents.
- También se introdujeron temas adicionales de *machine learning*, incluyendo el aprendizaje no supervisado, técnicas de visión por ordenador para la detección de objetos y la segmentación semántica, el manejo de secuencias usando redes neuronales convolucionales (RNC), procesamiento del lenguaje natural (PLN) usando redes neuronales recurrentes (RNR), RNC y transformadores, GAN y más.

Cambios entre la segunda y la tercera edición

Si has leído la segunda edición, estos son los principales cambios entre la segunda y la tercera edición:

- Todo el código se ha actualizado a las versiones más recientes de las bibliotecas. En concreto, esta tercera edición incluye muchas adiciones nuevas de Scikit-Learn (por ejemplo, seguimiento de nombres de características, *gradient boosting* basado en histogramas, propagación de etiquetas y más). También introduce la biblioteca Keras Tuner

para el ajuste de hiperparámetros, la biblioteca Transformers de Hugging Face para el procesamiento del lenguaje natural y nuevas capas de preprocesamiento y aumentos de datos de Keras.

- Se han añadido varios modelos de visión (ResNeXt, DenseNet, MobileNet, CSPNet y EfficientNet), así como directrices para elegir el adecuado.
- Ahora, el capítulo 15 analiza datos sobre cantidad de pasajeros del autobús y el tren en Chicago en vez de series temporales generadas, e introduce el modelo ARMA y sus variantes.
- En el capítulo 16 sobre el procesamiento del lenguaje natural ahora se crea un modelo de traducción inglés-español, usando primero una RNR codificador-descodificador y, después, utilizando un modelo transformador. El capítulo también aborda modelos de lenguaje como Switch Transformers, DistilBERT, T5 y PaLM (con desencadenamiento de cadena de pensamiento). Además, introduce transformadores de visión (ViT) y ofrece una visión general de algunos modelos visuales basados en transformadores, como los transformadores de imagen con eficiencia de datos (DeiT), Perceiver y DINO, además de un breve repaso general de algunos modelos multimodales, incluidos CLIP, DALL·E, Flamingo y GATO.
- El capítulo 17 sobre aprendizaje generativo introduce ahora modelos de difusión y muestra cómo implementar un modelo probabilístico de difusión para eliminar ruido (DDPM) desde cero.
- El capítulo 19 ha migrado de AI Platform de Google Cloud a Google Vertex AI, y utiliza Keras Tuner distribuido para la búsqueda de hiperparámetros a gran escala. Ahora, el capítulo incluye código TensorFlow.js con el que puedes experimentar en línea. También introduce técnicas de entrenamiento distribuido adicionales, incluyendo PipeDream y Pathways.
- Para permitir todo el contenido nuevo, algunas secciones se han convertido en contenido en línea en inglés, incluyendo las instrucciones de instalación, el *kernel* PCA (análisis de componentes principales), detalles matemáticos de mezclas gaussianas en modelos bayesianos, TF Agents y los antiguos apéndices A (soluciones a los ejercicios), C (matemáticas para máquinas de vector soporte) y E (arquitecturas adicionales de redes neuronales).

Otros recursos

Existen muchos recursos excelentes disponibles para aprender acerca del *machine learning*. Por ejemplo, el curso de *machine learning* de Andrew Ng en Coursera (`https://homl.info/ngcourse`) es increíble, aunque requiere una inversión de tiempo considerable. También hay muchos sitios web interesantes sobre *machine learning*, incluyendo la excepcional guía de usuario de Scikit-Learn (`https://homl.info/skdoc`). Quizá también te guste Dataquest (`https://www.dataquest.io/`), que ofrece tutoriales interactivos muy buenos, y blogs sobre *machine learning* como los que aparecen recogidos en Quora (`https://homl.info/1`).

Hay un montón de libros introductorios sobre *machine learning*, en particular:

- *Ciencia de datos desde cero* (Anaya Multimedia), de Joel Grus, presenta los fundamentos del *machine learning* e implementa algunos de los algoritmos principales en Python (desde cero).
- *Machine Learning: An Algorithmic Perspective*, segunda edición (Chapman & Hall), de Stephen Marsland, es una introducción estupenda al *machine learning* que abarca una amplia variedad de temas en profundidad con ejemplos de código en Python (también desde cero, pero utilizando NumPy).
- *Python Machine Learning* (Marcombo, Ediciones Técnicas), de Sebastian Raschkan, también es una gran introducción al *machine learning* y utiliza las bibliotecas de código abierto de Python (Pylearn 2 y Theano).
- *Deep Learning con Python* (Anaya Multimedia), de François Chollet, es un libro muy práctico que abarca una gran variedad de temas de manera clara y concisa, como cabría esperar del creador de la excelente biblioteca Keras. Prioriza los ejemplos de código sobre la teoría matemática.
- *The Hundred-Page Machine Learning Book* (autopublicado), de Andriy Burkov, es muy corto, pero abarca una cantidad impresionante de temas, que presenta con términos accesibles sin huir de las ecuaciones matemáticas.
- *Learning from Data* (AMLBook), de Yaser S. Abu-Mostafa, Malik Magdon-Ismail y Hsuan-Tien Lin, es un enfoque bastante teórico al *machine learning* que ofrece conocimientos profundos, en especial respecto al compromiso sesgo-varianza (véase el capítulo 4).
- *Inteligencia artificial: un enfoque moderno* (Grupo Anaya Publicaciones Generales), de Stuart Russell y Peter Norvig, es un libro genial (y gordísimo) que abarca una cantidad increíble de temas, incluyendo el *machine learning*. Ayuda a poner el *machine learning* en perspectiva.
- *Deep Learning for Coders with fastai and PyTorch* (O'Reilly), de Jeremy Howard y Sylvain Gugger, ofrece una introducción práctica y clarísima al *deep learning* utilizando las bibliotecas fastai y PyTorch.

Por último, unirse a sitios web de competiciones *machine learning*, como Kaggle.com, te permitirá practicar tus habilidades en problemas del mundo real, con la ayuda y las perspectivas de algunos de los mejores profesionales del *machine learning* que hay.

Convenciones

Para ayudarte a sacar el mayor partido al texto y saber dónde te encuentras en cada momento, a lo largo del libro utilizamos distintas convenciones:

- Las combinaciones de teclas se muestran en negrita, como por ejemplo **Control-A**.
- Los nombres de archivo, URL y código incluido en texto se muestran en un tipo de letra `monoespacial`.

- Los menús, submenús, opciones, cuadros de diálogo y demás elementos de la interfaz de las aplicaciones se muestran en un tipo de letra Arial.
- Dentro de los ejemplos de código, utilizaremos las siguientes convenciones:
  - Los comandos u otro texto que el usuario debería escribir de manera literal se muestran en negrita.
  - El texto que hay que sustituir por valores proporcionados por el usuario o determinados por el contexto se muestra en cursiva.

> **Nota:** En estos cuadros se incluye información importante directamente relacionada con el texto adjunto. Los trucos, sugerencias y comentarios afines relacionados con el tema analizado se reproducen en este formato.

Sobre la imagen de cubierta

El animal de la portada de este libro es la salamandra de fuego (*Salamandra salamandra*), un anfibio que se encuentra en la mayor parte de Europa. Su piel negra y brillante presenta manchas amarillas en la cabeza y el dorso, que indican la presencia de toxinas alcaloides. Este es un posible origen del nombre común de este anfibio: el contacto con estas toxinas (que también puede rociar a una distancia corta) causa convulsiones e hiperventilación. Bien los dolorosos venenos, bien la humedad de la piel de la salamandra (o ambas cosas) llevaron a la creencia equivocada de que estas criaturas no solo sobrevivían si se metían en fuego, sino que también podían apagarlo.

Las salamandras de fuego viven en bosques sombríos, ocultas en grietas húmedas y bajo troncos cerca de estanques u otros cuerpos de agua dulce que facilitan su reproducción. Aunque pasan la mayor parte de su vida en tierra, tienen a sus crías en el agua. Subsisten sobre todo gracias a una dieta a base de insectos, arañas, babosas y gusanos. Las salamandras de fuego pueden alcanzar algo más de 30 cm de longitud y en cautividad pueden llegar a vivir hasta 50 años.

