O'REILLY®

DỰA THEO ẤN BẢN LẦN THỨ HAI

Thực hành Học Máy với Scikit-Learn, Keras & TensorFlow

Khái niệm, Công cụ & Kỹ thuật Xây dựng những Hệ thống Thông minh

TẬP 1

Kiến thức Nền tảng về Học Máy

jupyter

Aurélien Géron

NHÓM DỊCH THUẬT MACHINE LEARNING CƠ BẢN

FUNiX Learn with Mentors

NXB THÔNG TIN VÀ TRUYỀN THÔNG

DỰA THEO ẤN BẢN LẦN THỨ HAI

Thực hành Học Máy với Scikit-Learn, Keras, và TensorFlow

Khái niệm, Công cụ & Kỹ thuật Xây dựng những Hệ thống Thông minh

Tập 1: Kiến thức Nền tảng về Học Máy

Aurélien Géron

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thực hành Học Máy với Scikit-Learn, Keras, và TensorFlow
Tập 1 - Kiến thức Nền tảng về Học Máy
Aurélien Géron

Copyright © 2019 Kiwisoft S.A.S. All rights reserved.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Vietnamese © 2021 by FUNiX & MLBVN.

Cuốn sách được xuất bản theo hợp đồng bản quyền giữa O'Reilly Media và Tổ chức Giáo dục trực tuyến FUNiX.

Toàn bộ bản quyền liên quan đến xuất bản phẩm này đã được đăng ký bảo hộ. Không phần nào trong xuất bản phẩm này được sao chép, phân phối hoặc chia sẻ lại dưới bất cứ hình thức nào (như ấn phẩm điện tử hoặc sách giấy) hay phương tiện nào, hoặc được lưu trữ trong bất kỳ cơ sở dữ liệu hay hệ thống truy cập nào mà không có sự cho phép bằng văn bản của Tổ chức Giáo dục trực tuyến FUNiX và MLBVN.

Liên hệ góp ý hoặc hợp tác: *handson-ml@mlbvn.org*

Lời tựa

Học Máy là một lĩnh vực đã xuất hiện từ những năm 1960, tuy nhiên nó chỉ thật sự nổi lên trong khoảng một thập kỷ vì những phát triển mang tính đột phá trong Học Sâu. Nhờ những phát triển về phần cứng tính toán và lượng dữ liệu khổng lồ được sinh ra bởi hàng tỷ người dùng trên Internet, Học Máy đã mang lại những ứng dụng hữu ích không chỉ trên Internet mà còn len lỏi vào cuộc sống thường nhật.

Hiện nay, Học Máy đã trở thành môn học chính thức trong nhiều chương trình Đại học tại Việt Nam bởi nhu cầu dạy và học về lĩnh vực này ngày một cao nhằm đáp ứng thị trường lao động đang cần nhân sự có chuyên môn làm việc trong Doanh nghiệp của mình. Tuy nhiên, lượng tài liệu tiếng Việt hiện nay là rất hạn chế.

Nhóm dịch thuật Machine Learning Cơ Bản (MLBVN) đã được thành lập sau đó với mong muốn đem lại nguồn tài liệu tiếng Việt có chất lượng từ những cuốn sách kinh điển về chủ đề Trí tuệ Nhân tạo, Học Máy,... trên khắp thế giới nhằm góp một phần nhỏ của mình vào sự phát triển chung của Cộng đồng Học tập & Nghiên cứu Trí tuệ Nhân tạo đang ngày một phát triển tại Việt Nam.

Nhóm chúng tôi làm việc trên tinh thần không vì lợi nhuận, những thành viên tham gia chuyển ngữ là những Nhà khoa học, Tiến sĩ, Thạc sĩ, Kỹ sư đang học tập và công tác tại nhiều nước trên thế giới, từng tham gia vào những dự án chuyển ngữ trước đó với nhiều sự quan tâm, kinh nghiệm, và chuyên môn trong ngành đã ngồi lại, kết nối và cùng nhau làm những điều nhỏ.

Trong hai năm qua, chúng tôi cùng những thành viên Cộng đồng Machine Learning Cơ Bản đã cùng nhau hoàn thiện hai cuốn sách là Khát khao Học Máy, Đắm mình vào Học Sâu, và nhận được những phản hồi tích cực từ cộng đồng. Tiếp nối với những thành công đó, chúng tôi quyết định thực hiện chuyển ngữ cuốn sách "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" của tác giả Aurélien Géron đã được cập nhật, tái bản lần thứ nhất vào năm 2019, và được xuất bản bởi O'Reilly Media.

Đây là một trong những tài liệu mới mẻ mang lại đầy đủ kiến thức cơ bản cũng như ví dụ thực hành cho cả Học Máy truyền thống lẫn Học Sâu hiện đại. Cuốn sách dẫn giải các thuật toán qua những ví dụ cụ thể với mã nguồn chi tiết đi kèm. Đặc biệt, mã nguồn của cuốn sách được viết trên ba thư viện Học Máy và Học Sâu phổ biến nhất, điều này cung cấp cho người đọc không chỉ là lý thuyết về các mô hình mà còn là cách để triển khai vào những bài toán thực tế. Chúng tôi tin rằng cuốn sách này rất hữu ích để độc giả có thể thực hành theo.

Để có được sự suôn sẻ trong quá trình thực hiện, chúng tôi xin gửi lời cảm ơn đến FUNiX - Tổ chức giáo dục trực tuyến và nền tảng Mentoring hỗ trợ người học công nghệ trực thuộc Tập đoàn FPT, đã đồng hành và giúp đỡ chúng tôi trong suốt quá trình hoàn thiện cuốn sách.

Đặc biệt, tôi xin gửi lời tri ân đến những thành viên đã trực tiếp tham gia vào dự án này, bao gồm Đoàn Võ Duy Thanh, Nguyễn Cảnh Thướng, Lê Khắc Hồng Phúc, Nguyễn Lê Quang Nhật, Nguyễn Thanh Hòa, Nguyễn Văn Quang, Nguyễn Văn Cường, Phạm Minh Đức, và Phạm Hồng Vinh. Cùng nhau đi qua những đề tài, dự án và chúng ta đã khẳng định với nhau sự hiện diện rất quan trọng của mình đối với nhóm. Xin cảm ơn sự nỗ lực, nhiệt thành, và trách nhiệm của

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

từng cá nhân cùng thời gian quý báu đã dành cho dự án; chúng ta sẽ không thể hoàn thành cuốn sách này một cách trọn vẹn và chất lượng nhất nếu thiếu đi bất kỳ một yếu tố nào cũng như thành viên nào được nêu trên.

Độc giả thân mến, thành quả của chúng tôi ngày hôm nay đang được bạn tham khảo, chúng tôi hy vọng mình đã làm tốt và cung cấp cho bạn được những kiến thức cần thiết.

Mặc dù rất cẩn trọng và cố gắng trong việc chuyển ngữ, hiệu đính; ấn bản này có thể sẽ không tránh khỏi những thiếu sót. Vì vậy, chúng tôi rất mong nhận được ý kiến của độc giả để có thể hoàn thiện hơn trong các ấn bản sau.

Mọi sự góp ý chi tiết về cuốn sách xin quý độc giả gửi về hộp thư *handson-ml@mlbvn.org*, và những thảo luận cần thiết với cuốn sách xin bạn truy cập đường dẫn *https://git.io/JnxCe* (khuyến khích) để thảo luận thêm cùng chúng tôi và những độc giả khác.

Thay mặt Nhóm dịch thuật,
Vũ Hữu Tiệp

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Mục lục

Lời nói đầu **x**

Chương 1. Toàn cảnh Học Máy **1**
Học Máy là gì? . . . 2
Tại sao lại dùng Học Máy? . . . 2
Các Ứng dụng Tiêu biểu . . . 5
Các kiểu Hệ thống Học Máy . . . 6
Học có Giám sát/không Giám sát . . . 7
Học theo Batch và Học Trực tuyến . . . 13
Học dựa trên Mẫu và dựa trên Mô hình . . . 15
Những Thách thức Chính của Học Máy . . . 21
Không đủ Dữ liệu Huấn luyện . . . 21
Dữ liệu Huấn luyện Không mang tính Đại diện . . . 22
Dữ liệu Kém Chất lượng . . . 23
Các Đặc trưng Không liên quan . . . 24
Quá khớp Dữ liệu Huấn luyện . . . 24
Dưới khớp Dữ liệu Huấn luyện . . . 26
Ôn tập . . . 26
Kiểm tra và Đánh giá . . . 27
Tinh Chỉnh Siêu Tham Số và Lựa Chọn Mô Hình . . . 27
Dữ liệu không tương đồng . . . 28
Bài tập . . . 29

Chương 2. Dự án Học Máy từ Đầu tới Cuối **31**
Làm việc với Dữ liệu Thực . . . 31
Nhìn vào Bức tranh Tổng thể . . . 32
Phát biểu Bài toán . . . 33
Lựa chọn Phép đo Chất lượng . . . 34
Kiểm tra các Giả định . . . 36
Thu thập Dữ liệu . . . 37
Khởi tạo Môi trường Làm việc . . . 37
Tải Dữ liệu . . . 40
Nhìn qua Cấu trúc Dữ liệu . . . 41
Tạo Tập Kiểm tra . . . 45
Khám phá và trực quan hóa để hiểu Dữ liệu . . . 49
Trực quan hóa Dữ liệu Địa lý . . . 49
Tìm sự Tương quan . . . 51
Thí nghiệm Kết hợp các Thuộc tính . . . 54
Chuẩn bị Dữ liệu cho các Thuật toán Học Máy . . . 55
Làm sạch Dữ liệu . . . 55
Xử lý các Thuộc tính Văn bản và Hạng mục . . . 57
Bộ Biến đổi Tùy chỉnh . . . 60

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Co giãn Đặc trưng . . . 61
Pipeline Biến đổi . . . 61
Chọn và Huấn luyện Mô hình . . . 63
Huấn luyện và Đánh giá trên tập Huấn luyện . . . 63
Kiểm định Chéo: Phương pháp Đánh giá tốt hơn . . . 64
Tinh chỉnh Mô hình . . . 66
Tìm kiếm dạng Lưới . . . 67
Tìm kiếm Ngẫu nhiên . . . 69
Phương pháp Ensemble . . . 69
Phân tích các Mô hình Tốt nhất và Lỗi của Chúng . . . 69
Đánh giá Hệ thống trên Tập Kiểm tra . . . 70
Triển khai, Theo dõi, và Bảo trì Hệ thống . . . 71
Hãy Thực Hành! . . . 74
Bài tập . . . 74

Chương 3. Bài toán Phân loại **75**
MNIST . . . 75
Huấn luyện một Bộ Phân loại Nhị phân . . . 77
Phép đo Chất lượng . . . 78
Đánh giá Độ Chính xác bằng Kiểm định chéo . . . 78
Ma trận Nhầm lẫn . . . 80
Precision và Recall . . . 81
Đánh đổi Precision/Recall . . . 82
Đường cong ROC . . . 86
Phân loại Đa lớp . . . 89
Phân tích Lỗi . . . 91
Phân loại Đa nhãn . . . 94
Phân loại Đa Đầu ra . . . 95
Bài tập . . . 96

Chương 4. Huấn luyện Mô hình **98**
Hồi quy Tuyến tính . . . 99
Phương trình Pháp tuyến . . . 100
Độ phức tạp Tính toán . . . 103
Hạ Gradient . . . 104
Hạ Gradient theo Batch . . . 107
Hạ Gradient Ngẫu nhiên . . . 109
Hạ Gradient theo Mini-batch . . . 112
Hồi quy Đa thức . . . 114
Đồ thị Quá trình học . . . 115
Mô hình Tuyến tính Điều chuẩn . . . 119
Hồi quy Ridge . . . 119
Hồi quy Lasso . . . 122
Elastic Net . . . 124
Phương pháp Dừng Sớm . . . 124
Hồi quy Logistic . . . 126
Ước lượng Xác suất . . . 126
Huấn luyện và Hàm Chi phí . . . 127
Ranh giới Quyết định . . . 128
Hồi quy Softmax . . . 131
Bài tập . . . 134

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 5. Máy Vector Hỗ trợ **136**
Phân loại với SVM Tuyến tính . . . 136
Phân loại Biên Mềm . . . 137
Phân loại SVM Phi Tuyến . . . 139
Hạt nhân Đa thức . . . 140
Đặc trưng Tương tự . . . 141
Hạt nhân Gaussian RBF . . . 142
Độ phức tạp Tính toán . . . 143
Hồi quy SVM . . . 144
Giải thích Mô hình . . . 146
Hàm Quyết định và Dự đoán . . . 146
Hàm Mục tiêu . . . 147
Quy hoạch Toàn phương . . . 148
Bài toán Đối ngẫu . . . 149
SVM Hạt nhân . . . 150
Bộ Phân loại SVM Trực tuyến . . . 152
Bài tập . . . 153

Chương 6. Cây Quyết định **154**
Huấn luyện và Biểu diễn một Cây Quyết định . . . 154
Dự đoán . . . 155
Ước lượng Xác suất các Lớp . . . 157
Thuật toán Huấn luyện CART . . . 158
Độ phức tạp Tính toán . . . 158
Độ Pha tạp Gini hay Entropy? . . . 159
Các Siêu tham số Điều chuẩn . . . 159
Hồi quy . . . 160
Sự Bất ổn . . . 162
Bài tập . . . 163

Chương 7. Học Ensemble và Rừng Ngẫu nhiên **165**
Bộ phân loại Biểu quyết . . . 165
Bagging và Pasting . . . 168
Bagging và Pasting trong Scikit-Learn . . . 169
Đánh giá Out-of-Bag . . . 170
Patch Ngẫu nhiên và Không gian con Ngẫu nhiên . . . 171
Rừng Ngẫu nhiên . . . 172
Cây Siêu Ngẫu nhiên . . . 172
Độ Quan trọng của Đặc trưng . . . 173
Boosting . . . 174
AdaBoost . . . 174
Gradient Boosting . . . 177
Stacking . . . 182
Bài tập . . . 185

Chương 8. Giảm Chiều **186**
Lời nguyền Chiều . . . 187
Các Phương pháp Giảm Chiều chính . . . 188
Phép chiếu . . . 188
Học Đa tạp . . . 190
PCA . . . 191
Bảo toàn Phương sai . . . 191

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thành phần Chính . . . 192
Chiếu Xuống d Chiều . . . 193
Sử dụng Scikit-Learn . . . 194
Tỉ lệ Phương sai được Giải thích . . . 194
Chọn Số chiều Hợp lý . . . 194
PCA cho Tác vụ Nén . . . 195
PCA Ngẫu nhiên . . . 196
PCA Gia tăng . . . 197
PCA Hạt nhân . . . 197
Lựa chọn Hạt nhân và Tinh chỉnh Siêu tham số . . . 198
LLE . . . 200
Các Kỹ thuật Giảm chiều Khác . . . 202
Bài tập . . . 203

Chương 9. Các kỹ thuật Học Không giám sát **205**
Phân cụm . . . 206
K-Điểm trung bình . . . 208
Hạn chế của K-Điểm trung bình . . . 217
Sử dụng Phân cụm để Phân vùng Ảnh . . . 218
Sử dụng Phân cụm để Tiền xử lý . . . 220
Ứng dụng Phân Cụm cho Học Bán Giám sát . . . 221
DBSCAN . . . 224
Các Thuật toán Phân cụm khác . . . 226
Hỗn hợp Gauss . . . 227
Phát hiện Bất thường Sử dụng Hỗn hợp Gauss . . . 232
Chọn Số Cụm . . . 234
Mô hình Hỗn hợp Bayes Gauss . . . 236
Các Thuật toán Phát hiện Bất thường và Tính mới . . . 240
Bài tập . . . 241

Phụ lục A. Lời giải các Bài tập **243**

Phụ lục B. Danh mục Công việc trong Dự án Học Máy **253**

Phụ lục C. Bài toán Đối ngẫu SVM **258**

Tài liệu tham khảo **261**

Chỉ mục **264**

Giới thiệu về Tác giả **284**

Lời bạt **284**

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lời nói đầu

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Làn sóng Học Máy

Năm 2006, Giáo sư Geoffrey Hinton và cộng sự đã xuất bản một bài báo khoa học[1] giới thiệu cách huấn luyện một mạng nơ-ron sâu có khả năng nhận dạng chữ viết tay với độ chính xác cao tại thời điểm đó (>98%). Họ gọi kỹ thuật này là "Học Sâu". Mạng nơ-ron sâu là một mô hình (rất) giản lược của vỏ não, gồm nhiều tầng chứa các nơ-ron nhân tạo. Huấn luyện một mạng nơ-ron sâu được xem là bất khả thi vào thời điểm đó[2], và đa số các nhà nghiên cứu đã từ bỏ ý tưởng này từ cuối những năm 1990. Bài báo này đã vực lại sự quan tâm của cộng đồng khoa học, và không lâu sau đó đã có nhiều bài báo mới đã chứng tỏ rằng Học Sâu không những khả thi, mà còn có thể đạt những thành tựu đáng kinh ngạc (nhờ sức mạnh tính toán và lượng dữ liệu khổng lồ) mà không kỹ thuật Học Máy nào khác có hy vọng đạt được. Kỹ thuật này nhanh chóng lan rộng tới những lĩnh vực Học Máy khác.