La población de salamandras de fuego se ha visto reducida a causa de la destrucción de su hábitat y de su captura para venderlas como mascotas, pero la mayor amenaza a la que se enfrentan es la sensibilidad de su piel permeable a contaminantes y microbios. Desde 2014, se han extinguido en partes de Holanda y Bélgica debido a la introducción de hongos.

La imagen de la cubierta es de *Wood's Illustrated Natural History*.

Parte I

Fundamentos del *machine learning*

1 El paisaje del *machine learning*

No hace tanto tiempo, si hubieses cogido tu teléfono y le hubieses preguntado por dónde ir a casa, te habría ignorado (y la gente habría puesto en duda tu cordura). Pero el *machine learning* ya no es ciencia ficción; miles de millones de personas lo usan a diario. La verdad es que lleva décadas en aplicaciones especializadas, como el reconocimiento óptico de caracteres (*Optical Character Recognition*, OCR). La primera aplicación de *machine learning* que se volvió realmente popular, mejorando las vidas de cientos de millones de personas, ya se apoderó del mundo en los 90: el filtro de *spam*. No es que sea exactamente un robot consciente de sí mismo, pero técnicamente puede considerarse *machine learning*: en realidad, ha aprendido tan bien que ya casi nunca necesitamos marcar un correo como *spam*. Le siguieron cientos de aplicaciones de *machine learning* que hoy en día controlan cientos de productos y características que utilizamos con regularidad: mensajes reproducidos por respuesta de voz interactiva, traducción automática, búsqueda de imágenes, recomendaciones de productos y mucho más.

¿Dónde empieza el *machine learning* y dónde acaba? ¿Qué significa exactamente para una máquina "aprender" algo? Si descargo una copia de todos los artículos de Wikipedia, ¿ha aprendido algo mi ordenador? ¿Es más inteligente de repente? En este capítulo, vamos a empezar por aclarar qué es el *machine learning* y por qué puede interesarnos utilizarlo.

Después, antes de ponernos a explorar el continente del *machine learning*, echaremos un vistazo al mapa y aprenderemos cuáles son las regiones principales y los puntos de referencia más notables: aprendizaje supervisado frente a aprendizaje no supervisado y sus variantes, aprendizaje *online* frente a aprendizaje por lotes, aprendizaje basado en instancias frente a aprendizaje basado en modelos. A continuación, echaremos un vistazo al flujo de trabajo de un proyecto de *machine learning* típico, hablaremos de los principales retos con los que puedes encontrarte y nos ocuparemos de cómo evaluar y perfeccionar un sistema de *machine learning*.

Este capítulo introduce un montón de conceptos fundamentales (y jerga) que todo científico de datos debería saberse de memoria. Será una visión general de alto nivel (es el único capítulo sin mucho código), todo bastante simple, pero mi objetivo es garantizar que todo está claro como el agua antes de que sigamos con el resto del libro. ¡Coge un café y vamos a empezar!

> **Truco:** Si ya estás familiarizado con los conceptos básicos del *machine learning*, puede que te interese pasar directamente al capítulo 2. Si no estás seguro, prueba a contestar todas las preguntas enumeradas al final de este capítulo antes de avanzar.

¿Qué es el *machine learning*?

El *machine learning* es la ciencia (y el arte) de programar ordenadores para que aprendan a partir de datos.

Vamos a ver una definición un poco más general:

> [El *machine learning* es el] campo de estudio que da a los ordenadores la capacidad de aprender sin ser programados de manera explícita.
>
> —Arthur Samuel, 1959

Y una más orientada a la ingeniería:

> Se dice que un programa de ordenador aprende de la experiencia E, con respecto a una tarea T y una medida de rendimiento R, si su rendimiento en T, medido por P, mejora con la experiencia E.
>
> —Tom Mitchell, 1997

El filtro de *spam* es un programa de *machine learning* que, al recibir ejemplos de correo basura (marcados por los usuarios) y ejemplos de correos corrientes (que no sean *spam*, también llamados "*ham*"), puede aprender a marcar el *spam*. Los ejemplos que el sistema utiliza para aprender se llaman conjunto de entrenamiento. Cada ejemplo de entrenamiento se llama instancia de entrenamiento (o muestra). La parte de un sistema de *machine learning* que aprende y realiza predicciones se denomina modelo. Las redes neuronales y los *random forests* son ejemplo de modelos.

En este caso, la tarea T es marcar el *spam* para los correos nuevos, la experiencia E son los datos de entrenamiento y la medida del rendimiento tiene que definirse; por ejemplo, podemos utilizar la proporción de correos clasificados correctamente. Esta medida de rendimiento particular se llama exactitud y se utiliza a menudo en tareas de clasificación.

Si descargamos una copia de todos los artículos de Wikipedia, sin más, nuestro ordenador tiene muchos más datos, pero no es de repente mejor en ninguna tarea. Eso no es *machine learning*.

¿Por qué utilizar *machine learning*?

Piensa en cómo escribirías un filtro de *spam* utilizando técnicas de programación tradicionales (véase la figura 1.1):

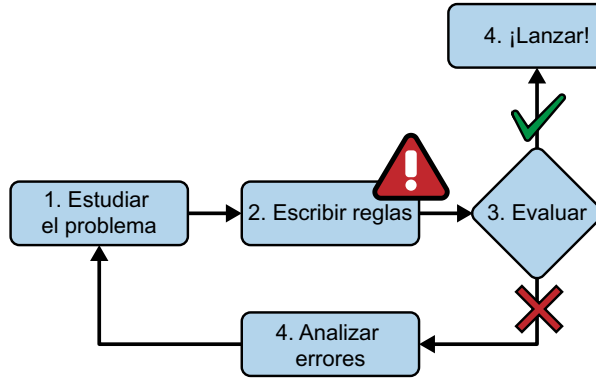


Figura 1.1. El enfoque tradicional.

1. Primero, examinarías cuál es el aspecto típico del *spam*. Podrías fijarte en que algunas palabras o frases (como "Para usted", "tarjeta de crédito", "gratis" e "impresionante") tienden a aparecer mucho en el asunto. Quizá también notarías otros patrones en el nombre del remitente, el cuerpo del correo y en otras partes del mismo.
2. Escribirías un algoritmo de detección para cada uno de los patrones observados y el programa marcaría los correos como *spam* si se detectase una cantidad de esos patrones.
3. Probarías el programa y repetirías los pasos 1 y 2 hasta que fuese lo bastante bueno para lanzarse.

Puesto que el problema es difícil, es probable que el programa se convierta en una lista larga de reglas complejas; sería bastante difícil de mantener.

Por el contrario, un filtro de *spam* basado en *machine learning* aprende de manera automática qué palabras y frases son buenas predictoras de *spam* al detectar patrones de palabras atípicamente frecuentes en ejemplos de *spam* comparados con ejemplos de *ham* (véase la figura 1.2). El programa es mucho más corto, más fácil de mantener y, probablemente, más exacto.

¿Qué pasa si las personas que envían *spam* se dan cuenta de que todos sus correos que contienen "Para usted" se bloquean? Podrían empezar a escribir "Para ti" en su lugar. Un filtro de *spam* que utilice técnicas de programación tradicionales tendría que actualizarse para marcar los correos que tuviesen "Para ti". Si la gente que envía *spam* sigue encontrando maneras de evitar el filtro, tendrás que seguir escribiendo reglas nuevas para siempre.

Por el contrario, un filtro de *spam* basado en técnicas de *machine learning* se da cuenta de forma automática de que "Para ti" se ha vuelto atípicamente frecuente en los correos marcados como *spam* por los usuarios y empieza a marcarlos sin nuestra intervención (véase la figura 1.3).

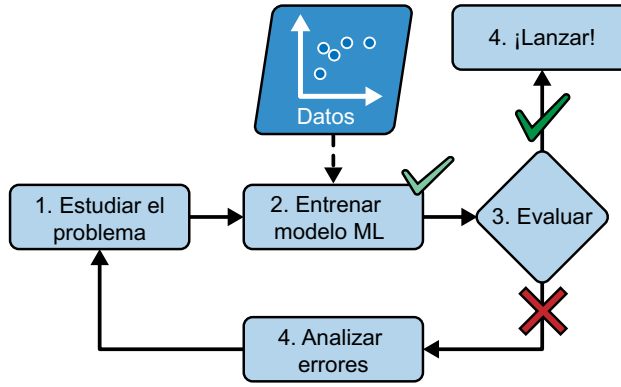


Figura 1.2. El enfoque del *machine learning*.