Sau gần một thập kỷ, Học Máy đã thống trị ngành công nghiệp: là tác nhân đằng sau sự thành công của hầu hết các sản phẩm công nghệ cao ngày nay, từ việc xếp hạng kết quả tìm kiếm trang web, nhận diện giọng nói trên điện thoại thông minh, đề xuất video; đến việc đánh bại nhà vô địch thế giới trong bộ môn cờ vây. Và trước khi bạn kịp nhận ra, nó sẽ thay bạn lái chính chiếc xe của mình.

Ứng dụng Học Máy vào Dự án của bạn

Ngay lúc này, chắc chắn rằng bạn hứng thú với Học Máy và rất muốn nhập cuộc.

Có những ý tưởng nào đang hiện hữu trong đầu bạn? Liệu chăng bạn muốn trao cho con robot tự chế của mình một bộ não, khiến nó nhận diện được khuôn mặt hoặc học cách đi xung quanh?

Hay là công ty bạn đang có hàng tấn dữ liệu (tệp log người dùng, dữ liệu tài chính, dữ liệu sản phẩm, dữ liệu cảm biến, dữ liệu thống kê đường dây nóng, báo cáo nhân sự, v.v.) và nhiều khả năng bạn sẽ khám phá ra một vài viên ngọc ẩn nếu biết tìm đúng chỗ. Với Học Máy, bạn có thể đạt được không chỉ những điều sau mà còn nhiều hơn thế:

- Phân đoạn khách hàng và chọn chiến lược tiếp thị tốt nhất cho mỗi phân khúc.
- Đề xuất các sản phẩm phù hợp cho khách hàng dựa trên sản phẩm được mua bởi những khách hàng tương tự.
- Phát hiện những giao dịch có khả năng cao là giả mạo.

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

[2] Dù mạng nơ-ron tích chập sâu của Yann LeCun đã hoạt động rất tốt trong tác vụ nhận dạng ảnh từ những năm 1990, tuy nó không đa dụng.

- Dự báo doanh thu của năm sau.

Bất kể mục đích là gì, bạn cũng đã quyết định học Học Máy và triển khai nó trong dự án của bạn. Đó là một dự định tuyệt vời!

Mục tiêu và Hướng tiếp cận

Cuốn sách này giả định rằng bạn gần như không biết gì về Học Máy. Mục đích của cuốn sách là cung cấp cho bạn những khái niệm, công cụ, và trực giác cần thiết để lập trình những ứng dụng có khả năng *học từ dữ liệu*.

Chúng tôi sẽ đề cập đến một lượng lớn các kỹ thuật, từ cơ bản và thông dụng nhất (như Hồi quy Tuyến tính) đến một vài kỹ thuật Học Sâu thường giành chiến thắng trong các cuộc thi.

Thay vì tự lập trình một phiên bản giản lược cho từng thuật toán, ta sẽ dùng những framework Python đã được dùng để phát triển sản phẩm thực tế:

- Scikit-Learn rất dễ sử dụng, có sẵn nhiều thuật toán Học Máy được lập trình một cách tối ưu, nên nó sẽ là một điểm khởi đầu tuyệt vời để học về Học Máy. Scikit-Learn được tạo ra bởi David Cournapeau vào năm 2007, và giờ được phát triển bởi một nhóm những nhà nghiên cứu tại Viện Nghiên cứu Khoa Học Máy tính và Tự động hóa Pháp (Inria).
- TensorFlow là một thư viện phức tạp hơn cho tác vụ tính toán số học phân tán. Nó giúp việc chạy và huấn luyện một mạng nơ-ron kích thước lớn trở nên khả thi và hiệu quả bằng cách phân phối tác vụ tính toán cho hàng nghìn, thậm chí hàng trăm máy chủ đa GPU (graphics processing unit - bộ xử lý đồ họa). TensorFlow (TF) được phát triển bởi Google, là framework đứng sau nhiều dự án Học Máy quy mô lớn. TF trở thành dự án mã nguồn mở vào tháng 11 năm 2015, và phiên bản 2.0 được phát hành vào tháng 9 năm 2019.
- Keras là một API Học Sâu bậc cao giúp việc huấn luyện và chạy mạng nơ-ron trở nên rất đơn giản. Keras có thể chạy dựa trên nền tảng TensorFlow, Theano hoặc Microsoft Cognitive Toolkit (tên gọi cũ là CNTK). TensorFlow cung cấp một phiên bản tùy chỉnh của API này, gọi là *tf.keras*, nhằm hỗ trợ các tính năng nâng cao cho Tensorflow (ví dụ như khả năng nạp dữ liệu một cách hiệu quả).

Cuốn sách này thiên về hướng ứng dụng, phát triển hiểu biết về Học Máy một cách trực quan thông qua các ví dụ thực hành cụ thể với chỉ một chút lý thuyết. Dù bạn có thể đọc cuốn sách này mà không cần laptop, chúng tôi rất khuyến khích bạn thực hành với các đoạn mã nguồn mẫu có sẵn dưới dạng các Jupyter Notebook được cung cấp tại *https://github.com/mlbvn/handson-ml2-vn*.

Kiến thức cần có

Cuốn sách này giả định rằng bạn đã có kinh nghiệm lập trình Python và đã quen thuộc với các thư viện khoa học chính của Python, cụ thể là NumPy, Pandas, và Matplotlib.

Đồng thời, nếu bạn quan tâm đến các kiến thức nền tảng, bạn cần kha khá kiến thức toán đại học như giải tích, đại số tuyến tính, xác suất và thống kê.

Trong trường hợp bạn chưa biết Python, *https://learnpython.org/* là trang web tuyệt vời để bắt đầu học. Bản hướng dẫn chính thức tại Python.org cũng khá tốt.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Nếu bạn chưa từng dùng Jupyter, Chương 2 sẽ hướng dẫn cách cài đặt và các thao tác cơ bản. Đây là một công cụ mạnh mẽ mà bạn nên biết cách sử dụng.

Nếu bạn không quen với các thư viện khoa học của Python, các Jupyter Notebook kèm theo có bao gồm một số hướng dẫn. Ngoài ra, chúng tôi cũng cung cấp thêm một bản hướng dẫn ngắn gọn về đại số tuyến tính.

Lộ trình

Cuốn sách này được chia thành hai tập. **Tập 1: Kiến thức Nền tảng về Học Máy**, đề cập đến các chủ đề sau:

- Định nghĩa Học Máy, ứng dụng, các nhóm chính và những khái niệm cơ bản trong các hệ thống Học Máy.
- Các bước cơ bản trong một dự án Học Máy.
- Học bằng cách khớp một mô hình theo dữ liệu.
- Tối ưu hàm mất mát.
- Quản lý, làm sạch và chuẩn bị dữ liệu.
- Chọn và thiết kế đặc trưng.
- Chọn mô hình và tinh chỉnh siêu tham số (*hyperparameter tuning*) bằng kiểm định chéo (*cross-validation*).
- Các thách thức trong Học Máy, đặc biệt là dưới khớp (*underfitting*) và quá khớp (*overfitting*) (đánh đổi độ chệch/phương sai).
- Các thuật toán học phổ biến nhất: Hồi quy Tuyến tính và Đa thức (*Linear and Polynomial Regression*), Hồi quy Logistic (*Logistic Regression*), k-điểm gần nhất (*k-Nearest Neighbors*), Máy Vector Hỗ trợ (*Support Vector Machines*), Cây Quyết định (*Decision Trees*), Rừng Ngẫu nhiên (*Random Forests*) và các phương pháp Ensemble.
- Giảm chiều dữ liệu huấn luyện để đối phó với "lời nguyền số chiều" (*curse of dimensionality*).
- Các kỹ thuật học không giám sát (*unsupervised learning*) khác, bao gồm phân cụm (*clustering*), ước lượng mật độ (*density estimation*) và phát hiện bất thường (*anomaly detection*).

Tập 2: Mạng nơ-ron Nhân tạo & Học sâu, đề cập đến các chủ đề sau:

- Mạng nơ-ron là gì và chúng hoạt động tốt ở những bài toán nào.
- Xây dựng và huấn luyện các mạng nơ-ron sử dụng TensorFlow và Keras.
- Các kiến trúc mạng nơ-ron quan trọng nhất: mạng truyền xuôi (*feedforward nets*) cho dữ liệu dạng bảng (*tabular data*), mạng tích chập (*convolutional nets*) cho thị giác máy tính, mạng hồi tiếp (*recurrent nets*) và bộ nhớ ngắn hạn dài (*long short-term memory - LSTM*) cho xử lý chuỗi, kiến trúc mã hóa/giải mã (encoder/decoders) và Transformers cho xử lý ngôn ngữ tự nhiên, các bộ tự mã hóa (*autoencoder*) và mạng đối sinh (*GAN*) để học các mô hình sinh (*generative learning*).
- Các kỹ thuật huấn luyện mạng Học Sâu.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- Cách xây dựng một tác tử (*agent*) (ví dụ như bot trong trò chơi) có thể học các chiến lược tốt thông qua quá trình thử và sai (*trial and error*) trong học tăng cường (*Reinforcement Learning*).
- Nạp và tiền xử lý hiệu quả lượng dữ liệu lớn.
- Huấn luyện và triển khai các mô hình TensorFlow ở quy mô lớn.

Phần đầu chủ yếu dựa trên Scikit-Learn, và phần sau sẽ sử dụng TensorFlow và Keras.

Lưu ý

Dù Học Sâu (*Deep Learning*) quả thật là một trong những lĩnh vực thú vị nhất trong Học Máy, bạn vẫn cần nắm vững các kiến thức cơ bản trước. Hơn nữa, phần lớn các bài toán có thể được giải quyết khá tốt với các kỹ thuật đơn giản như Rừng Ngẫu nhiên và các phương pháp Ensemble (được thảo luận trong Tập 1). Học Sâu phù hợp nhất với các bài toán phức tạp như nhận dạng ảnh, nhận dạng giọng nói hoặc xử lý ngôn ngữ tự nhiên, với điều kiện bạn có đủ dữ liệu, sức mạnh tính toán, và sự kiên nhẫn.

Các thay đổi trong ấn bản thứ hai

Lần tái bản này có sáu mục tiêu chính:

1. Đề cập thêm các chủ đề khác trong Học Máy: nhiều kỹ thuật học không giám sát hơn (như phân cụm, phát hiện bất thường, ước lượng mật độ và các mô hình hỗn hợp - *mixture models*); nhiều kỹ thuật huấn luyện các mạng nơ-ron sâu hơn (như các mạng tự chuẩn hóa - *self-normalized network*); thêm các kỹ thuật trong thị giác máy tính (như Xception, SENet, phát hiện vật thể với YOLO, và phân vùng ảnh (*semantic segmentation*) với R-CNN); xử lý chuỗi bằng mạng nơ-ron tích chập (*convolutional neural network - CNN*, gồm WaveNet); xử lý ngôn ngữ tự nhiên bằng mạng nơ-ron hồi tiếp (*recurrent neural network - RNN*), CNN, Transformer; và GAN.
2. Đề cập thêm các thư viện và API khác (Keras, Data API, TF-Agents cho Học Tăng cường), huấn luyện và triển khai các mô hình TF ở quy mô lớn sử dụng Distribution Strategies API, TF-Serving, và Google Cloud AI Platform. Cùng với đó giới thiệu ngắn gọn về TF Transform, TFLite, TF Addons/Seq2Seq, và TensorFlow.js.
3. Thảo luận một vài kết quả nghiên cứu quan trọng gần đây trong Học sâu.
4. Chuyển đổi tất cả các chương sử dụng TensorFlow thành TensorFlow 2, và sử dụng Keras API trong TensorFlow (`tf.keras`) bất cứ khi nào có thể.
5. Cập nhật ví dụ mã nguồn với các phiên bản mới nhất của Scikit-Learn, NumPy, Pandas, Matplotlib và các thư viện khác.
6. Giải thích rõ một vài phần và sửa một số lỗi, nhờ vào các góp ý của độc giả.

Một số chương được thêm mới, một số khác được hiệu đính, và một số ít được sắp xếp lại. Hãy truy cập *https://git.io/JCJYA* để biết thêm chi tiết các thay đổi trong ấn bản này.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các Tài liệu khác

Có rất nhiều tài liệu hay về Học Máy. Ví dụ như khoá học tuyệt vời về Học Máy trên Coursera của Andrew Ng, dù cần đầu tư nhiều thời gian (có thể là vài tháng).

Và cũng có nhiều trang web thú vị về Học Máy, tất nhiên bao gồm hướng dẫn sử dụng Scikit-Learn. Có thể bạn cũng sẽ thích Dataquest, một trang web cung cấp các tài liệu hướng dẫn có tính tương tác tốt, và các blog về Học Máy được liệt kê trên Quora. Cuối cùng, trang web về Học sâu có một danh sách tài liệu để bạn có thể tham khảo thêm.

Đây là những cuốn sách vỡ lòng khác về Học Máy:

- *Data Science from Scratch* của Joel Grus (O'Reilly) mô tả các nguyên tắc cơ bản của Học Máy và hướng dẫn lập trình (từ đầu) một số thuật toán quan trọng chỉ với Python thuần.
- *Machine Learning: An Algorithmic Perspective* của Stephen Marsland (Chapman & Hall) là một cuốn sách giới thiệu xuất sắc về Học Máy, đi sâu vào nhiều chủ đề với các ví dụ bằng mã lập trình bằng Python (sử dụng Numpy).
- *Python Machine Learning* của Sebastian Raschka (Packt Publishing) cũng là một cuốn sách giới thiệu tuyệt vời về Học Máy, tận dụng sức mạnh của các thư viện Python mã nguồn mở (Pylearn 2 và Theano).
- *Deep Learning with Python* của François Chollet (Manning) là một cuốn sách rất thực tiễn, trình bày nhiều chủ đề một cách rõ ràng và súc tích được viết bởi tác giả của thư viện Keras nổi tiếng. Cuốn sách này tập trung nhiều vào mã nguồn hơn là lý thuyết thuần túy.
- *The Hundred-Page Machine Learning Book* của Andriy Burkov là một cuốn sách súc tích và bao gồm khá nhiều các chủ đề. Cuốn sách được trình bày bằng các thuật ngữ dễ hiểu nhưng không hề né tránh các phương trình toán học.
- *Learning from Data* (AMLBook) của Yaser S. Abu-Mostafa, Malik Magdon-Ismail, và Hsuan-Tien Lin tiếp cận ML theo hướng thiên về lý thuyết, nhờ đó cung cấp cho bạn đọc cái nhìn sâu sắc hơn, đặc biệt là về đánh đổi độ lệch/phương sai (tham khảo Chương 4).
- *Artificial Intelligence: A Modern Approach*, 3rd Edition của Stuart Russell và Peter Norvig (Pearson) là một cuốn sách tuyệt vời (và khá dày) bao hàm một lượng lớn các chủ đề, trong đó có Học Máy. Cuốn sách giúp độc giả có cái nhìn sâu sắc hơn về Học Máy.

Cuối cùng, bằng cách tham gia những cuộc thi Học Máy trên Kaggle.com, bạn sẽ có cơ hội thực hành kỹ năng của mình trên những bài toán thực tế với sự trợ giúp của các chuyên gia Học Máy hàng đầu hiện nay.

Các Quy ước được Sử dụng trong Cuốn sách này

Các quy ước về kiểu chữ sau đây được sử dụng trong cuốn sách:

In Nghiêng
Các thuật ngữ mới, đường dẫn (URL), địa chỉ email, tên tệp và phần mở rộng tệp.

`Độ rộng không đổi`
Được sử dụng cho các đoạn mã, cũng như trong đoạn văn khi nhắc đến các phần tử của chương trình như tên biến, tên hàm, cơ sở dữ liệu, kiểu dữ liệu, biến môi trường, câu lệnh và từ khóa.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

`Độ rộng không đổi in đậm`
Biểu thị các câu lệnh hoặc văn bản mà người dùng cần nhập.

`Độ rộng không đổi in nghiêng`
Biểu thị văn bản cần được thay thế bằng các giá trị do người dùng cung cấp hoặc bằng các giá trị được xác định bởi ngữ cảnh.

Mẹo

Mẹo hoặc gợi ý.

Ghi chú

Giải thích thêm.

Lưu ý

Cảnh báo hoặc nội dung cần chú trọng.

Mã nguồn Minh họa

Độc giả có thể tải các Jupyter Notebook có sẵn với đầy đủ tài liệu hỗ trợ (như mã nguồn, bài tập) tại *https://github.com/mlbvn/handson-ml2-vn*.

Một số đoạn mã thường xuyên lặp lại, đơn giản hoặc không liên quan đến Học Máy sẽ được lược bỏ. Điều này giúp cuốn sách tập trung vào những phần mã nguồn quan trọng và dành thêm không gian cho các chủ đề khác. Mã nguồn đầy đủ có thể được tìm thấy trong các Jupyter Notebook.

Lưu ý, các mã nguồn minh họa có hiển thị kết quả đầu ra sẽ được trình bày cùng với ký hiệu nhắc Python (`>>>` và `...`), nhằm phân biệt giữa mã nguồn với kết quả đầu ra. Ví dụ, đoạn mã sau định nghĩa hàm `square()` được dùng để tính và hiển thị kết quả bình phương của 3:

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

Với đoạn mã không hiển thị đầu ra, ký hiệu nhắc sẽ không được sử dụng. Tuy nhiên, kết quả đầu ra có thể được hiển thị dưới dạng ghi chú như sau:

```
def square(x):
    return x ** 2

result = square(3)  # result is 9
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Sử dụng Mã nguồn Minh họa

Cuốn sách này được viết với mục đích giúp bạn hoàn thành công việc của mình. Nhìn chung, bạn có thể dùng các mã nguồn minh họa trong cuốn sách này cho các chương trình và tài liệu của mình. Bạn không cần phải liên hệ để xin phép trừ khi bạn cần sao chép một phần đáng kể các đoạn mã đó. Ví dụ: viết một chương trình sử dụng một vài đoạn mã trong cuốn sách này không cần xin phép, nhưng việc bán hoặc phân phối đĩa CD-ROM chứa các ví dụ trong cuốn sách này thì cần được cấp phép. Trả lời câu hỏi, trích dẫn nội dung hoặc mã nguồn minh họa thì không cần xin phép nhưng kết hợp một lượng đáng kể các mã nguồn minh họa từ cuốn sách này vào tài liệu sản phẩm của bạn thì cần được cấp phép.

Chúng tôi hoan nghênh việc trích dẫn nhưng không bắt buộc. Khi trích dẫn thường bao gồm tựa sách, tác giả, nhà xuất bản, và mã số ISBN. Ví dụ: "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, by Aurélien Géron (O'Reilly). Copyright 2019 Kiwisoft S.A.S., 978-1-492-03264-9." Nếu bạn cảm thấy việc sử dụng các mã nguồn minh họa không phù hợp với mục đích sử dụng hợp pháp hoặc điều kiện cấp phép ở trên, vui lòng liên hệ với chúng tôi theo địa chỉ *permissions@oreilly.com*.

Nền tảng học Trực tuyến O'Reilly

Ghi chú

Trong gần 40 năm, *O'Reilly Media* đã cung cấp các khóa đào tạo, với kiến thức và cái nhìn chuyên sâu về kinh doanh và công nghệ, từ đó giúp đỡ các công ty đi đến thành công.

Chúng tôi sở hữu mạng lưới độc nhất các chuyên gia và các nhà sáng chế, những người chia sẻ kiến thức và chuyên môn của họ thông qua sách, bài báo, hội nghị, và nền tảng học trực tuyến của chúng tôi. Nền tảng học trực tuyến O'Reilly cung cấp theo yêu cầu của bạn các khóa đào tạo trực tiếp, lộ trình học tập chuyên sâu, môi trường lập trình tương tác, cùng nguồn tài liệu, và video phong phú từ O'Reilly và hơn 200 nhà xuất bản khác. Để biết thêm thông tin, vui lòng truy cập *https://oreilly.com*.

Liên hệ với Chúng tôi

Vui lòng gửi các bình luận và câu hỏi liên quan đến cuốn sách tới nhà xuất bản:

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (tại Hoa Kỳ hoặc Canada)

707-829-0515 (gọi quốc tế hoặc trong nước)

707-829-0104 (fax)

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chúng tôi có riêng một trang web cho cuốn sách này. Tại đây chúng tôi liệt kê các hiệu đính, ví dụ và thông tin liên quan. Bạn có thể truy cập nó tại địa chỉ *https://homl.info/oreilly2*.

Để bình luận hoặc đặt các câu hỏi kỹ thuật về cuốn sách, bạn có thể gửi email tới *bookquestions@oreilly.com*.

Để biết thêm thông tin về các sản phẩm sách, khóa học, hội nghị, và tin tức, truy cập website của chúng tôi tại *https://www.oreilly.com*.

Tìm chúng tôi trên Facebook: *https://facebook.com/oreilly*

Theo dõi chúng tôi trên Twitter: *https://twitter.com/oreillymedia*

Xem chúng tôi trên Youtube: *https://www.youtube.com/oreillymedia*

Lời cảm ơn

Dù nằm mơ tôi cũng không dám tưởng tượng ấn bản đầu tiên của cuốn sách lại được đón nhận bởi nhiều độc giả đến vậy. Tôi đã nhận được rất nhiều tin nhắn từ độc giả bốn phương với rất nhiều câu hỏi, cả các lỗi sai, và phần lớn gửi đến tôi những lời động viên khích lệ. Tôi không biết phải nói sao để bày tỏ hết lòng cảm ơn của mình đối với sự ủng hộ vô bờ bến của quý độc giả. Cảm ơn mọi người rất nhiều! Nếu bạn tìm thấy lỗi trong các mã nguồn minh họa, hay chỉ đơn thuần muốn được trả lời các khúc mắc, vui lòng để lại issue trên Github, hoặc nếu thấy lỗi chính tả, hãy tạo hiệu đính (với ấn bản tiếng Việt vui lòng để lại phản hồi trên Github Discussion). Một vài độc giả còn chia sẻ cuốn sách này giúp họ tìm công việc đầu tiên, hoặc giải quyết một vấn đề cụ thể như thế nào. Những phản hồi như vậy thật sự tạo động lực rất lớn cho tôi. Nếu cảm thấy cuốn sách này hữu ích, tôi rất mong bạn có thể chia sẻ câu chuyện của mình với tôi, có thể riêng tư (như qua LinkedIn) hoặc công khai (như một dòng tweet hoặc qua đánh giá trên Amazon).

Tôi cũng rất biết ơn những độc giả rất tuyệt vời, mặc dù bận rộn cũng dành thời gian quý báu đưa nhận xét rất có tâm cho cuốn sách. Đặc biệt, tôi xin gửi lời cảm ơn tới François Chollet, người đã bình luận và đánh giá cho tất cả các chương viết bằng Keras và TensorFlow, cũng như đưa ra nhiều góp ý rất sâu sắc. Vì Keras là một trong những phần được bổ sung chính trong ấn bản này, được nhận xét bởi chính tác giả của thư viện là một điều vô giá. Tôi đặc biệt khuyến khích bạn tham khảo thêm cuốn sách *Deep Learning with Python* (Manning) của François: súc tích, rõ ràng và chuyên sâu về thư viện Keras. Đặc biệt cảm ơn tới Ankur Patel, người đóng góp đánh giá tất cả các chương của ấn bản này và đưa ra các phản hồi tuyệt vời, đặc biệt là Chương 9, đề cập đến các kỹ thuật học không giám sát. Anh ấy có thể viết cả một cuốn sách về chủ đề này... À mà, khoan đã, anh ấy đã viết mất rồi! Bạn cũng có thể tham khảo *Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data* (O'Reilly). Tôi cũng vô cùng cảm ơn Olzhas Akpambetov, người đóng góp đánh giá cho tất cả các chương của phần hai, chạy thử phần lớn mã nguồn, và đưa ra những góp ý tuyệt vời. Tôi rất biết ơn Mark Daoust, Jon Krohn, Dominic Monn, và Josh Patterson đã góp ý kỹ càng phần hai của cuốn sách và cho tôi lời khuyên từ kinh nghiệm của họ. Họ thật sự rất tận tâm và đã đưa ra những phản hồi vô cùng hữu ích.

Khi chuẩn bị ấn bản này, tôi rất may mắn được các thành viên nhóm TensorFlow giúp đỡ, đặc biệt là Martin Wicke, người trả lời và chuyển đến những người có thể trả lời hàng tá câu hỏi của tôi, gồm Karmel Allison, Paige Bailey, Eugene Brevdo, William Chargin, Daniel "Wolff" Dobson, Nick Felt, Bruce Fontaine, Goldie Gadde, Sandeep Gupta, Priya Gupta, Kevin Haas, Konstantinos Katsiapis ,Viacheslav Kovalevskyi, Allen Lavoie, Clemens Mewald, Dan Moldovan, Sean Morgan, Tom O'Malley, Alexandre Passos, André Susano Pinto, Anthony Platanios, Oscar Ramirez, Anna Revinskaya, Saurabh Saxena, Ryan Sepassi, Jiri Simsa, Xiaodan Song, Christina

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Sorokin, Dustin Tran, Todd Wang, Pete Warden (người cũng đã đóng góp đánh giá cho cuốn sách trong ấn bản đầu tiên), Edd Wilder-James, và Yuefeng Zhou. Tôi vô cùng cảm ơn mọi người và các thành viên khác trong nhóm TensorFlow, không chỉ bởi sự giúp đỡ của mọi người, mà còn vì đã tạo ra một thư viện tuyệt vời! Đặc biệt cảm ơn Irene Giannoumis và Robert Crowe của nhóm TFX đã đánh giá chương 4 và chương 10 (trong Tập 2) một cách kỹ càng.

Tôi cũng muốn gửi lời cảm ơn đến những nhân viên của O'Reilly, đặc biệt là Nicole Taché, người và đưa ra các phản hồi sâu sắc và luôn nhiệt tình động viên và giúp đỡ tôi, tôi không thể mơ về một biên tập viên tốt hơn. Cảm ơn Michele Cronin, người đã nhiệt tình giúp đỡ (và kiên nhẫn) khi bắt đầu ấn bản này, và Kristen Brown, biên tập viên sản xuất của ấn bản thứ hai, người chứng kiến cuốn sách đi qua tất cả các bước (cô ấy cũng giúp sửa và cập nhật mỗi lần in thêm trong ấn bản đầu tiên). Cảm ơn Rachel Monaghan vì đã biên soạn cẩn thận cuốn sách trong ấn bản đầu tiên cũng như Amanda Kersey cho lần biên soạn thứ hai. Cảm ơn Johnny O'Toole người phụ trách quan hệ với Amazon và trả lời rất nhiều câu hỏi của tôi. Cảm ơn Marie Beaugureau, Ben Lorica, Mike Loukides, and Laurel Ruma vì đã tin tưởng và giúp tôi xác định phạm vi dự án này. Cảm ơn Matt Hacker và team Atlas đã giải đáp các vướng mắc của tôi về định dạng AsciiDoc và LaTeX, và Nick Adams, Rebecca Demarest, Rachel Head, Judith McConville, Helen Monroe, Karen Montgomery, Rachel Roumeliotis và tất cả mọi người ở O'Reilly đã đóng góp vào cuốn sách này.

Tôi cũng muốn cảm ơn những đồng nghiệp cũ ở Google, đặc biệt là nhóm phân loại video YouTube đã dạy tôi rất nhiều về Học Máy. Không có họ tôi đã không thể có ấn bản đầu tiên. Đặc biệt cảm ơn những người hướng dẫn của tôi: Clément Courbet, Julien Dubois, Mathias Kende, Daniel Kitachewsky, James Pack, Alexander Pak, Anosh Raj, Vitor Sessak, Wiktor Tomczak, Ingrid von Glehn, và Rich Washington. Cảm ơn tất cả những người khác làm việc cùng tôi tại YouTube và các nhóm nghiên cứu tuyệt vời của Google tại Mountain View. Cảm ơn Martin Andrews, Sam Witteveen, và Jason Zaman vì đã chào đón tôi vào nhóm Google Developer Experts tại Singapore, với sự trợ giúp từ Soonson Kwon, và vì tất cả những thảo luận tuyệt vời về Học sâu và TensorFlow. Bất cứ ai có hứng thú về Học sâu tại Singapore nên tham gia Deep Learning Singapore meetup của họ. Tôi cũng đặc biệt cảm ơn Jason đã chia sẻ chuyên môn về TFLite cho Chương 10 (trong Tập 2)!

Tôi sẽ không bao giờ quên những người đã đóng góp phản biện cho ấn bản đầu của cuốn sách này, bao gồm David Andrzejewski, Lukas Biewald, Justin Francis, Vincent Guilbeau, Eddy Hung, Karim Matrah, Grégoire Mesnil, Salim Sémaoune, Iain Smears, Michel Tessier, Ingrid von Glehn, Pete Warden, và người anh trai Sylvain. Đặc biệt cảm ơn Haesun Park với những phản hồi tuyệt vời, cũng như đã phát hiện một vài lỗi khi dịch ấn bản đầu ra tiếng Hàn. Anh ấy cũng dịch các Jupyter notebook ra tiếng Hàn, chưa kể tài liệu về TensorFlow. Tôi không biết tiếng Hàn, nhưng qua chất lượng các phản hồi từ anh ấy, các bản dịch chắc chắn vô cùng tuyệt vời! Haesun cũng đóng góp một vài lời giải cho các bài tập trong ấn bản này.