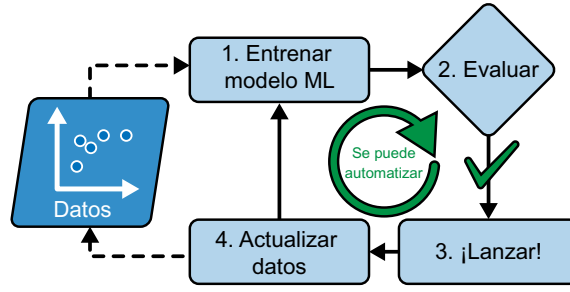


Figura 1.3. Adaptación al cambio de forma automática.

Otra área en la que destaca el *machine learning* es la solución de problemas que, o bien son demasiado complejos para enfoques tradicionales, o bien no tienen un algoritmo conocido. Por ejemplo, piensa en el reconocimiento de discurso. Digamos que queremos empezar con algo sencillo y escribir un programa capaz de distinguir las palabras "one" y "two". Puede que te des cuenta de que la palabra "two" empieza con un sonido agudo ("T"), así que podrías incrustar en el código un algoritmo que mida la intensidad del sonido agudo y utilice eso para distinguir todos los casos de "one" y "two", pero está claro que esa técnica no va a escalarse a miles de palabras pronunciadas por millones de personas muy diferentes en entornos ruidosos y en docenas de idiomas. La mejor solución, al menos a día de hoy, es escribir un algoritmo que aprenda por sí mismo cuando se le den muchas grabaciones de ejemplo para cada palabra.

Por último, el *machine learning* puede ayudar a los humanos (véase la figura 1.4). Los modelos de *machine learning* pueden inspeccionarse para ver lo que han aprendido (aunque, con algunos modelos, eso puede ser difícil). Por ejemplo, una vez que un filtro de *spam* se ha entrenado con el *spam* suficiente, puede inspeccionarse con facilidad para revelar la lista de palabras y combinación de palabras que él cree que son las mejores predictoras de *spam*. A veces, esto revelará correlaciones insospechadas o tendencias nuevas y, por tanto, llevará a una

mejor comprensión del problema. Explorar a fondo grandes cantidades de datos para descubrir patrones ocultos se conoce como "minería de datos", y el *machine learning* es excelente para ello.

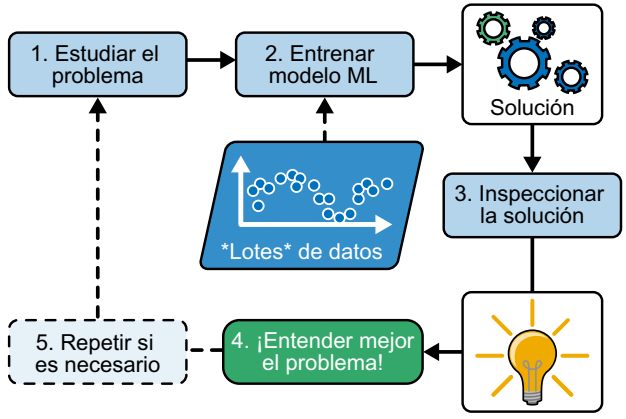


Figura 1.4. El *machine learning* puede ayudar a los humanos a aprender.

Para resumir, el *machine learning* es genial para:

- Problemas cuyas soluciones existentes requieren muchos ajustes o largas listas de reglas (a menudo, un modelo de *machine learning* puede simplificar código y tener un rendimiento mejor que el enfoque tradicional).
- Problemas complejos para los cuales el uso de un enfoque tradicional no ofrece una buena solución (quizá las mejores técnicas de *machine learning* pueden encontrar una solución).
- Entornos cambiantes (un sistema de *machine learning* puede reentrenarse fácilmente con datos nuevos y mantenerse siempre actualizado).
- Obtener un entendimiento de problemas complejos y grandes cantidades de datos.

Ejemplos de aplicaciones

Vamos a ver algunos ejemplos concretos de tareas de *machine learning*, junto con las técnicas para abordarlas:

- **Analizar imágenes de productos en una cadena de producción para clasificarlas de forma automática:** Esto es una clasificación de imágenes, que suele realizarse utilizando redes neuronales convolucionales (RNC; consulta el capítulo 14) o algunos transformadores (consulta el capítulo 16).
- **Detectar tumores en escáneres cerebrales:** Esto es una segmentación semántica de imágenes, donde cada píxel en la imagen se clasifica (porque queremos determinar la ubicación y el tamaño exactos de los tumores), por lo general utilizando RNC o transformadores.

- **Clasificar artículos de noticias de manera automática:** Esto es procesamiento de lenguaje natural y, más concretamente, clasificación de texto, que puede abordarse utilizando redes neuronales recurrentes (RNR) y RNC, pero los transformadores funcionan aún mejor (consulta el capítulo 16).
- **Marcar de manera automática comentarios ofensivos en foros de discusión:** También es clasificación de texto y utiliza las mismas herramientas que el procesamiento de lenguaje natural.
- **Resumir documentos largos de forma automática:** Se trata de una rama del procesamiento de lenguaje natural denominada resumen de texto y utiliza las mismas herramientas.
- **Crear un chatbot o un asistente personal:** Esto implica muchos componentes del procesamiento de lenguaje natural, incluyendo la comprensión del lenguaje natural y módulos de preguntas-respuestas.
- **Predecir los ingresos de tu empresa para el año siguiente, en función de muchas métricas de rendimiento:** Se trata de una tarea de regresión (por ejemplo, predecir valores) que puede abordarse utilizando cualquier modelo de regresión, como uno de regresión lineal o uno de regresión polinomial (consulta el capítulo 4), una máquina de vector soporte de regresión (consulta el capítulo 5), un *random forest* de regresión (consulta el capítulo 7) o una red neuronal artificial (consulta el capítulo 10). Si quieres tener en cuenta secuencias de métricas de rendimiento anteriores, puede que te interese utilizar redes neuronales recurrentes, redes neuronales convolucionales o transformadores (consulta los capítulos 15 y 16).
- **Hacer que tu aplicación reaccione a órdenes de voz:** Esto es reconocimiento de discurso, que requiere el procesamiento de muestras de audio: puesto que se trata de secuencias largas y complejas, por lo general se procesan utilizando redes neuronales recurrentes, redes neuronales convolucionales o transformadores (consulta los capítulos 15 y 16).
- **Detectar fraudes con tarjetas de crédito:** Esto es una detección de anomalías, que puede abordarse utilizando *isolation forests*, modelos de mezclas gaussianas (véase el capítulo 9) o autocodificadores (véase el capítulo 17).
- **Segmentar clientes en función de sus compras para poder diseñar una estrategia de marketing diferente para cada segmento:** Esto es agrupamiento, que puede lograrse usando k-medias, DBSCAN y más (consulta el capítulo 9).
- **Representar un conjunto de datos de alta dimensión complejo en un diagrama claro y detallado:** Esto es visualización de datos, que a menudo implica técnicas de reducción de dimensionalidad (consulta el capítulo 8).
- **Recomendar un producto en el que un cliente pueda estar interesado, en función de compras anteriores:** Se trata de un sistema de recomendación. Un enfoque es introducir compras anteriores (u otra información sobre el cliente) en una red neuronal artificial

(consulta el capítulo 19) y hacer que genere como resultado la siguiente compra más probable. Esta red neuronal se entrenaría típicamente en secuencias de compras anteriores de todos los clientes.

- **Crear un bot inteligente para un juego:** Esto se aborda a menudo con aprendizaje por refuerzo (consulta el capítulo 18), que es una rama del *machine learning* que entrena agentes (como bots) para que elijan las acciones que maximizarán la recompensa con el tiempo (por ejemplo, un bot podría recibir una recompensa cada vez que el jugador pierda algunos puntos de vida), dentro de un entorno dado (como el juego). El famoso programa AlphaGo que derrotó al campeón mundial de go se creó utilizando aprendizaje por refuerzo.

La lista podría seguir, pero creo que esto ya te dará una idea de la increíble amplitud y la complejidad de las tareas que puede abordar el *machine learning* y los tipos de técnicas que se usarían para cada tarea.