Cuối cùng, tôi xin thể hiện lòng biết ơn vô hạn đến người vợ yêu quý Emmanuelle, cùng ba người con Alexandre, Rémi, và Gabrielle đã khuyến khích tôi thực hiện cuốn sách này. Tôi cũng biết ơn họ vì sự tò mò không thể thỏa mãn: giải thích một số khái niệm khó nhất trong cuốn sách này cho vợ và các con giúp tôi làm rõ suy nghĩ của mình và trực tiếp cải thiện nhiều phần trong cuốn sách. Họ cũng chuẩn bị cà phê và bánh quy giúp tôi. Tôi còn có thể mong muốn điều gì tuyệt vời hơn nữa đây?

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 1

Toàn cảnh Học Máy

Phần lớn mọi người khi nghe tới thuật ngữ "Học Máy" sẽ liên tưởng ngay đến những con rô bốt phục vụ hay người máy chết chóc trong bộ phim Kẻ Hủy Diệt. Tuy nhiên, Học Máy giờ đây không chỉ là giấc mơ viễn tưởng nữa, mà chúng đã có mặt ở đây rồi. Trên thực tế, Học Máy đã được sử dụng từ hàng thập kỷ qua với các ứng dụng chuyên biệt, chẳng hạn như nhận dạng ký tự quang học. Thế nhưng ứng dụng đầu tiên đưa Học Máy đến với đại chúng, thứ đã cải thiện chất lượng cuộc sống của hàng trăm triệu người và trở nên cực kỳ phổ biến vào những năm 1990 là *bộ lọc thư rác*. Ứng dụng này không cao siêu như mạng Skynet trong Kẻ Hủy Diệt, nhưng lại đủ tiêu chuẩn để được xem là một kỹ thuật Học Máy. Và thực tế, bộ lọc hoạt động tốt đến nỗi hiếm khi người dùng phải tự gán nhãn thư rác nữa. Theo sau đó, hàng trăm ứng dụng Học Máy được âm thầm đưa vào hàng trăm sản phẩm và tính năng mà chúng ta đang dùng mỗi ngày, từ các hệ thống đề xuất tới tính năng tìm kiếm bằng giọng nói.

Vậy Học Máy bắt đầu từ đâu và sẽ đi đến đâu? Việc một chiếc máy *học* được một điều gì đó thật sự là thế nào? Nếu ta tải về trang Wikipedia thì máy tính có thật sự học được cái gì không? Nó sẽ tự nhiên thông minh hơn chăng? Trong chương này, chúng ta sẽ bắt đầu giải thích rõ định nghĩa và lý do mà ta có thể muốn sử dụng Học Máy.

Trước khi bắt đầu khám phá thế giới Học Máy, hãy cùng nhìn tổng quan để nắm rõ các "vùng" chính và các "địa điểm" nổi bật trong Học Máy: học có và không giám sát, học trực tuyến và học theo batch, học dựa trên mẫu và học dựa trên mô hình. Sau đó, chúng ta sẽ xem xét các bước tiến hành một dự án Học Máy, thảo luận về các thách thức sẽ gặp phải và giải thích cách đánh giá cũng như tinh chỉnh một hệ thống Học Máy.

Chương này giới thiệu rất nhiều lý thuyết căn bản (và thuật ngữ chuyên ngành) mà hầu hết các nhà khoa học dữ liệu đều phải thuộc lòng. Chương này sẽ mang tính khái quát cao (đây là chương duy nhất không có nhiều đoạn mã) và khá đơn giản. Tuy nhiên, bạn nên nắm rõ ngọn ngành trước khi tiếp tục tìm hiểu phần còn lại của cuốn sách. Giờ thì hãy chuẩn bị một ly cà phê và bắt đầu thôi!

Mẹo

Nếu đã quen thuộc với các khái niệm cơ bản của Học Máy, bạn có thể bắt đầu đọc từ Chương 2. Nếu bạn không chắc chắn lắm về điều này, hãy thử trả lời các câu hỏi ở cuối chương trước khi đọc tiếp.

Học Máy là gì?

Học Máy là một môn khoa học (và cả nghệ thuật) về cách lập trình máy tính để chúng có thể *học từ dữ liệu*.

Dưới đây là định nghĩa tổng quát hơn:

> [Học Máy là] lĩnh vực nghiên cứu nhằm giúp máy tính có khả năng học mà không cần lập trình một cách tường minh.
>
> — Arthur Samuel, 1959

Và định nghĩa mang tính kỹ thuật hơn:

> Một chương trình máy tính được cho là học từ kinh nghiệm E với tác vụ T và phép đo chất lượng P nào đó, nếu chất lượng của tác vụ T, được đo bởi P, cải thiện theo kinh nghiệm E.
>
> — Tom Mitchell, 1997

Bộ lọc thư rác chính là một chương trình Học Máy có khả năng học để phân loại đâu là thư rác từ các mẫu cho trước (thư được đánh dấu là rác bởi người dùng) và các mẫu thư thường (không phải thư rác). Tập hợp các mẫu mà hệ thống dùng để học được gọi là *tập huấn luyện*. Mỗi mẫu dữ liệu huấn luyện được gọi là *mẫu huấn luyện* (hay *mẫu*). Trong ví dụ này, tác vụ T là việc gán nhãn thư rác cho thư điện tử mới, kinh nghiệm E là *dữ liệu huấn luyện*, và ta cần định nghĩa thêm phép đo chất lượng P. Một lựa chọn khả thi cho P là tỷ lệ phân loại thư đúng, và phép đo chất lượng cụ thể này được gọi là *độ chính xác*. Phép đo này thường được dùng trong các bài toán phân loại.

Nếu chỉ tải xuống một bản sao của trang Wikipedia, máy tính của bạn sẽ có thêm rất nhiều dữ liệu nhưng nó sẽ không tự nhiên làm việc tốt hơn. Do đó việc tải xuống một bản sao của trang Wikipedia không phải là Học Máy.

Tại sao lại dùng Học Máy?

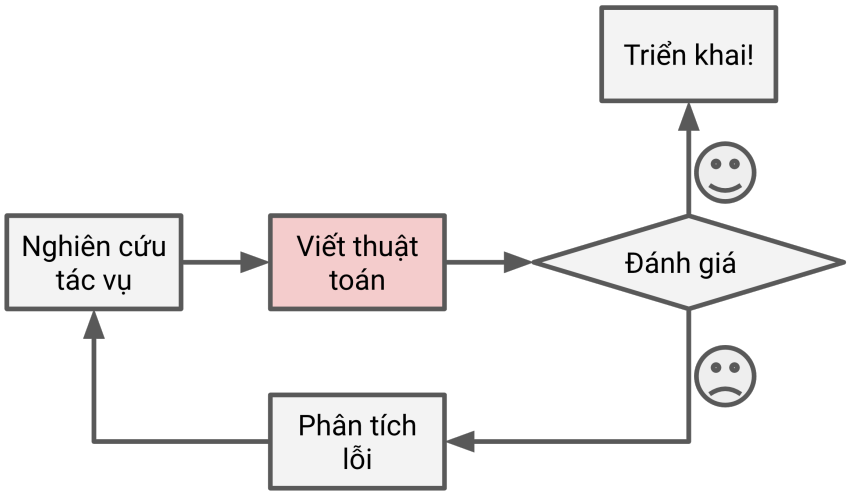
Hãy xem xét cách viết một bộ lọc thư rác bằng kỹ thuật lập trình truyền thống (Hình 1.1):

1. Đầu tiên ta cần kiểm tra xem thư rác thường trông như thế nào. Ta có thể phát hiện một số từ hoặc cụm từ (như "4U," "credit card," "free," và "amazing") hay xuất hiện trong tiêu đề thư. Ta cũng có thể thấy một vài khuôn mẫu khác ở tên người gửi, nội dung thư và ở các phần khác của thư.

2. Nếu ta viết thuật toán nhận diện cho từng khuôn mẫu trên, chương trình sẽ đánh dấu một thư điện tử là thư rác nếu một vài khuôn mẫu đó được tìm thấy.

3. Tiếp đến, ta kiểm thử chương trình và lặp lại hai bước trên cho đến khi đạt mức chất lượng để triển khai.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



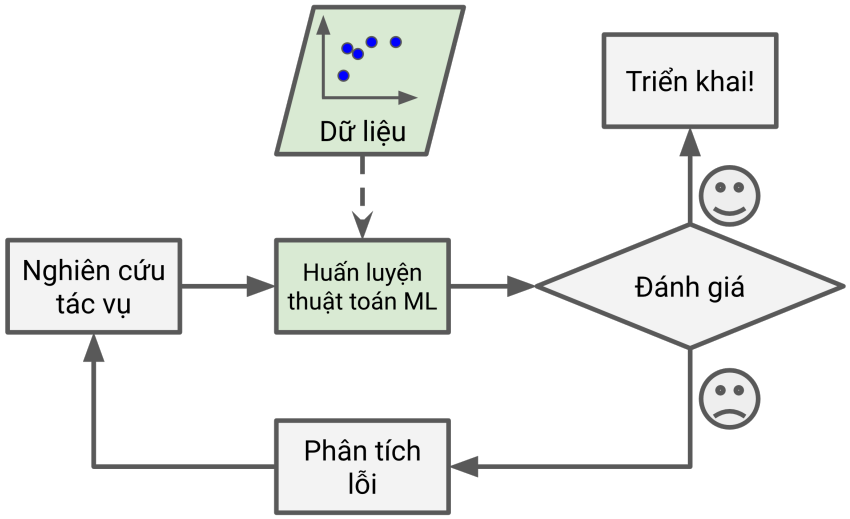
Hình 1.1. Tiếp cận theo hướng truyền thống

Vì đây là một bài toán khó, khả năng cao chương trình này sẽ trở thành một danh sách dài các quy luật phức tạp và khó bảo trì.

Trái lại, một bộ lọc thư rác dựa trên các kỹ thuật Học Máy sẽ tự động học được những từ và cụm từ nào là dấu hiệu của thư rác bằng cách nhận diện các khuôn mẫu có tần suất cao bất thường trong các mẫu thư rác so với các mẫu thư bình thường (Hình 1.2). Chương trình này ngắn hơn hẳn, dễ bảo trì hơn, và rất có thể sẽ chính xác hơn.

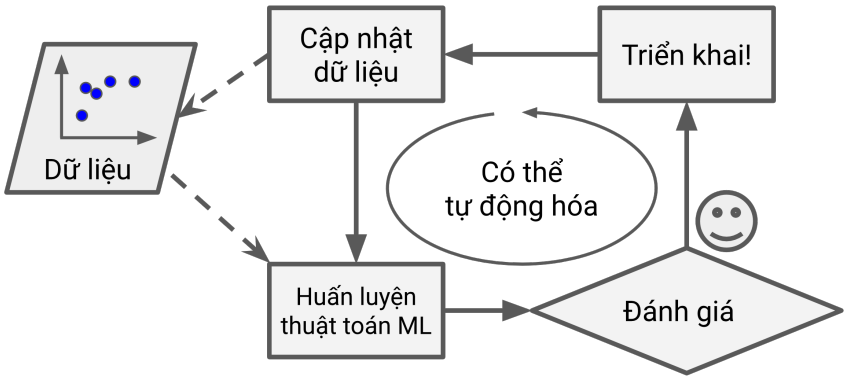
Nếu những kẻ gửi thư rác nhận ra rằng các thư điện tử chứa cụm "4U" bị chặn thì sao? Có thể chúng sẽ bắt đầu thay thế "4U" bằng "For U". Một bộ lọc thư rác được lập trình theo hướng truyền thống sẽ cần được cập nhật để đánh dấu những thư điện tử chứa cụm "For U". Nếu những kẻ này vẫn cố gắng lách qua bộ lọc thư rác, ta phải luôn phải cập nhật thêm các quy luật mới.

Trái lại, một bộ lọc thư rác dựa trên các kỹ thuật Học Máy sẽ tự động nhận thấy rằng cụm từ "For U" đã bắt đầu xuất hiện nhiều bất thường trong các bức thư rác được đánh dấu bởi người dùng, và nó sẽ bắt đầu đánh dấu các thư này mà không cần sự can thiệp bên ngoài (Hình 1.3).



Hình 1.2. Tiếp cận theo hướng Học Máy

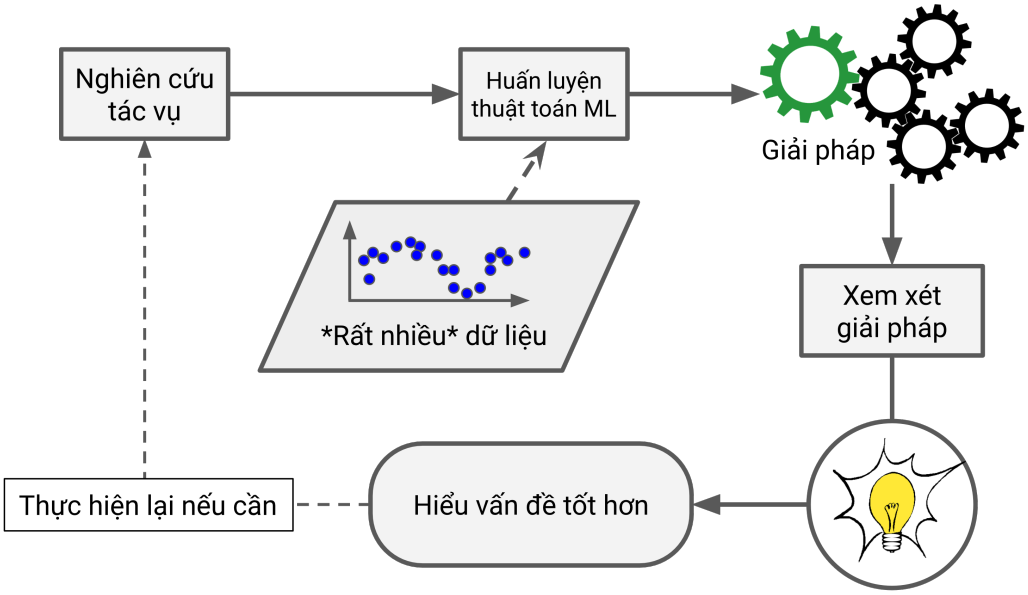
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.3. Tự động thích ứng với thay đổi

Một lĩnh vực khác mà Học Máy tỏa sáng là những bài toán quá phức tạp để có thể giải quyết theo hướng truyền thống hoặc không có sẵn giải thuật. Hãy lấy nhận diện giọng nói làm ví dụ. Giả sử ta muốn viết một chương trình đơn giản có khả năng phân biệt hai từ "one" và "two". Dễ thấy rằng từ "two" bắt đầu với một âm cao ("T"), nên ta chỉ cần dùng cao độ âm thanh làm tiêu chí phân loại. Nhưng tất nhiên kỹ thuật này không thể hoạt động tốt với hàng ngàn từ được phát âm bởi hàng triệu người khác nhau trong không gian ồn ào và khi có cả tá ngôn ngữ khác nhau. Giải pháp tốt nhất (ít nhất là tại thời điểm viết quyển sách này) là viết một thuật toán có khả năng tự học khi được cung cấp nhiều bản thu âm mẫu của mỗi từ.

Cuối cùng, Học Máy còn có thể giúp con người học (Hình 1.4). Ta có thể kiểm tra các thuật toán học máy để biết những gì mà chúng đã học được (mặc dù việc này có thể trở nên khó khăn đối với một số thuật toán nhất định). Ví dụ, sau khi một bộ lọc thư rác đã được huấn luyện với đủ mẫu thư rác, ta có thể dễ dàng xem được danh sách các từ và tổ hợp từ được thuật toán cho là những dấu hiệu tốt nhất để nhận biết thư rác. Đôi khi chúng sẽ tiết lộ sự tương quan mà ta không hề hay biết hoặc các xu hướng mới, từ đó giúp ta hiểu rõ bài toán hơn. Việc áp dụng các kỹ thuật học máy để khai phá lượng dữ liệu lớn có thể giúp tìm ra các khuôn mẫu mà ta không thể thấy trực tiếp. Đây được gọi là *khai phá dữ liệu (data mining)*.



Hình 1.4. Học Máy có thể giúp con người học

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tóm lại, Học Máy rất tốt cho:

- Những bài toán mà các giải pháp hiện có đòi hỏi quá nhiều quy luật hoặc cần tinh chỉnh nhiều: một thuật toán Học Máy thường có thể đơn giản hóa mã nguồn và hoạt động tốt hơn so với hướng truyền thống.
- Những bài toán phức tạp mà các phương pháp truyền thống không hoạt động tốt: giải pháp có thể là những kỹ thuật Học Máy tốt nhất.
- Môi trường thay đổi: một hệ thống Học Máy có thể thích ứng với dữ liệu mới.
- Việc khám phá tri thức từ các bài toán phức tạp và lượng dữ liệu lớn.

Các Ứng dụng Tiêu biểu

Hãy cùng điểm qua một vài ví dụ cụ thể về các tác vụ Học Máy, cũng như một số kỹ thuật để giải quyết các tác vụ đó:

Phân tích hình ảnh và tự động phân loại sản phẩm trên dây chuyền sản xuất
Đây là bài toán phân loại ảnh và thường được giải quyết bằng mạng nơ-ron tích chập (*Convolutional Neural Network* – CNN; sẽ được đề cập trong Tập 2).

Phát hiện khối u trong ảnh quét não
Đây là bài toán phân vùng theo nhóm (*semantic segmentation*), trong đó mọi điểm ảnh đều được phân loại (vì ta cần xác định vị trí chính xác và hình dạng của khối u). CNN cũng là phương pháp thường được sử dụng trong bài toán này.

Phân loại tin tức tự động
Đây là bài toán xử lý ngôn ngữ tự nhiên (*Natural Language Processing* – NLP), cụ thể hơn là phân loại văn bản, được giải quyết bằng mạng nơ-ron hồi tiếp (*Recurrent Neural Network* – RNN), CNN, hoặc Transformer (sẽ được đề cập trong Tập 2).

Đánh dấu bình luận phản cảm trong diễn đàn một cách tự động
Đây cũng là bài toán phân loại văn bản, sử dụng chung các công cụ NLP đã kể trên.

Tóm tắt tài liệu tự động
Đây là một nhánh của NLP được gọi là tóm tắt văn bản (*text summarization*), và cũng sử dụng các công cụ như trên.

Tạo một chatbot hoặc trợ lý cá nhân
Bài toán này liên quan đến nhiều tác vụ trong NLP, bao gồm hiểu ngôn ngữ tự nhiên (*Natural Language Understanding* – NLU) và hệ thống hỏi-đáp.

Dự báo doanh thu công ty của năm tiếp theo, dựa trên nhiều chỉ số hiệu suất
Đây là tác vụ hồi quy (nghĩa là dự đoán giá trị) và có thể được giải quyết bằng bất kỳ mô hình hồi quy nào, như Hồi quy Tuyến tính (*Linear Regression*) hoặc Hồi quy Đa thức (tham khảo Chương 4), SVM hồi quy (tham khảo Chương 5), Rừng Ngẫu nhiên Hồi quy (tham khảo Chương 7), hoặc mạng nơ-ron nhân tạo (*artificial neural network* – sẽ được đề cập trong Tập 2). Nếu muốn đưa vào mô hình một chuỗi các chỉ số hiệu suất trong quá khứ, ta có thể sử dụng RNN, CNN, hoặc Transformer (sẽ được đề cập trong Tập 2).

Tương tác với ứng dụng thông qua giọng nói
Đây là bài toán nhận diện giọng nói (*speech recognition*) và đòi hỏi ta phải xử lý các đoạn âm thanh. Vì bản thân các đoạn âm thanh là các chuỗi dài và phức tạp, chúng thường được xử lý bằng RNN, CNN hoặc Transformer (sẽ được đề cập trong Tập 2).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phát hiện gian lận thẻ tín dụng
Đây là bài toán phát hiện bất thường (*anomaly detection* – tham khảo Chương 9).

Phân nhóm khách hàng dựa trên sản phẩm tiêu thụ để thiết kế chiến lược tiếp thị khác nhau cho mỗi phân khúc
Đây là bài toán phân cụm (*clustering* – tham khảo Chương 9).

Biểu diễn một tập dữ liệu phức tạp, nhiều chiều trong biểu đồ một cách rõ ràng và hữu ích
Đây là bài toán trực quan hóa dữ liệu, thường liên quan tới các kỹ thuật giảm chiều (tham khảo Chương 8).

Gợi ý sản phẩm mà khách hàng có thể sẽ quan tâm dựa trên những sản phẩm mà họ đã mua trong quá khứ
Đây là bài toán xây dựng hệ thống đề xuất. Một hướng tiếp cận là đưa các đơn hàng trong quá khứ (và các thông tin khác về khách hàng) vào một mạng nơ-ron nhân tạo (sẽ được đề cập trong Tập 2) để dự đoán sản phẩm có khả năng cao sẽ được mua tiếp theo. Mạng nơ-ron này thường được huấn luyện với chuỗi các sản phẩm đã mua trong quá khứ của mọi khách hàng.

Xây dựng bot thông minh biết chơi trò chơi
Bài toán này thường được giải quyết thông qua Học Tăng Cường (*Reinforcement Learning* hay *RL* – sẽ được đề cập trong Tập 2), một nhánh của Học Máy với mục tiêu huấn luyện tác nhân (*agent* – ở đây là bot) để chọn các hành động sao cho phần thưởng được cực đại hóa theo thời gian (ví dụ, con bot có thể được thưởng mỗi khi người chơi bị mất máu), trong một môi trường cho trước (trường hợp này là trong một trò chơi). Chương trình AlphaGo nổi tiếng từng đánh bại nhà vô địch thế giới trong bộ môn cờ vây đã được xây dựng thông qua RL.

Danh sách này còn rất rất dài, nhưng hy vọng những ví dụ trên đã giúp bạn nắm được phần nào về độ rộng và phức tạp của các tác vụ mà Học Máy có thể giải quyết cũng như các kỹ thuật tương ứng mà ta có thể áp dụng.

Các kiểu Hệ thống Học Máy

Có rất nhiều kiểu hệ thống Học Máy khác nhau, và chúng có thể được phân loại thành các hạng mục rộng theo các tiêu chí sau:

- Chúng có được huấn luyện dưới sự giám sát của con người hay không (học giám sát, học không giám sát, học bán giám sát và học tăng cường)
- Chúng có thể học từ các dòng dữ liệu gia tăng hay không (học trực tuyến so với học theo batch)
- Chúng hoạt động bằng cách chỉ đơn thuần so sánh các điểm dữ liệu mới với các điểm dữ liệu đã biết, hay bằng cách phát hiện các khuôn mẫu trong dữ liệu huấn luyện và xây dựng một mô hình dự đoán, tương tự công việc của các nhà khoa học (học dựa trên mẫu so với học dựa trên mô hình).

Các tiêu chí này không xung khắc lẫn nhau, và có thể được kết hợp một cách tùy ý. Ví dụ, một hệ thống lọc thư rác tân tiến có thể học liên tục bằng cách huấn luyện một mô hình mạng nơ-ron sâu trên các mẫu thư rác và thư thông thường mới. Do đó, đây là một hệ thống học trực tuyến, dựa trên mô hình và có giám sát.

Hãy cùng xem xét từng yếu tố trên kỹ hơn một chút.

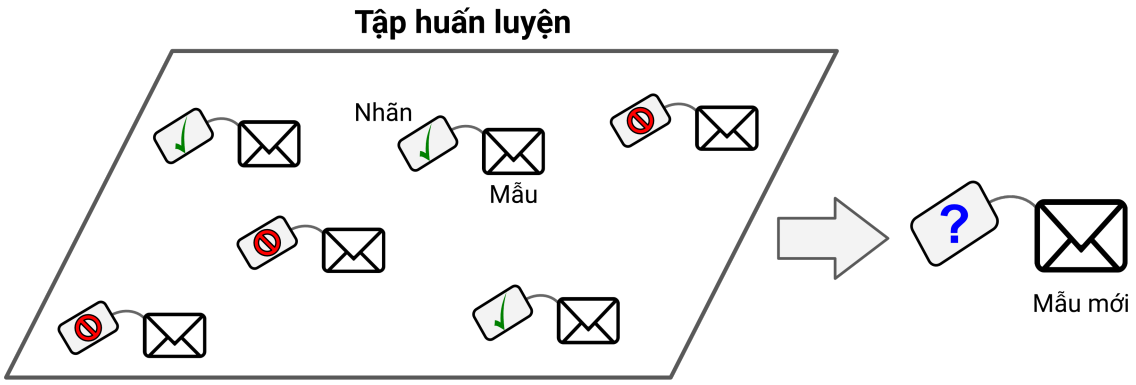
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Học có Giám sát/không Giám sát

Các hệ thống học máy có thể được phân loại dựa trên mức độ và kiểu giám sát được thực hiện khi chúng đang ở trong giai đoạn huấn luyện. Có bốn hạng mục chính: học có giám sát, học không giám sát, học bán giám sát và học tăng cường.

Học Có Giám Sát

Trong *học có giám sát* (*supervised learning*), tập huấn luyện mà ta đưa vào thuật toán đã bao gồm cả kết quả mong muốn, và kết quả đó được gọi là *nhãn* (Hình 1.5).



Hình 1.5. Một tập huấn luyện đã được gán nhãn cho tác vụ phân loại thư rác (một ví dụ của học có giám sát)

Một tác vụ học có giám sát điển hình là *phân loại* (*classification*). Bộ lọc thư rác là ví dụ tiêu biểu cho tác vụ này: nó được huấn luyện với nhiều mẫu thư điện tử cùng với *nhãn* tương ứng (thư rác hoặc thư thông thường), từ đó học cách phân loại các thư điện tử mới.

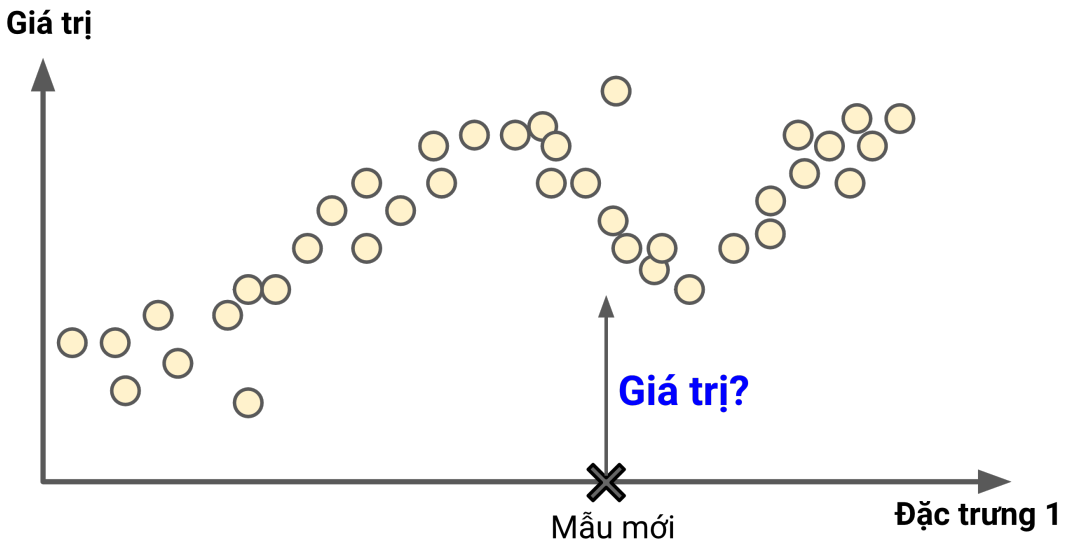
Một tác vụ điển hình khác là dự đoán giá trị số *mục tiêu*, chẳng hạn như giá xe hơi từ một tập hợp các *đặc trưng* cho trước (số dặm, tuổi xe, hãng xe, v.v) được gọi là *yếu tố dự đoán* (*predictor*). Tác vụ này có tên gọi là *hồi quy* (*regression* – Hình 1.6).[1] Để huấn luyện hệ thống này, ta cần cung cấp cho nó rất nhiều các mẫu xe hơi cùng các yếu tố dự đoán và nhãn của chúng (giá xe).

> **Ghi chú**
>
> Trong Học Máy, *thuộc tính* (*attribute*) là một kiểu dữ liệu (ví dụ: "số dặm"), trong khi *đặc trưng* (*feature*) có thể có nhiều nghĩa tùy thuộc vào ngữ cảnh. Nhìn chung, một đặc trưng là một thuộc tính đi kèm với giá trị của nó (ví dụ, "số dặm = 15,000"). Tuy nhiên, nhiều người thường sử dụng hai từ *thuộc tính* và *đặc trưng* với ý nghĩa giống nhau.

Lưu ý rằng một số thuật toán hồi quy cũng có thể được sử dụng để phân loại và ngược lại. Ví dụ, *Hồi quy Logistic* (*Logistic Regression*) thường được sử dụng để phân loại, bởi nó có thể trả về một giá trị tương ứng với xác suất mà mẫu dữ liệu thuộc về một lớp nhất định (ví dụ: 20% khả năng là thư rác).

[1] Cái tên nghe có vẻ kỳ lạ này là một thuật ngữ thống kê được giới thiệu bởi Francis Galton khi ông đang nghiên cứu về việc con cái của những người cao thường có xu hướng thấp hơn cha mẹ của chúng. Vì những đứa trẻ có chiều cao thấp hơn, ông gọi hiện tượng này là *hồi quy đến trung bình* (*regression to the mean*). Tên gọi này về sau được sử dụng cho các phương pháp phân tích mối tương quan giữa các biến của ông.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



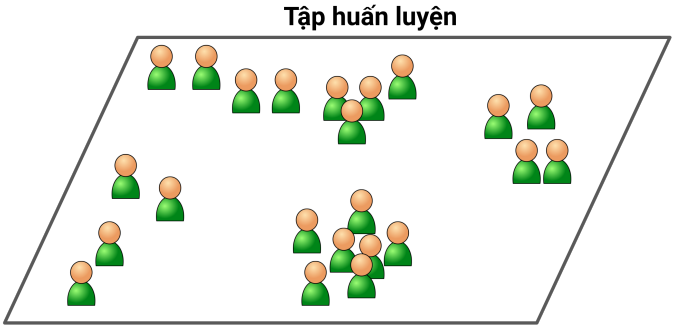
Hình 1.6. Một bài toán hồi quy: dự đoán một giá trị dựa trên đặc trưng đầu vào (thông thường có thể có nhiều đầu vào và đôi khi là nhiều đầu ra)

Sau đây là một số thuật toán học có giám sát quan trọng nhất (được đề cập trong cuốn sách này):

- k-Điểm Gần nhất
- Hồi quy Tuyến tính
- Hồi quy Logistic
- Máy Vector Hỗ trợ
- Cây Quyết định và Rừng Ngẫu nhiên
- Mạng nơ-ron[2]

Học Không Giám Sát

Trong *học không giám sát* (*unsupervised learning*), như bạn đã có thể đoán được, dữ liệu huấn luyện không được gán nhãn (Hình 1.7). Hệ thống cố gắng tự học mà không cần giáo viên.



Hình 1.7. Một tập dữ liệu không có nhãn dành cho tác vụ học không giám sát

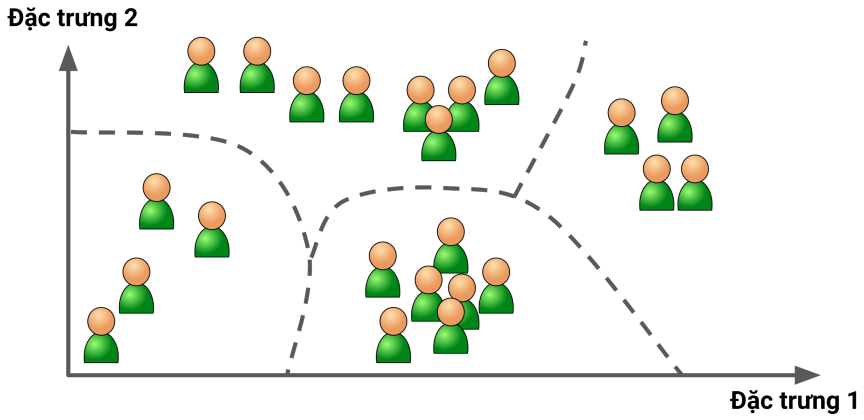
[2] Một số kiến trúc mạng nơ-ron có thể học một cách không giám sát, ví dụ như bộ tự mã hóa (*autoencoder*) và máy Boltzmann giới hạn (*restricted Boltzmann machines*). Chúng cũng có thể học theo hướng bán giám sát, ví dụ như mạng niềm tin sâu (*deep belief network*) hoặc thông qua quá trình tiền huấn luyện không giám sát.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Dưới đây là một số thuật toán học không giám sát (đa số sẽ được đề cập trong các Chương 8 và Chương 9).

- Phân cụm (Clustering)
  - K-Điểm trung bình (K-Means)
  - DBSCAN
  - Phân cụm phân cấp (Hierarchical Cluster Analysis – HCA)
- Phát hiện bất thường và tính mới (Anomaly detection and novelty detection)
  - SVM một lớp (One-class SVM)
  - Rừng cô lập (Isolation Forest)
- Trực quan hóa (visualization) và giảm chiều (dimensionality reduction)
  - Phân tích thành phần chính (Principal Component Analysis – PCA)
  - PCA hạt nhân (Kernel PCA)
  - Embedding tuyến tính cục bộ (Locally Linear Embedding – LLE)
  - Embedding lân cận ngẫu nhiên theo phân phối t (t-Distributed Stochastic Neighbor Embedding – t-SNE)
- Học luật kết hợp (Association rule)
  - Apriori
  - Eclat

Giả sử bạn có rất nhiều dữ liệu về người đọc blog của mình. Bạn có thể sẽ muốn chạy một thuật toán *phân cụm* để phát hiện các nhóm người đọc giống nhau (Hình 1.8). Bạn không hề cho thuật toán biết mỗi người thuộc nhóm nào mà nó sẽ phải tự tìm các mối liên kết. Ví dụ, thuật toán có thể thấy rằng 40% người đọc là nam giới, thích đọc truyện tranh và thường đọc blog của bạn vào buổi tối, trong khi 20% còn trẻ, thích khoa học viễn tưởng và thường đọc blog vào cuối tuần. Nếu bạn sử dụng thuật toán *phân cụm phân cấp*, mỗi nhóm có thể được phân chia thành các nhóm nhỏ hơn. Việc phân cụm có thể giúp bạn viết bài nhắm đến từng nhóm.

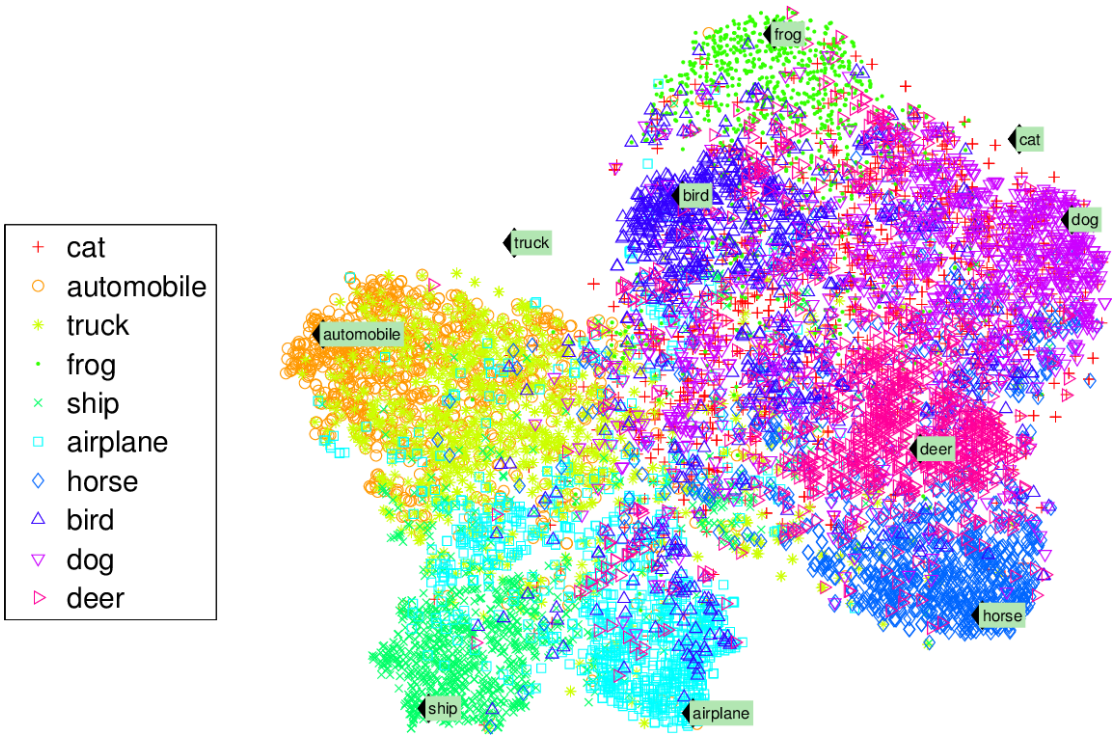


Hình 1.8. Phân cụm