Tipos de sistemas de *machine learning*

Hay tantos tipos diferentes de sistemas de *machine learning* que resulta útil clasificarlos en categorías amplias, según los siguientes criterios:

- Cómo se supervisan durante el entrenamiento (aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semisupervisado, aprendizaje autosupervisado y otros).
- Si pueden o no aprender de forma gradual sobre la marcha (aprendizaje *online* frente a aprendizaje por lotes).
- Si funcionan comparando simplemente puntos de datos nuevos con puntos de datos conocidos o si detectan patrones en los datos de entrenamiento y crean un modelo predictivo, como hacen los científicos (aprendizaje basado en instancias frente a aprendizaje basado en modelos).

Estos criterios no son exclusivos; podemos combinarlos como queramos. Por ejemplo, un filtro de *spam* de última generación puede aprender sobre la marcha utilizando un modelo de red neuronal profunda entrenado con ejemplos proporcionados por humanos de *spam* y *ham*; esto lo convierte en un sistema de aprendizaje supervisado *online* basado en modelos.

Vamos a fijarnos en cada uno de estos criterios con más detenimiento.

Supervisión del entrenamiento

Los sistemas de *machine learning* pueden clasificarse según la cantidad y el tipo de supervisión que tengan durante el entrenamiento. Hay muchas categorías, pero vamos a hablar de las más importantes: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje autosupervisado, aprendizaje semisupervisado y aprendizaje por refuerzo.

Aprendizaje supervisado

En el aprendizaje supervisado, el conjunto de entrenamiento que introducimos en el algoritmo incluye las soluciones deseadas, denominadas "etiquetas" (véase la figura 1.5).

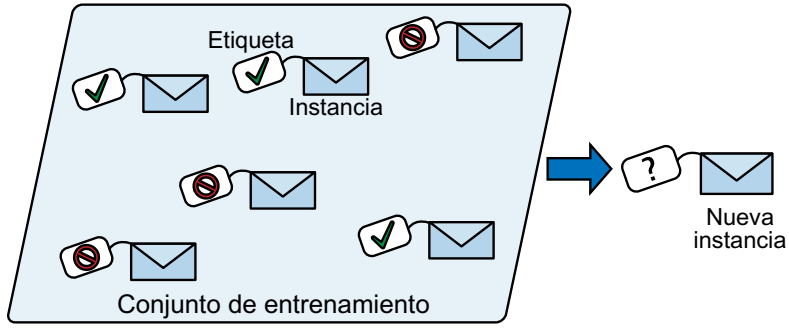


Figura 1.5. Un conjunto de entrenamiento etiquetado para la clasificación de *spam* (un ejemplo de aprendizaje supervisado).

Una tarea típica del aprendizaje supervisado es la clasificación. El filtro de *spam* es un buen ejemplo de esto: se entrena con muchos correos de ejemplo junto con su clase (*spam* o *ham*) y debe aprender a clasificar correos nuevos.

Otra tarea típica es predecir un valor numérico "objetivo", como el precio de un coche, si se da un conjunto de características (kilometraje, edad, marca, etc.). Este tipo de tarea se llama "regresión" (véase la figura 1.6).[1] Para entrenar al sistema, tenemos que darle muchos ejemplos de coche, incluyendo sus características y sus objetivos (es decir, sus precios).

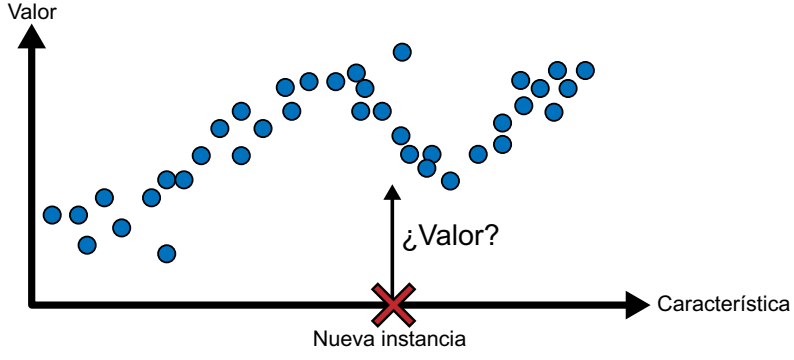


Figura 1.6. Un problema de regresión: predecir un valor cuando se da una característica de entrada (suele haber múltiples características de entrada y, a veces, múltiples valores de salida).

1. Dato curioso: este nombre que suena tan raro es un término estadístico introducido por Francis Galton mientras estudiaba el hecho de que los hijos de personas altas tienden a ser más bajos que sus progenitores. Puesto que los hijos eran más bajos, llamó a esto "regresión a la media". Este nombre se aplicó después a los métodos que utilizaba para analizar correlaciones entre variables.

Ten en cuenta que algunos modelos de regresión pueden usarse también para la clasificación y viceversa. Por ejemplo, la regresión logística suele utilizarse para la clasificación, ya que puede generar como salida un valor que se corresponde con la probabilidad de pertenecer a una clase dada (por ejemplo, un 20 % de probabilidades de ser *spam*).

> **Nota:** En general, las palabras "objetivo" y "etiqueta" se tratan como sinónimos en el aprendizaje supervisado, pero objetivo es más común en las tareas de regresión y etiqueta es más común en las tareas de clasificación. Además, las características se denominan a veces predictores o atributos. Estos términos pueden referirse a muestras individuales (por ejemplo, "la característica kilometraje de este coche es igual a 15.000") o a todas las muestras (por ejemplo, "la característica kilometraje tiene una fuerte correlación con el precio").

Aprendizaje no supervisado

En el aprendizaje no supervisado, como podrás suponer, los datos de entrenamiento no están etiquetados (véase la figura 1.7). El sistema intenta aprender sin profesor.

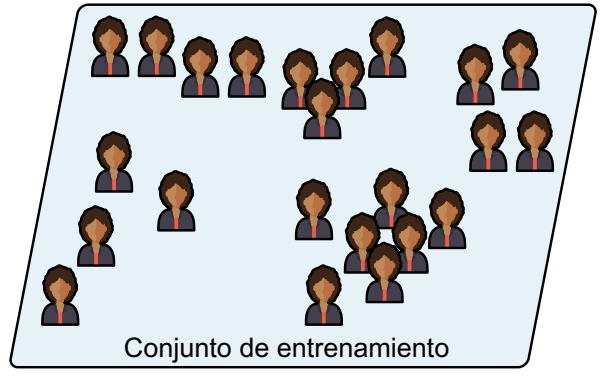


Figura 1.7. Un conjunto de entrenamiento sin etiquetar para aprendizaje no supervisado.

Por ejemplo, digamos que tenemos muchos datos sobre los visitantes de nuestro *blog*. Puede que queramos ejecutar un algoritmo de agrupamiento para intentar detectar grupos de visitantes similares (véase la figura 1.8). En ningún momento le decimos al algoritmo a qué grupo pertenece un visitante: encuentra esas conexiones sin nuestra ayuda. Por ejemplo, puede que se dé cuenta de que el 40 % de nuestros visitantes son adolescentes a los que les encantan los cómics y, por lo general, leen el *blog* después del instituto, mientras que el 20 % son adultos a los que les gusta la ciencia ficción que visitan el *blog* los fines de semana. Si utilizamos un algoritmo de agrupamiento jerárquico, puede que subdivida también cada grupo en grupos más pequeños. Esto puede ayudarnos a dirigir nuestras publicaciones para cada grupo.

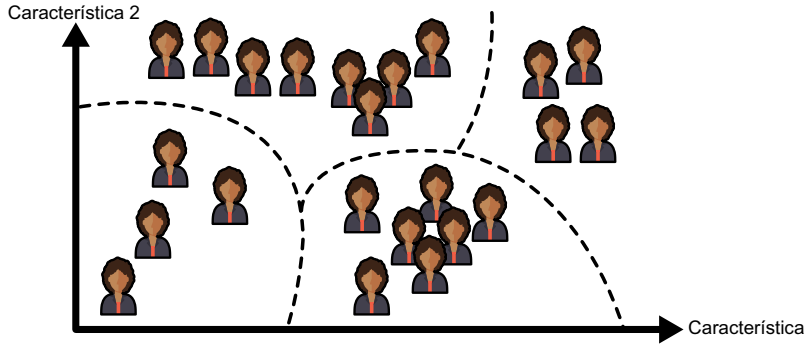


Figura 1.8. Agrupamiento.

Los algoritmos de visualización también son buenos ejemplos de aprendizaje no supervisado: les introducimos muchos datos complejos y sin etiquetar y generan representaciones 2D o 3D de los datos que pueden trazarse con facilidad (véase la figura 1.9). Estos algoritmos intentan preservar tanta estructura como sea posible (por ejemplo, intentando evitar que agrupamientos separados en el espacio de entrada se solapen en la visualización), de manera que podamos entender cómo se organizan los datos y, quizá, identificar patrones insospechados.

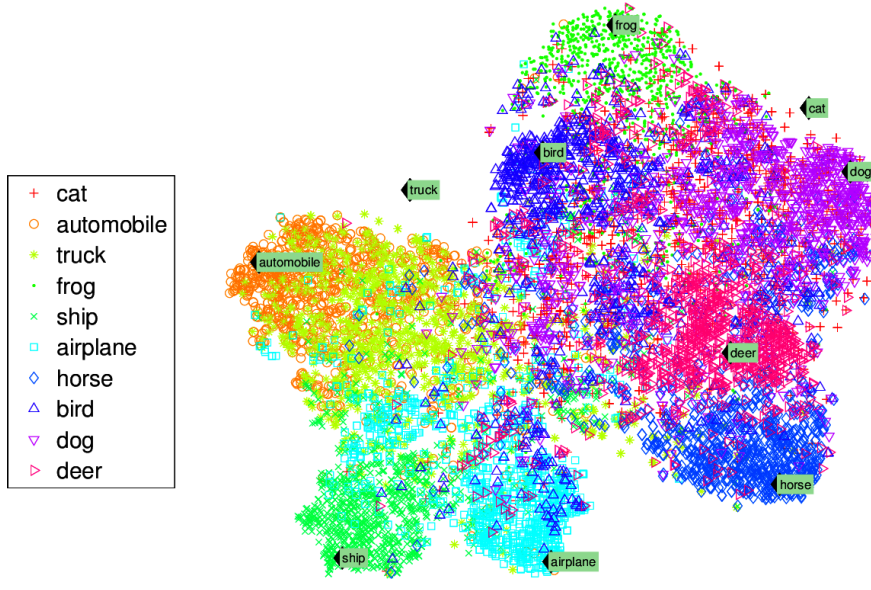


Figura 1.9. Ejemplo de una visualización t-SNE resaltando agrupamientos semánticos.[2]

2. Observa que los animales están bastante bien separados de los vehículos y que los caballos están cerca de los ciervos, pero lejos de los pájaros. Figura reproducida con permiso de Richard Socher *et al.*, "*Zero-Shot Learning Through Cross-Modal Transfer*", actas de la 26.ª *International Conference on Neural Information Processing Systems* 1 (2013): 935-943.

Una tarea relacionada es la reducción de la dimensionalidad, en la que el objetivo es simplificar los datos sin perder demasiada información. Una manera de hacerlo es fusionar varias características correlacionadas en una. Por ejemplo, el kilometraje de un coche puede estar estrechamente correlacionado con su edad, así que el algoritmo de reducción de la dimensionalidad combinará ambas cosas en una característica que represente el deterioro del coche. Esto se denomina "extracción de características".

> **Truco:** A menudo, es conveniente intentar reducir el número de dimensiones de los datos de entrenamiento utilizando un algoritmo de reducción de la dimensionalidad antes de introducirlos en otro algoritmo de *machine learning* (como un algoritmo de aprendizaje supervisado). Se ejecutará mucho más rápido, los datos ocuparán menos espacio de memoria en el disco y, en algunos casos, puede que el rendimiento sea mejor.

Otra tarea no supervisada importante es la detección de anomalías, por ejemplo, detectar transacciones inusuales en tarjetas de crédito para prevenir el fraude, encontrar defectos de fábrica o eliminar de manera automática valores atípicos de un conjunto de datos antes de introducirlo en otro algoritmo de aprendizaje. Se muestran al sistema instancias normales en su mayoría durante el entrenamiento, así que aprende a reconocerlas; después, cuando ve una nueva instancia, puede decir si parece una normal o si es probable que sea una anomalía (véase la figura 1.10). Una tarea muy similar es la detección de novedades: su objetivo es detectar instancias nuevas que parezcan diferentes de todas las instancias del conjunto de entrenamiento. Esto requiere tener un conjunto de entrenamiento muy "limpio", libre de cualquier instancia que queramos que el algoritmo detecte. Por ejemplo, si tenemos miles de imágenes de perros y un 1 % de esas imágenes representan chihuahuas, entonces un algoritmo de detección de novedades no debería tratar imágenes de chihuahuas nuevas como novedades. Por otro lado, los algoritmos de detección de anomalías podrían considerar que esos perros son tan raros y tan diferentes a otros perros que es probable que los clasifiquen como anomalías (sin ánimo de ofender a los chihuahuas).

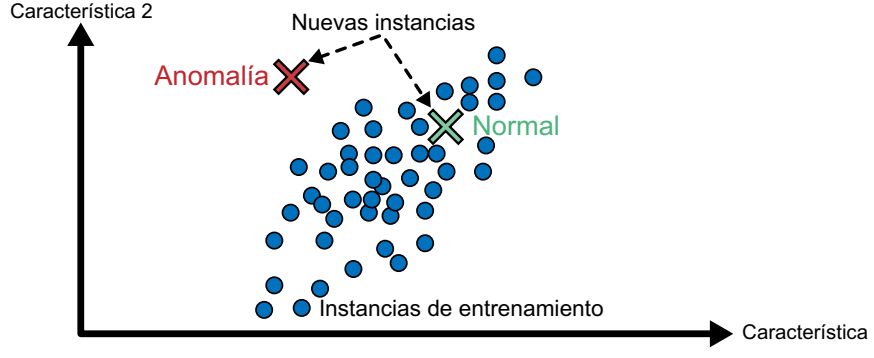


Figura 1.10. Detección de anomalías.

Por último, otra tarea no supervisada común es el aprendizaje de reglas de asociación, en el que el objetivo es explorar cantidades enormes de datos y descubrir relaciones interesantes entre los atributos. Por ejemplo, supongamos que tenemos un supermercado. Ejecutar una regla de asociación en nuestros registros de ventas podría revelar que la gente que compra salsa barbacoa y patatas fritas también tiende a comprar bistecs. Por tanto, quizá nos convenga colocar esos artículos cerca.

Aprendizaje semisupervisado

Puesto que etiquetar datos suele llevar mucho tiempo y supone un gasto importante, a menudo tendremos un montón de instancias sin etiquetar y unas pocas etiquetadas. Algunos algoritmos pueden ocuparse de datos que están etiquetados en parte. Esto se llama aprendizaje semisupervisado (véase la figura 1.11).

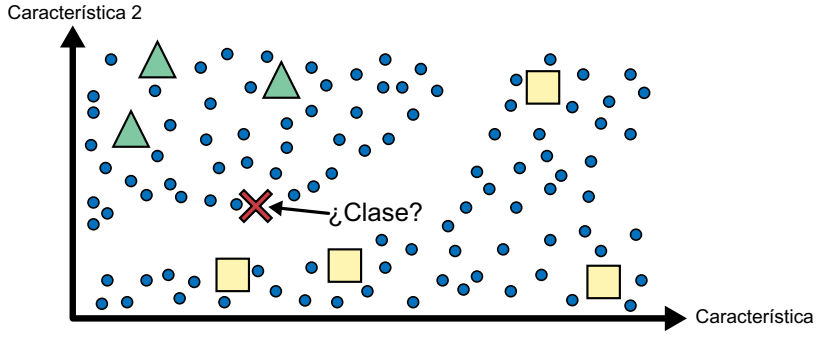


Figura 1.11. Aprendizaje semisupervisado con dos clases (triángulos y cuadrados): los ejemplos sin etiquetar (círculos) ayudan a clasificar una nueva instancia (la cruz) en la clase de los triángulos en vez de en la clase de los cuadrados, incluso aunque esté más cerca de los cuadrados etiquetados.

Algunos servicios de alojamiento de fotos, como Google Fotos, son buenos ejemplos de esto. Una vez que subimos todas nuestras fotos de familia al servicio, este reconoce de forma automática que la misma persona A aparece en las fotos 1, 5 y 11, mientras que otra persona B aparece en las fotos 2, 5 y 7. Esta es la parte no supervisada del algoritmo (agrupamiento). Ahora lo único que necesita el sistema es que le digamos quiénes son esas personas. Solo hay que añadir una etiqueta por persona[3] y él será capaz de poner nombre a todo el mundo en todas las fotos, lo cual resulta muy útil para buscar fotografías.