Các thuật toán *trực quan hóa* cũng là ví dụ tiêu biểu của học không giám sát: ta đưa vào rất nhiều dữ liệu phức tạp, không có nhãn, và thuật toán trả về biểu diễn hai hoặc ba chiều của

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

dữ liệu, có thể được minh họa dễ dàng bằng đồ thị (Hình 1.9). Các thuật toán này cố gắng giữ nguyên cấu trúc nhiều nhất có thể (ví dụ như giữ các cụm phân biệt trong không gian đầu vào không chồng lấn lên nhau khi biểu diễn), nên ta có thể hiểu cách dữ liệu được tổ chức và xác định các khuôn mẫu ẩn trong dữ liệu.



Hình 1.9. Ví dụ trực quan hóa bằng t-SNE cho các cụm ngữ nghĩa[3]

Một tác vụ liên quan là *giảm chiều*, với mục tiêu là đơn giản hóa dữ liệu mà không làm mất quá nhiều thông tin. Một phương pháp khả thi là gộp các đặc trưng tương quan với nhau thành một. Ví dụ, quãng đường đi được của một chiếc xe có thể tương quan mạnh với tuổi thọ của chiếc xe đó, nên thuật toán giảm chiều sẽ gộp chúng lại thành một đặc trưng biểu diễn sự hao mòn của chiếc xe. Đây được gọi là *trích xuất đặc trưng* (*feature extraction*).

Mẹo

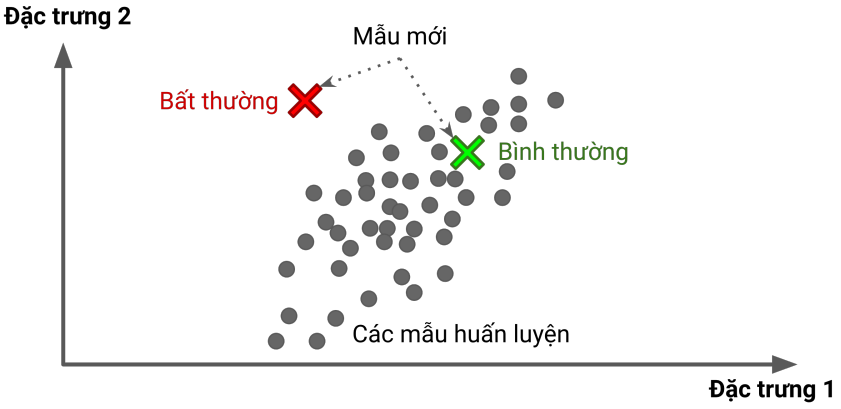
Giảm chiều dữ liệu huấn luyện trước khi đưa chúng vào một thuật toán Học Máy (ví dụ như một thuật toán học có giám sát) thường là một ý tưởng tốt. Thuật toán sẽ chạy nhanh hơn, dữ liệu sẽ chiếm ít dung lượng ổ cứng và bộ nhớ hơn, và trong một vài trường hợp có thể đem lại chất lượng tốt hơn.

Một tác vụ không giám sát quan trọng khác là *phát hiện bất thường*, ví dụ như phát hiện các giao dịch thẻ tín dụng bất thường để ngăn chặn sai phạm, bắt lỗi trong dây chuyền sản xuất, hoặc tự động loại bỏ các điểm ngoại lại trong tập dữ liệu trước khi đưa chúng vào các thuật toán học. Hệ thống được huấn luyện trên hầu hết các mẫu bình thường nên có thể nhận ra các mẫu này. Sau đó, khi thấy một mẫu mới, hệ thống có thể chỉ ra mẫu này là bình thường hay bất

[3] Có thể thấy rằng động vật và phương tiện giao thông nằm ở hai phía riêng biệt, và ngựa gần với hươu nhưng xa so với chim. Hình ảnh được tái tạo dưới sự cho phép của Richard Socher và các cộng sự, "Zero-Shot Learning Through Cross-Modal Transfer," *Proceedings of the 26th International Conference on Neural Information Processing Systems* 1 (2013): 935–943.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

thường (tham khảo Hình 1.10). Một tác vụ tương tự là *phát hiện tính mới*. Tác vụ này nhắm đến việc phát hiện các điểm không giống các điểm khác trong tập huấn luyện. Tác vụ này đòi hỏi một tập huấn luyện rất "sạch", không hề chứa các điểm mà thuật toán sẽ cần phải phát hiện. Ví dụ, nếu bạn có hàng nghìn ảnh chó, và 1% số ảnh đó là của giống chó Chihuahua, thuật toán phát hiện tính mới sẽ không xét các bức ảnh Chihuahua mới là ảnh có tính mới. Mặt khác, các thuật toán phát hiện bất thường vẫn có thể xem giống chó này là hiếm và khác các giống chó còn lại, nên có lẽ sẽ phân loại chúng là bất thường (ở đây không có ý kì thị Chihuahua).

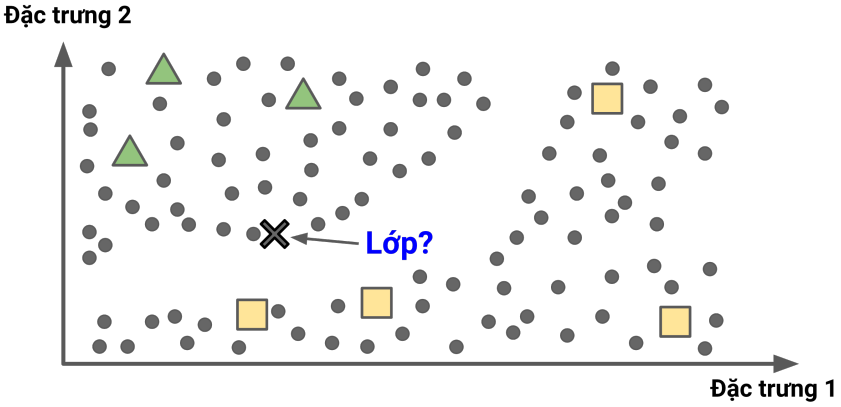


Hình 1.10. Phát hiện bất thường

Cuối cùng, một tác vụ không giám sát phổ biến khác là *học luật kết hợp (association rule learning)*, với mục tiêu là đào sâu vào lượng lớn dữ liệu để khám phá các mối quan hệ thú vị giữa các thuộc tính. Để lấy ví dụ, giả sử bạn sở hữu một siêu thị. Việc áp dụng luật kết hợp cho các hóa đơn bán hàng có thể hé lộ rằng người mua sốt BBQ và bim bim có xu hướng mua thêm thịt bò. Nhờ đó, bạn có thể đặt các mặt hàng này gần nhau.

Học bán giám sát

Sự tốn kém về thời gian lẫn chi phí của quá trình gán nhãn dữ liệu dẫn đến hệ quả là chỉ một phần nhỏ dữ liệu được gán nhãn. Các thuật toán có thể làm việc với tập dữ liệu được gán nhãn một phần được gọi là thuật toán *học bán giám sát* (*semisupervised learning* – Hình 1.11).



Hình 1.11. Học bán giám sát với hai lớp (hình tam giác và hình vuông): các mẫu chưa gán nhãn (hình tròn) giúp phân loại một đối tượng mới (hình chữ thập) vào lớp hình tam giác thay vì hình vuông, mặc dù nó gần với các hình vuông đã được gán nhãn hơn

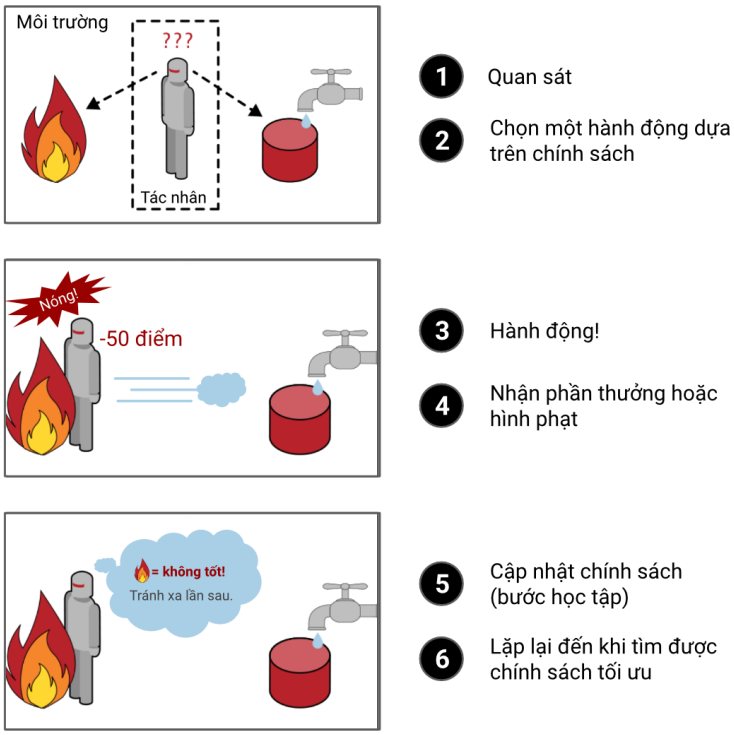
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các dịch vụ lưu trữ ảnh là một ví dụ tiêu biểu, điển hình như Google Photos. Sau khi tải lên tất cả ảnh gia đình của bạn lên đó, chúng sẽ tự động nhận diện cùng một người A xuất hiện trong các ảnh 1, 5 và 11, còn người B xuất hiện ở ảnh 2, 5 và 7. Đây là phần không giám sát của thuật toán (phân cụm). Giờ hệ thống chỉ cần bạn chỉ ra những người này là ai. Bằng cách thêm một nhãn cho mỗi người[4], hệ thống có thể gán tên cho tất cả mọi người trong ảnh, giúp việc tìm kiếm ảnh trở nên thuận tiện hơn.

Đa phần các thuật toán học bán giám sát là sự kết hợp giữa các thuật toán học không giám sát và có giám sát. Ví dụ, *mạng niềm tin sâu* (*deep belief network* – DBN) dựa trên các thành phần không giám sát có tên là *máy Boltzmann giới hạn* (*restricted Boltzmann machine* – RBM) chồng lên nhau. RBM được huấn luyện tuần tự theo cơ chế không giám sát, rồi sau đó toàn bộ hệ thống được tinh chỉnh bằng các kỹ thuật có giám sát.

Học Tăng cường

Học Tăng cường là phương pháp học có cấu trúc rất khác. Hệ thống học trong RL được gọi là *tác nhân* (*agent*). Nó có thể quan sát môi trường xung quanh, chọn và thực hiện các hành động, sau đó nhận về *điểm thưởng* – *reward* (hoặc *lượng phạt* – *penalty* dưới dạng điểm thưởng âm, như trong Hình 1.12). Hệ thống sau đó cần tự học chiến lược tốt nhất để nhận về nhiều điểm thưởng nhất qua thời gian, và chiến lược này được gọi là *chính sách* (*policy*). Một chính sách định nghĩa hành động mà tác nhân nên chọn khi ở trong một tình huống cụ thể.



Hình 1.12. Học Tăng cường

[4] Đó là khi hệ thống hoạt động hoàn hảo. Trong thực tế, dịch vụ có thể tạo nhiều cụm cho mỗi người hoặc đôi lúc nhầm lẫn giữa hai người nhìn giống nhau. Nên có thể ta sẽ cần cung cấp một vài nhãn cho mỗi người và xoá bớt một vài cụm một cách thủ công.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ví dụ, nhiều rô bốt được lập trình với thuật toán học tăng cường để học cách đi lại. AlphaGo của DeepMind cũng là một ví dụ về học tăng cường: nó xuất hiện trên trang nhất của các bản tin trong tháng 5 năm 2017 khi đánh bại nhà vô địch cờ vây thế giới Ke Jie (Kha Khiết). AlphaGo đã học được chính sách dẫn đến chiến thắng bằng cách phân tích hàng triệu ván cờ, rồi tự chơi rất nhiều ván với chính nó. Chú ý rằng AlphaGo không học trong khi thi đấu với Ke Jie mà chỉ đơn thuần áp dụng chính sách nó đã học được từ trước.

Học theo Batch và Học Trực tuyến

Một tiêu chí khác để phân loại các hệ thống Học Máy nằm ở việc chúng có thể liên tục học từ các dòng dữ liệu gia tăng hay không.

Học theo Batch

Các hệ thống *học theo batch* không có khả năng học gia tăng: chúng phải được huấn luyện bằng tất cả dữ liệu khả dụng. Nhìn chung thì việc này sẽ tốn rất nhiều thời gian và tài nguyên tính toán, nên quá trình huấn luyện thường diễn ra một cách ngoại tuyến. Đầu tiên, hệ thống được huấn luyện và hoàn tất việc học trước khi được triển khai thực tế. Trong quá trình triển khai thực tế, hệ thống chỉ áp dụng lại những gì đã được học. Đây được gọi là *học ngoại tuyến (offline learning)*. Nếu ta muốn một hệ thống học theo batch cập nhật thêm dữ liệu mới (ví dụ như một loại thư rác mới), ta cần huấn luyện một phiên bản mới của hệ thống từ đầu với toàn bộ dữ liệu (bao gồm cả dữ liệu mới và cũ), rồi thay hệ thống cũ bằng hệ thống mới.

May mắn là toàn bộ quá trình huấn luyện, đánh giá và triển khai một hệ thống Học Máy có thể được tự động hóa một cách khá dễ dàng (minh họa trong Hình 1.3), nên ngay cả một hệ thống học theo batch cũng có thể thích ứng với thay đổi. Ta chỉ cần cập nhật dữ liệu và huấn luyện một phiên bản mới lại từ đầu khi cần thiết.

Phương án này đơn giản và thường hoạt động tốt, nhưng huấn luyện trên toàn bộ dữ liệu có thể tốn hàng giờ, nên hệ thống thường được huấn luyện lại chỉ sau 24 giờ hoặc thậm chí sau mỗi tuần. Nếu hệ thống cần thích ứng với dữ liệu thay đổi nhanh (ví dụ như giá cổ phiếu), ta sẽ cần một giải pháp linh hoạt hơn.

Thêm vào đó, huấn luyện trên toàn bộ dữ liệu tiêu tốn rất nhiều tài nguyên tính toán (CPU, dung lượng bộ nhớ và ổ đĩa, I/O ổ đĩa và mạng, v.v.). Nếu có rất nhiều dữ liệu, việc tự động huấn luyện lại mô hình hàng ngày có thể tiêu tốn rất nhiều tiền. Nếu lượng dữ liệu có kích thước khổng lồ, học theo batch còn có thể trở nên bất khả thi.

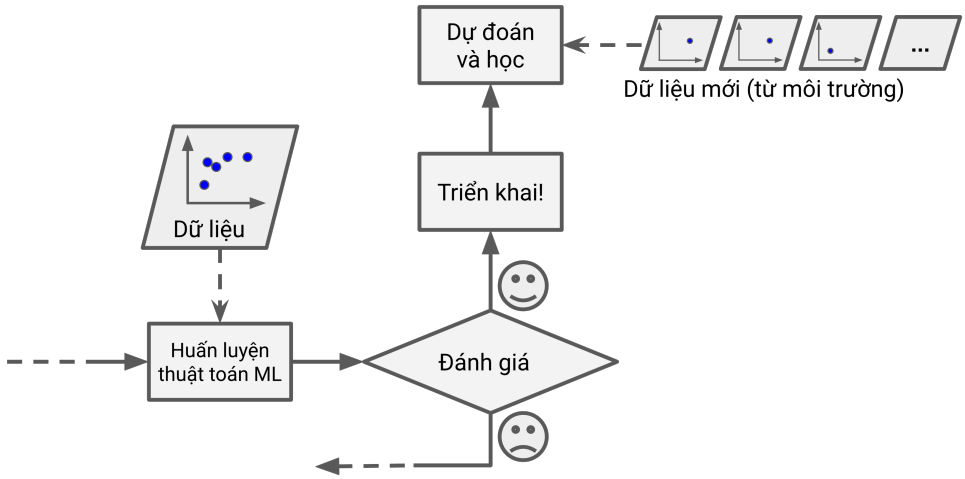
Cuối cùng, nếu hệ thống cần có khả năng học độc lập với tài nguyên hạn chế (ví dụ như ứng dụng điện thoại thông minh hoặc rover trên Sao Hoả), việc lưu trữ lượng lớn dữ liệu huấn luyện và sử dụng nhiều tài nguyên để huấn luyện hàng giờ mỗi ngày là không thể.

May mắn thay, chúng ta có một lựa chọn tốt hơn cho tất cả các trường hợp trên, đó là sử dụng các thuật toán có khả năng học gia tăng.

Học Trực tuyến

Trong *học trực tuyến* (*online learning*), ta huấn luyện mô hình một cách gia tăng bằng cách tuần tự truyền dữ liệu theo từng điểm dữ liệu hoặc theo lô nhỏ gọi là *mini-batch*. Mỗi bước học rất nhanh và không tốn nhiều tài nguyên, nên hệ thống có thể dễ dàng học với dữ liệu mới khi cần (tham khảo Hình 1.13).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.13. Trong học trực tuyến, mô hình được huấn luyện và triển khai thực tế, rồi tiếp tục học khi có dữ liệu mới