La mayoría de los algoritmos de aprendizaje semisupervisado son combinaciones de algoritmos no supervisados y supervisados. Por ejemplo, puede utilizarse un algoritmo de agrupamiento para agrupar instancias similares y, después, cada instancia no etiquetada puede etiquetarse con la etiqueta más común de su grupo. Una vez que se ha etiquetado el conjunto de datos completo, es posible utilizar cualquier algoritmo de aprendizaje supervisado.

3. Eso es cuando el sistema funciona a la perfección. En la práctica, a menudo crea varios agrupamientos por persona y, a veces, mezcla dos personas que se parecen, así que puede que tengamos que introducir varias etiquetas por persona y arreglar a mano algunos de los agrupamientos.

Aprendizaje autosupervisado

Otro enfoque al *machine learning* implica generar de verdad un conjunto de datos totalmente etiquetado a partir de uno totalmente sin etiquetar. De nuevo, una vez que el conjunto de datos completo está etiquetado, puede utilizarse cualquier algoritmo de aprendizaje supervisado. Este enfoque se denomina aprendizaje autosupervisado.

Por ejemplo, si tienes un conjunto de datos grande de imágenes sin etiquetar, puedes enmascarar de forma aleatoria una parte pequeña de cada imagen y, después, entrenar un modelo para recuperar la imagen original (véase la figura 1.12). Durante el entrenamiento, las imágenes enmascaradas se utilizan como las entradas del modelo y las imágenes originales se utilizan como las etiquetas.

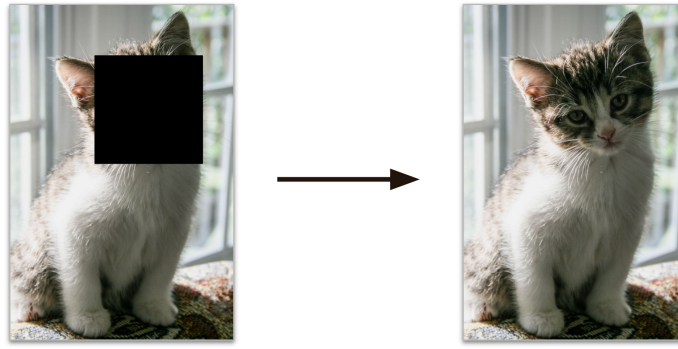


Figura 1.12. Ejemplo de aprendizaje autosupervisado: entrada (izquierda) y objetivo (derecha).

El modelo resultante puede ser bastante útil en sí mismo, por ejemplo, para reparar imágenes dañadas o eliminar objetos no deseados de fotografías. Pero, con mucha frecuencia, un modelo entrenado utilizando aprendizaje autosupervisado no es el objetivo final. Por lo general, lo que nos interesa es ajustar y perfeccionar el modelo para una tarea un poco diferente, una que nos importe de verdad.

Por ejemplo, supongamos que lo que queremos en realidad es un modelo de clasificación de mascotas: si le damos una imagen de cualquier animal, nos dirá a qué especie pertenece. Si tienes un conjunto de datos grande de fotos de mascotas sin etiquetar, puedes empezar por entrenar un modelo de reparación de imágenes usando aprendizaje autosupervisado. Una vez que logre un buen rendimiento, debería ser capaz de distinguir diferentes especies de mascotas: cuando repare una imagen de un gato cuya cara esté enmascarada, debe saber que no tiene que añadir la cara de un perro. Suponiendo que la arquitectura de tu modelo lo permita (y la mayoría de las arquitecturas de redes neuronales lo hacen), es posible ajustar después el modelo de manera que prediga especies de mascotas en vez de reparar imágenes. El paso final consiste en ajustar el modelo en un conjunto de datos etiquetado: el modelo ya sabe qué aspecto tienen gatos, perros y otras especies de mascotas, así que este paso solo es necesario para que el modelo pueda aprender la correspondencia entre las especies que ya conoce y las etiquetas que esperamos de él.

> **Nota:** Transferir el conocimiento de una tarea a otra se denomina aprendizaje por transferencia y es una de las técnicas del *machine learning* en la actualidad, sobre todo cuando se usan redes neuronales profundas (es decir, redes neuronales compuestas por muchas capas de neuronas). Lo veremos con detalle en la parte II.

Algunas personas consideran que el aprendizaje autosupervisado es una parte del aprendizaje no supervisado, ya que trabaja con conjuntos de datos completamente sin etiquetar. Pero el aprendizaje autosupervisado utiliza etiquetas (generadas) durante el entrenamiento, así que en ese aspecto se acerca más al aprendizaje supervisado. Y el término "aprendizaje no supervisado" se usa por lo general cuando se trata con tareas como el agrupamiento, la reducción de la dimensionalidad o la detección de anomalías, mientras que el aprendizaje autosupervisado se centra en las mismas tareas que el aprendizaje supervisado, principalmente la clasificación y la regresión. En resumen, es mejor tratar el aprendizaje autosupervisado como una categoría propia.

Aprendizaje por refuerzo

El aprendizaje por refuerzo es algo totalmente diferente. El sistema de aprendizaje, que en este contexto se denomina "agente", puede observar el entorno, seleccionar y realizar acciones y recibir "recompensas" a cambio (o castigos en forma de recompensas negativas, como se muestra en la figura 1.13). Debe aprender por sí mismo cuál es la mejor estrategia, denominada "política", para obtener la mayor recompensa con el tiempo. Una política define qué acción debería elegir el agente cuando se encuentra en una situación determinada.

Por ejemplo, muchos robots implementan aprendizaje por refuerzo para aprender a andar. El programa AlphaGo de DeepMind también es un buen ejemplo de aprendizaje por refuerzo: saltó a los titulares en mayo de 2017 cuando derrotó a Ke Jie, el número uno del mundo en aquel momento, en una partida de go. Aprendió su política ganadora analizando millones de partidas y, después, jugó muchas partidas contra sí mismo. Ten en cuenta que el aprendizaje estaba desactivado durante las partidas contra el campeón; AlphaGo solo estaba aplicando la política que había aprendido. Como veremos en la siguiente sección, esto se denomina aprendizaje *offline*.

Aprendizaje por lotes frente a aprendizaje *online*

Otro criterio utilizado para clasificar sistemas de *machine learning* es si el sistema puede o no aprender de manera gradual a partir de un flujo de datos entrantes.

Aprendizaje por lotes

En el aprendizaje por lotes, el sistema es incapaz de aprender de manera gradual: debe entrenarse utilizando todos los datos disponibles. Por lo general, esto requiere mucho tiempo y muchos recursos computacionales, así que suele hacerse *offline*. Primero se entrena el sistema

y, después, se lanza a producción y se ejecuta sin seguir aprendiendo; solo aplica lo que ha aprendido. Esto se llama aprendizaje *offline*. Por desgracia, el rendimiento de un modelo tiende a deteriorarse con el tiempo poco a poco, simplemente porque el mundo sigue evolucionando mientras el modelo permanece inalterado. Este fenómeno se conoce como *data drift* o deriva de datos. La solución es reentrenar con regularidad el modelo con datos actualizados. La frecuencia con la que se necesita hacerlo depende del caso de uso: si el modelo clasifica imágenes de gatos y perros, su rendimiento se deteriorará muy despacio, pero, si trata con sistemas de evolución rápida, por ejemplo, realizando predicciones sobre el mercado financiero, es probable que se deteriore bastante deprisa.

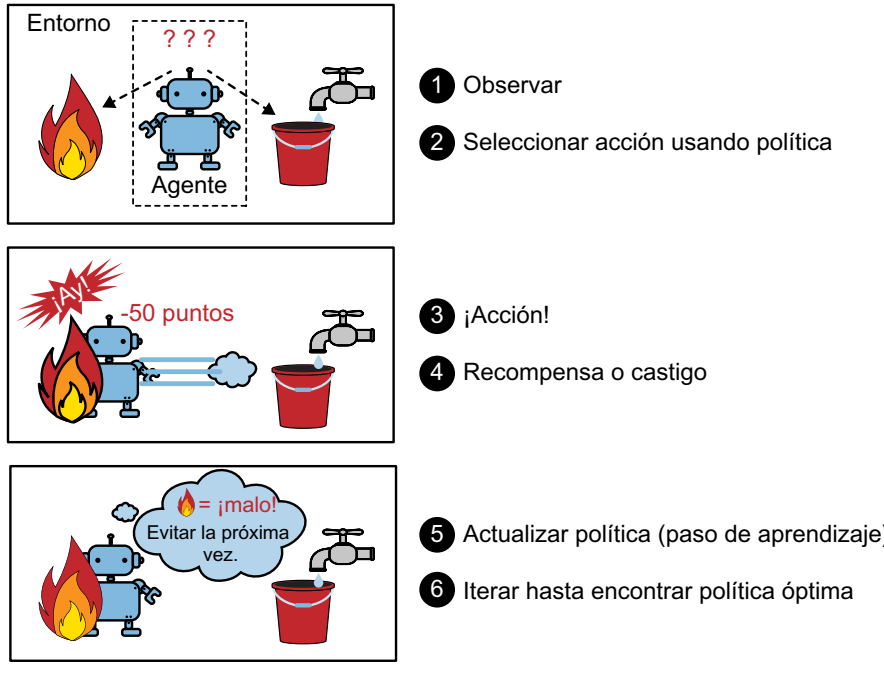


Figura 1.13. Aprendizaje por refuerzo.

> **Advertencia:** Incluso un modelo entrenado para clasificar imágenes de gatos y perros puede necesitar reentrenarse con regularidad, no porque los gatos y los perros muten de la noche a la mañana, sino porque las cámaras siguen cambiando, junto con los formatos de imagen, la nitidez, el brillo y las proporciones. Además, puede que a la gente le encante una raza diferente el año que viene o que decidan poner gorritos a sus mascotas, ¿quién sabe?

Si queremos que un sistema de aprendizaje por lotes conozca datos nuevos (como un tipo nuevo de *spam*), tenemos que entrenar una versión nueva del sistema desde cero en el conjunto de datos completo (no solo los datos nuevos, sino también los viejos) y, después, sustituirlo

por el nuevo. Por suerte, el proceso completo de entrenamiento, evaluación y lanzamiento de un sistema de *machine learning* puede automatizarse con bastante facilidad (como mostraba la figura 1.3), así que incluso un sistema de aprendizaje por lotes puede adaptarse al cambio. Solo tenemos que actualizar los datos y entrenar una nueva versión del sistema desde cero con la frecuencia que sea necesaria.

La solución es sencilla y a menudo funciona bien, pero entrenar utilizando el conjunto de datos completo puede llevar varias horas, así que, por lo general, entrenaríamos un sistema nuevo solo cada 24 horas o incluso una vez a la semana. Si el sistema necesita adaptarse a datos que cambian con rapidez (por ejemplo, para predecir precios de acciones), entonces necesitamos una solución más reactiva.

Además, entrenar en el conjunto de datos completo requiere muchos recursos computacionales (CPU, espacio de memoria, espacio en el disco, E/S de disco, E/S de red, etc.). Si tenemos muchos datos y automatizamos el sistema para entrenar desde cero todos los días, acabará costándonos mucho dinero. Si la cantidad de datos es enorme, puede incluso que sea imposible utilizar un algoritmo de aprendizaje por lotes.

Por último, si nuestro sistema necesita ser capaz de aprender de manera autónoma y tiene recursos limitados (por ejemplo, una aplicación para *smartphone* o un *rover* en Marte), llevar a todas partes grandes cantidades de datos de entrenamiento y destinar muchos recursos a entrenar varias horas cada día es un defecto crítico.

Una opción mejor en todos estos casos es utilizar algoritmos que sean capaces de aprender de manera gradual.

Aprendizaje *online*

En el aprendizaje *online*, entrenamos el sistema de datos de forma gradual al introducir instancias de datos de manera secuencial, ya sea individualmente o en grupos pequeños llamados minilotes. Cada paso del aprendizaje es rápido y barato, así que el sistema puede aprender acerca de datos nuevos sobre la marcha, según llegan (véase la figura 1.14).

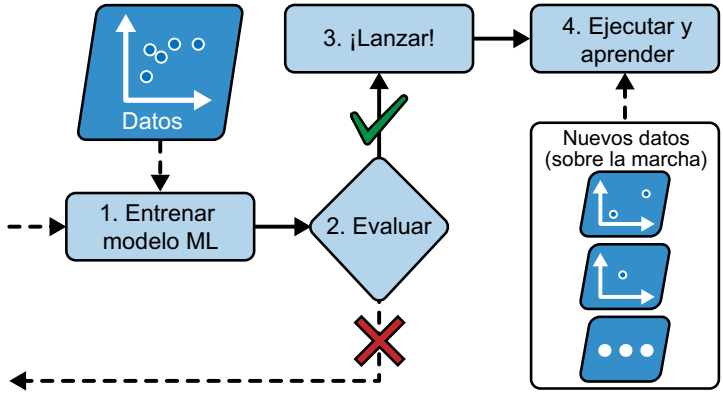


Figura 1.14. En el aprendizaje *online*, un modelo se entrena y se lanza a producción y, después, sigue aprendiendo a medida que entran nuevos datos.

El aprendizaje *online* es útil para sistemas que necesitan adaptarse al cambio con mucha rapidez (por ejemplo, para detectar patrones nuevos en el mercado bursátil). También es una buena opción si tenemos recursos computacionales limitados; por ejemplo, si el modelo se entrena en un dispositivo móvil.

Además, los algoritmos de aprendizaje *online* pueden utilizarse para entrenar sistemas en conjuntos de datos enormes que no caben en la memoria principal de una máquina (esto se llama aprendizaje *out of core* o "fuera del núcleo"). El algoritmo carga parte de los datos, ejecuta un paso de entrenamiento en esos datos y repite el proceso hasta que se ha ejecutado en todos los datos (véase la figura 1.15).

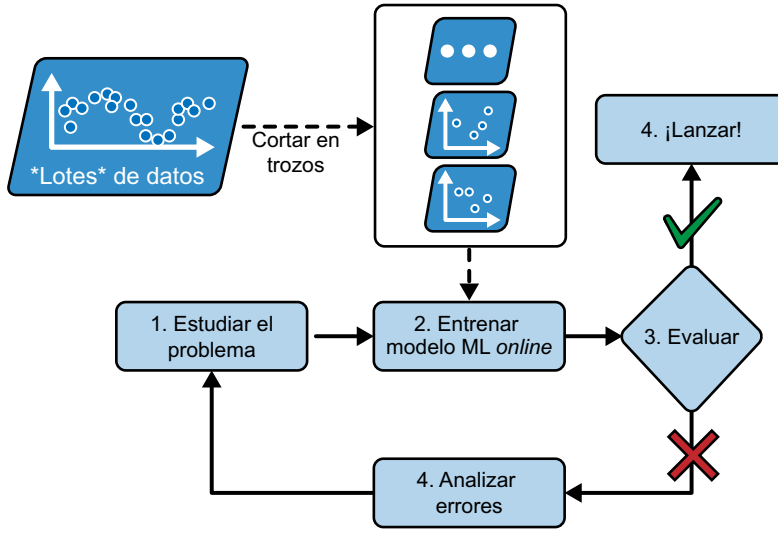


Figura 1.15. Utilizar aprendizaje *online* para manejar conjuntos de datos enormes.

Un parámetro importante de los sistemas de aprendizaje *online* es lo rápido que deberían adaptarse a los datos cambiantes: esto se denomina "tasa de aprendizaje". Si establecemos una tasa de aprendizaje alta, entonces el sistema se adaptará con rapidez a datos nuevos, pero también tenderá a olvidar rápido los datos antiguos (y no nos interesa que un filtro de *spam* marque solo los últimos tipos de *spam* que se le han enseñado). Por el contrario, si establecemos una tasa de aprendizaje baja, el sistema tendrá más inercia, es decir, aprenderá más despacio, pero también será menos sensible al ruido de los datos nuevos o a las secuencias de puntos de datos no representativos (valores atípicos).

> **Advertencia:** El aprendizaje *out of core* suele hacerse *offline* (es decir, no en el sistema vivo), así que el nombre "aprendizaje *online*" puede resultar confuso. Piense en ello como "aprendizaje gradual".

Un gran reto con el aprendizaje *online* es que, si se introducen en el sistema datos malos, el rendimiento de este disminuirá, posiblemente con rapidez (dependiendo de la calidad de los datos y la tasa de aprendizaje). Si es un sistema vivo, los clientes lo notarán. Por ejemplo, los datos malos podrían provenir de un fallo (como un sensor que no funciona en un robot) o de alguien que está intentando manipular el sistema (por ejemplo, llenando de *spam* un motor de búsqueda para aparecer más arriba en los resultados de la búsqueda). Para reducir este riesgo, tenemos que monitorizar el sistema con atención y desactivar el aprendizaje de inmediato (y posiblemente revertirlo a un estado anterior que funcionase) si detectamos un descenso del rendimiento. También nos conviene monitorizar los datos de entrada y reaccionar ante los datos anormales; por ejemplo, utilizando un algoritmo de detección de anomalías (consulta el capítulo 9).

Aprendizaje basado en instancias frente a aprendizaje basado en modelos

Otra manera de categorizar los sistemas de *machine learning* es fijarse en cómo generalizan. La mayoría de las tareas de *machine learning* tienen que ver con hacer predicciones. Eso significa que, si se le da a un sistema una cantidad de ejemplos de entrenamiento, este tiene que ser capaz de hacer buenas predicciones para (generalizar a) ejemplos que nunca ha visto antes. Tener una buena medida del rendimiento en los datos de entrenamiento está bien, pero no es suficiente; el verdadero objetivo es tener un buen rendimiento en instancias nuevas.

Hay dos enfoques principales para la generalización: el aprendizaje basado en instancias y el aprendizaje basado en modelos.

Aprendizaje basado en instancias

Es posible que la forma más trivial de aprendizaje sea aprender de memoria, sin más. Si creásemos un filtro de *spam* de esa manera, este simplemente marcaría todos los correos que fuesen idénticos a correos que ya han marcado los usuarios; no es la peor solución, pero, desde luego, tampoco es la mejor.

En vez de marcar solo correos que sean idénticos a correos *spam* conocidos, el filtro podría programarse para marcar también correos que sean muy similares a correos *spam* conocidos Esto requiere una medida de la similitud entre dos correos. Una (muy básica) podría ser contar el número de palabras que tienen en común. El sistema marcaría un correo como *spam* si tiene muchas palabras en común con un correo *spam* conocido.

Esto se denomina "aprendizaje basado en instancias": el sistema aprende los ejemplos de memoria y, después, generaliza a nuevos casos utilizando una medida de similitud para compararlos con los ejemplos aprendidos (o un subconjunto de ellos). Por ejemplo, en la figura 1.16, la nueva instancia se clasificaría como un triángulo porque la mayoría de las instancias más similares pertenecen a esa clase.

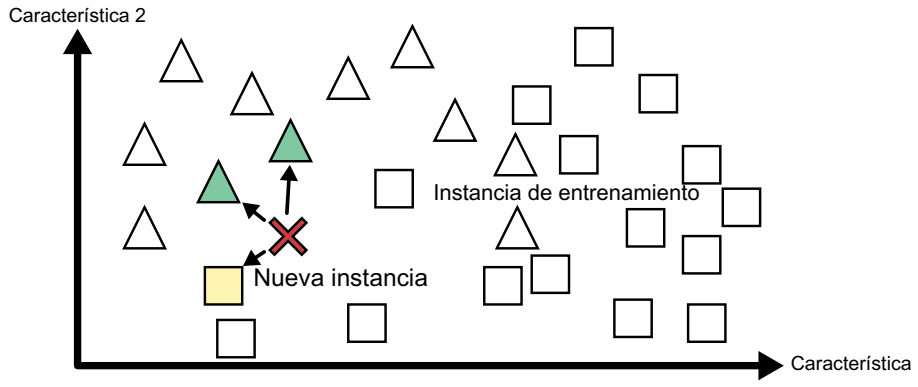


Figura 1.16. Aprendizaje basado en instancias.

Aprendizaje basado en modelos y un flujo de trabajo típico de *machine learning*

Otra forma de generalizar a partir de un conjunto de ejemplos es crear un modelo de esos ejemplos y, después, utilizarlo para hacer predicciones. Esto se denomina "aprendizaje basado en modelos" (véase la figura 1.17).

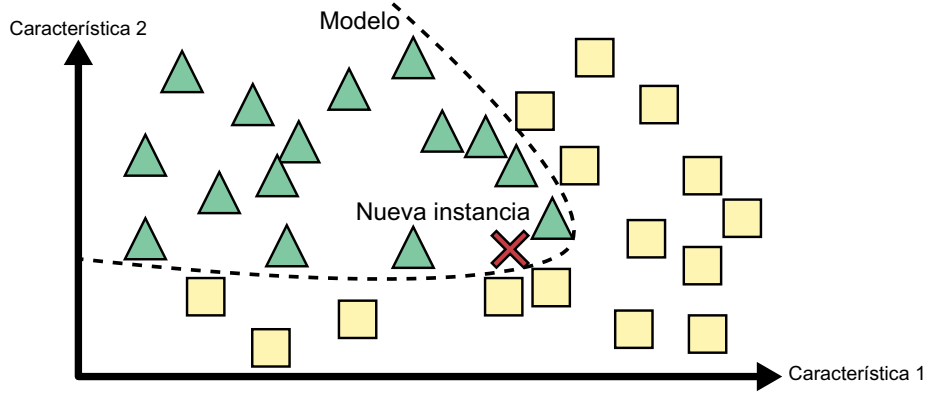


Figura 1.17. Aprendizaje basado en modelos.

Por ejemplo, supongamos que queremos saber si el dinero hace más feliz a la gente, así que descargamos los datos del Índice para una Vida Mejor del sitio web de la OCDE (`https://www.oecdbetterlifeindex.org`) y las estadísticas del Banco Mundial (`https://ourworldindata.org`) sobre el producto interior bruto (PIB) per cápita. A continuación, unimos las dos tablas y ordenamos por PIB per cápita. La tabla 1.1 muestra un fragmento del resultado.

Tabla 1.1. ¿El dinero hace más feliz a la gente?

País	PIB per cápita (dólares EE. UU.)	Satisfacción vital
Turquía	28.384	5,5
Hungría	31.008	5,6
Francia	42.026	6,5
Estados Unidos	60.236	6,9
Nueva Zelanda	42.404	7,3
Australia	48.698	7,3
Dinamarca	55.938	7,6

Vamos a trazar los datos para estos países (véase la figura 1.18).

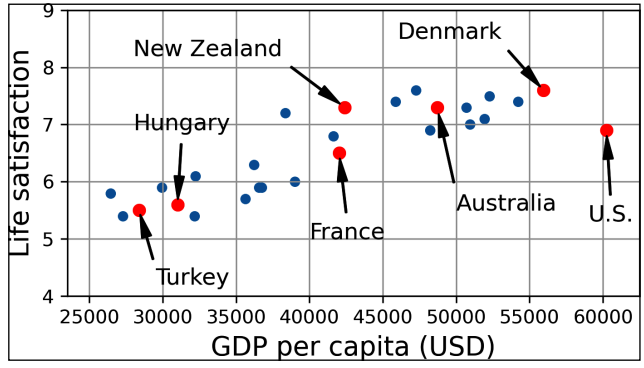


Figura 1.18. ¿Ves una tendencia aquí?

¡Parece haber una tendencia! Aunque los datos son ruidosos (es decir, son aleatorios en parte), parece que la satisfacción vital sube más o menos de forma lineal a medida que aumenta el PIB per cápita del país. Por tanto, decidimos que el modelo de satisfacción vital sea una función lineal del PIB per cápita. Este paso se llama selección de modelo: hemos seleccionado un modelo lineal de satisfacción vital con solo un atributo, PIB per cápita (véase la ecuación 1.1).

Ecuación 1.1. Un modelo lineal simple.

$$\text{satisfacción_vital} = \theta_0 + \theta_1 \times \text{PIB_per_cápita}$$

Este modelo tiene dos parámetros de modelo, θ_0 y θ_1.[4] Al ajustar estos parámetros, podemos hacer que nuestro modelo represente cualquier función lineal, como muestra la figura 1.19.

4. Por convención, la letra griega θ (zeta) se utiliza con frecuencia para representar parámetros de modelos.