Học trực tuyến là giải pháp tốt cho các hệ thống nhận dữ liệu là các luồng liên tục (như giá cổ phiếu) và cần thích ứng với thay đổi một cách nhanh chóng hoặc độc lập. Phương pháp này cũng là lựa chọn tốt nếu tài nguyên tính toán bị giới hạn: một khi hệ thống đã học xong từ dữ liệu mới, dữ liệu này không còn cần thiết nữa và có thể được loại bỏ (trừ khi ta muốn quay lại trạng thái trước đó và "nạp lại" dữ liệu). Điều này giúp tiết kiệm rất nhiều dung lượng lưu trữ.

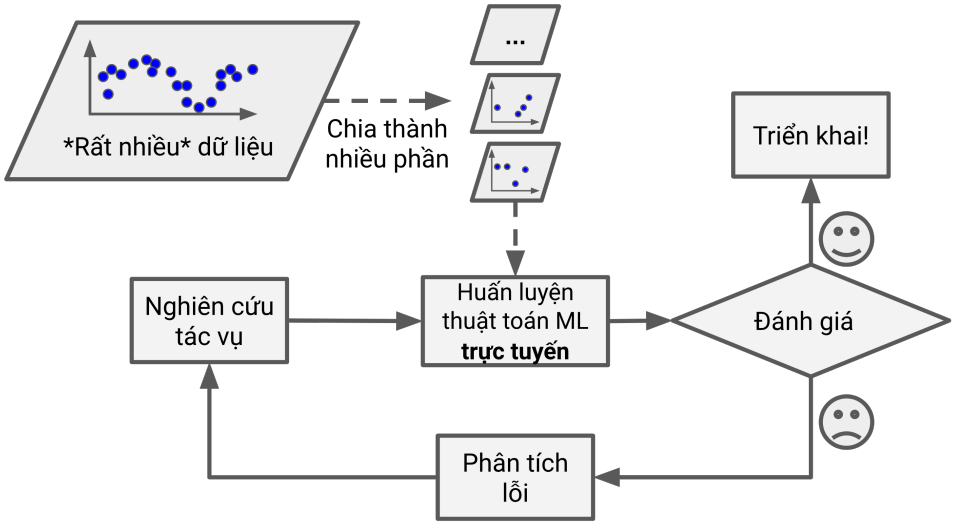
Các thuật toán học trực tuyến cũng có thể được sử dụng để huấn luyện hệ thống trên các tập dữ liệu khổng lồ và không nằm vừa trong bộ nhớ chính (đây được gọi là *học ngoài bộ nhớ chính – out-of-core learning*). Thuật toán chỉ nạp một phần dữ liệu, huấn luyện trên phần đó, rồi lặp lại quá trình cho tới khi đã chạy trên toàn bộ dữ liệu (tham khảo Hình 1.14).

Lưu ý

Học ngoài bộ nhớ chính thường được thực hiện ngoại tuyến (không được thực hiện trên hệ thống đang được triển khai), nên cái tên *học trực tuyến* có thể sẽ gây hiểu lầm. Hãy nghĩ về nó như là *học gia tăng*.

Một tham số quan trọng trong các hệ thống học trực tuyến là tốc độ thích ứng với dữ liệu đang thay đổi: tham số này được gọi là *tốc độ học* (*learning rate*). Nếu tốc độ học cao, hệ thống sẽ nhanh chóng thích ứng với dữ liệu mới, nhưng cũng sẽ có xu hướng quên dữ liệu cũ nhanh hơn (ta không muốn một bộ lọc thư rác chỉ lọc các loại thư rác nó thấy gần đây nhất). Ngược lại, nếu tốc độ học thấp, hệ thống sẽ có sức ỳ lớn hơn – tức sẽ học chậm hơn, nhưng cũng sẽ bớt nhạy cảm với nhiễu trong dữ liệu mới hoặc với chuỗi dữ liệu không có tính đại diện (ngoại lai).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.14. Sử dụng học trực tuyến cho các tập dữ liệu khổng lồ

Một thách thức lớn với học trực tuyến là nếu dữ liệu kém chất lượng được đưa vào mô hình, chất lượng của mô hình sẽ giảm. Nếu đó là một hệ thống trực tiếp (*live system*), khách hàng sẽ nhận ra. Ví dụ, dữ liệu kém chất lượng có thể đến từ một cảm biến bị hỏng trên rô bốt, hoặc từ một người đang cố gắng đánh lừa công cụ tìm kiếm để kết quả của họ nằm ở vị trí đầu trên trang tìm kiếm. Để giảm thiểu rủi ro này, ta cần giám sát hệ thống chặt chẽ và ngay lập tức dừng huấn luyện (và có thể quay lại trạng thái hoạt động tốt trước đó) khi phát hiện thấy chất lượng của mô hình giảm đi đáng kể. Ta cũng có thể giám sát dữ liệu đầu vào và xử lý dữ liệu bất thường (ví dụ, bằng cách sử dụng một thuật toán phát hiện bất thường).

Học dựa trên Mẫu và dựa trên Mô hình

Một cách khác để phân loại các thuật toán học máy là dựa vào cách chúng *khái quát hóa*. Phần lớn các tác vụ học máy sẽ xoay quanh việc đưa ra dự đoán. Nghĩa là với các mẫu dữ liệu huấn luyện cho trước, mô hình cần có khả năng đưa ra những dự đoán tốt (khái quát hóa tốt) đối với các mẫu dữ liệu mà nó chưa từng thấy. Việc có một phép đo phù hợp để đánh giá quá trình huấn luyện là tốt nhưng vẫn không đủ. Mục tiêu cuối cùng của mô hình là hoạt động tốt trên dữ liệu mới.

Có hai hướng tiếp cận chính để khái quát hóa: học dựa trên mẫu (*instance-based learning*) và học dựa trên mô hình (*model-based learning*).

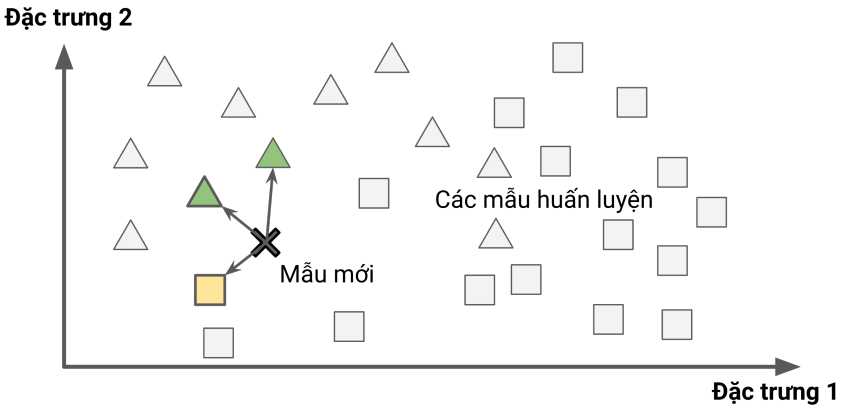
Học dựa trên Mẫu

Có lẽ phương thức học đơn giản nhất chính là học thuộc lòng. Nếu ta muốn tạo một bộ lọc thư rác theo cách này, nó chỉ cần lọc ra tất cả các thư giống hệt thư được người dùng đánh dấu trước đó là thư rác. Đây không phải là giải pháp tệ nhất, nhưng chắc chắn cũng không phải là giải pháp tốt nhất.

Thay vì chỉ đánh dấu các thư hoàn toàn giống thư rác, bộ lọc cũng có thể được lập trình để phát hiện các thư gần giống với các thư rác đã biết. Để làm vậy, ta cần có một *phép đo độ tương đồng* (*measure of similarity*) giữa hai lá thư. Một phép đo độ tương đồng (rất cơ bản) giữa hai lá thư là đếm số từ cùng xuất hiện trong cả hai. Hệ thống sẽ đánh dấu một bức thư là thư rác nếu nó có nhiều từ trùng lặp với một thư rác đã biết.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

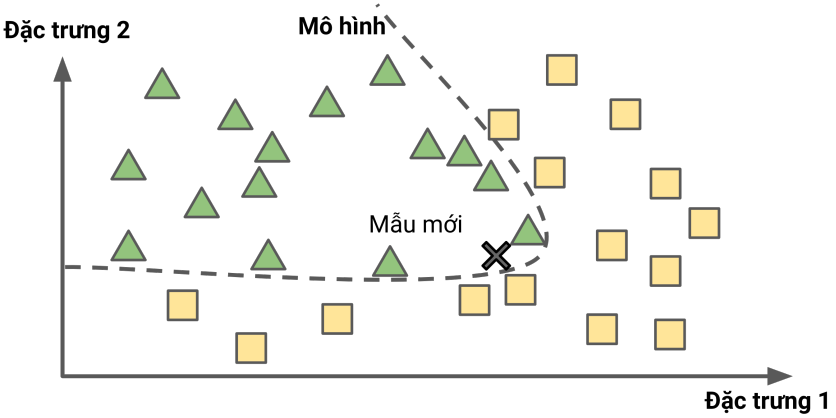
Phương pháp này được gọi là *học dựa trên mẫu* (*instance-based learning*): hệ thống học thuộc các mẫu dữ liệu, rồi khái quát hóa với các mẫu dữ liệu mới bằng việc sử dụng một phép đo độ tương đồng để so sánh với toàn bộ (hoặc một phần) dữ liệu đã học. Ví dụ, trong Hình 1.15, mẫu mới sẽ được gán nhãn là hình tam giác vì đa số các mẫu giống nó nhất đều thuộc lớp hình tam giác.



Hình 1.15. Học dựa trên mẫu

Học dựa trên Mô hình

Một cách khác để khái quát hóa từ dữ liệu cho trước là xây dựng một mô hình từ dữ liệu rồi dùng mô hình đó để đưa ra các *dự đoán*. Phương pháp này được gọi là *học dựa trên mô hình – model-based learning* (Hình 1.16).



Hình 1.16. Học dựa trên mô hình

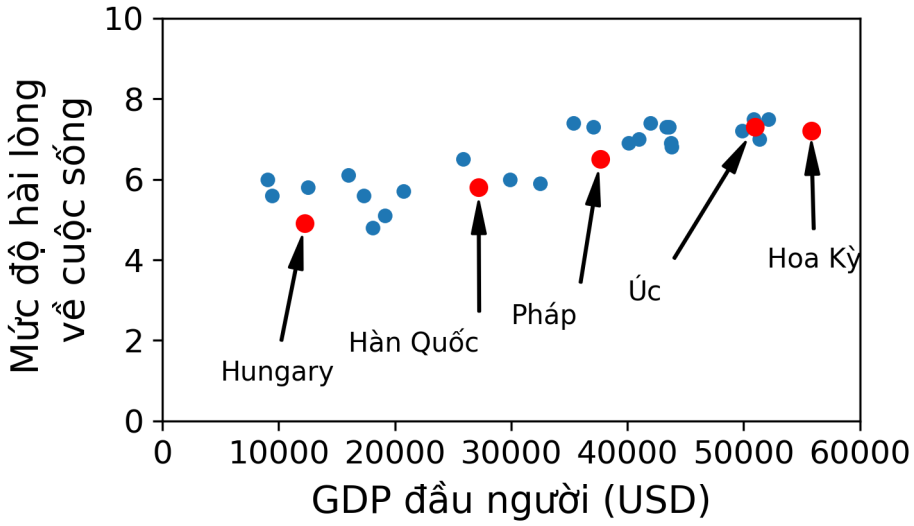
Ví dụ, giả sử ta muốn biết tiền có làm con người hạnh phúc hay không, nên ta tải xuống tập dữ liệu Better Life Index từ trang web của OECD, và thống kê về tổng sản phẩm quốc nội (GDP) bình quân đầu người từ trang web của IMF. Sau đó ta gộp hai bảng lại và sắp xếp theo GDP đầu người. Bảng 1.1 là một phần được trích từ bảng này.

Hãy vẽ đồ thị cho dữ liệu của các nước trên (Hình 1.17).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bảng 1.1. Tiền có giúp con người hạnh phúc không?

Quốc gia	GDP đầu người (USD)	Mức độ hài lòng về cuộc sống
Hungary	12,240	4.9
Hàn Quốc	27,195	5.8
Pháp	37,675	6.5
Úc	50,962	7.3
Hoa Kỳ	55,805	7.2



Hình 1.17. Bạn có nhận thấy xu hướng nào không?

Chúng ta có thể thấy một xu hướng ở đây! Mặc dù dữ liệu có *nhiễu* (tức là phần nào đó có tính ngẫu nhiên), ta vẫn nhận ra rằng mức độ hài lòng về cuộc sống dường như tăng lên tuyến tính với GDP đầu người. Do đó, ta quyết định mô hình hóa mức độ hài lòng bằng một hàm tuyến tính theo GDP đầu người. Bước này được gọi là *lựa chọn mô hình*: ta chọn một *mô hình tuyến tính* của mức độ hài lòng với duy nhất một thuộc tính là GDP đầu người (Phương trình 1.1).

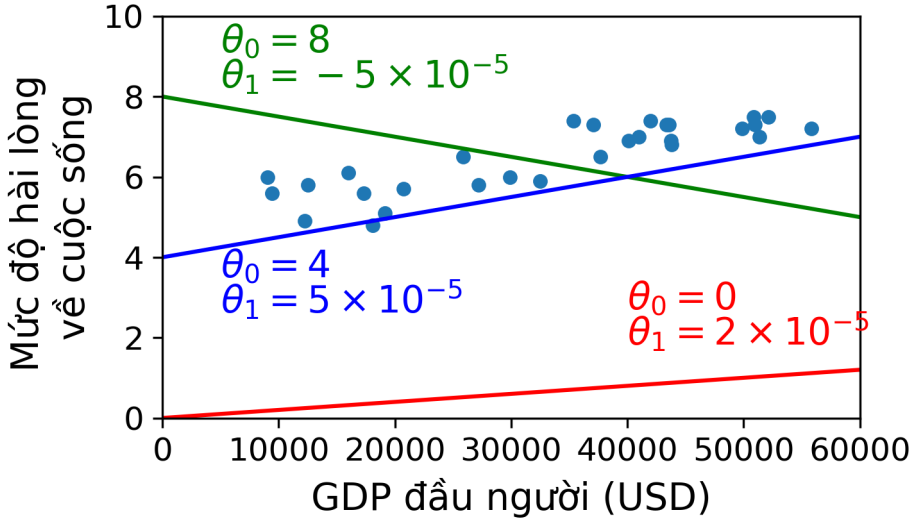
$$\texttt{life_satisfaction} = \theta_0 + \theta_1 \times \texttt{GDP_per_capita}$$

Phương trình 1.1. Một mô hình tuyến tính đơn giản

Mô hình này có hai *tham số (model parameter)* là θ_0 và θ_1.[5] Bằng cách thay đổi tham số này, mô hình có thể biểu diễn bất kỳ hàm tuyến tính nào, như trong Hình 1.18.

[5] Theo quy ước, chữ cái Hy Lạp θ (theta) thường được sử dụng để biểu diễn các tham số mô hình.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.18. Một vài hàm tuyến tính khả thi

Trước khi có thể sử dụng mô hình, ta cần chọn giá trị cho các tham số θ_0 và θ_1. Làm sao để biết giá trị nào giúp mô hình hoạt động tốt nhất? Để trả lời câu hỏi này, ta cần chỉ định một phép đo chất lượng. Ta có thể định nghĩa một *hàm lợi ích – utility function* (hoặc *hàm khớp – fitness function*) để đo độ *tốt* của mô hình, hoặc một hàm chi phí (*cost function*) để đo độ *tệ* của mô hình đó. Với các bài toán Hồi quy Tuyến tính, ta thường dùng một hàm chi phí để đo khoảng cách giữa dự đoán của mô hình và mẫu huấn luyện, với mục tiêu là cực tiểu hóa khoảng cách này.

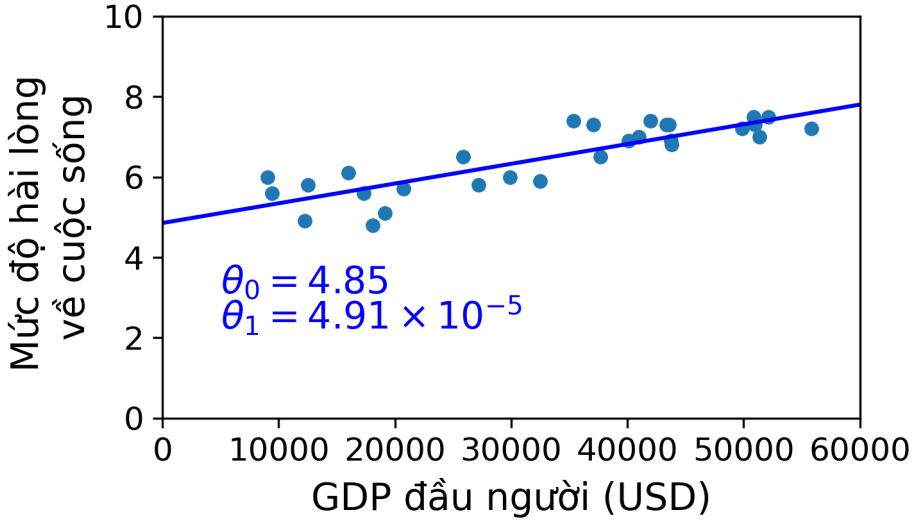
Đây là lúc thuật toán Hồi quy Tuyến tính phát huy tác dụng: chỉ cần đưa vào các mẫu huấn luyện và thuật toán sẽ tìm các tham số giúp mô hình tuyến tính khớp dữ liệu tốt nhất. Quá trình này được gọi là *huấn luyện* mô hình. Trong trường hợp của ta, thuật toán tìm được các giá trị tham số tối ưu là $\theta_0 = 4.85$ và $\theta_1 = 4.91 \times 10^{-5}$.

Lưu ý

Cùng một từ "mô hình" có thể hiểu là *kiểu mô hình* (như Hồi quy Tuyến tính), *kiến trúc mô hình được định nghĩa hoàn toàn* (*fully-specified model architecture* – như Hồi quy Tuyến tính một đầu vào một đầu ra), hoặc *mô hình đã được huấn luyện cuối cùng* (*final trained model*) và sẵn sàng đưa ra dự đoán (như Hồi quy Tuyến tính có một đầu vào và một đầu ra, với $\theta_0 = 4.85$ và $\theta_1 = 4.91 \times 10^{-5}$). Việc lựa chọn mô hình bao gồm chọn kiểu mô hình và định nghĩa toàn bộ kiến trúc của nó. Huấn luyện mô hình là chạy một thuật toán để tìm tham số mô hình sao cho dữ liệu được khớp tốt nhất (và hy vọng có thể dự đoán tốt trên dữ liệu mới).

Giờ mô hình đã khớp tốt nhất có thể với dữ liệu huấn luyện (với một mô hình tuyến tính), như có thể thấy trong Hình 1.19.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.19. Mô hình tuyến tính khớp dữ liệu huấn luyện tốt nhất

Cuối cùng thì mô hình đã sẵn sàng để đưa ra dự đoán. Ví dụ, giả sử ta muốn biết mức độ hạnh phúc của người Cộng hoà Síp, nhưng dữ liệu OECD lại không có câu trả lời. Rất may mắn, mô hình có thể đưa ra một dự đoán tốt: ta tra cứu GDP đầu người của Cộng hoà Síp và biết con số đó là 22,587, rồi áp dụng mô hình và tính được mức độ hài lòng về cuộc sống nằm đâu đó quanh mức $4.85 + 22{,}587 \times 4.91 \times 10^{-5} = 5.96$.

Để giúp mọi thứ trở nên thú vị hơn, Ví dụ 1.1 chứa mã nguồn Python để nạp, chuẩn bị dữ liệu[6], minh họa trực quan bằng đồ thị phân tán, rồi huấn luyện một mô hình tuyến tính và đưa ra dự đoán.[7]

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

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


# Prepare the data
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]

# Visualize the data
```

[6] Định nghĩa hàm `prepare_country_stats()` không được đưa ra ở đây (hãy xem Jupyter Notebook của chương này để biết chi tiết). Đó chỉ là một vài dòng mã pandas để gộp dữ liệu mức độ hài lòng từ OECD với dữ liệu GDP đầu người từ IMF.

[7] Nếu bạn chưa hiểu hết toàn bộ đoạn mã thì cũng không sao, chúng tôi sẽ trình bày về Scikit-Learn ở những chương tiếp theo.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

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

# Select a linear model
model = sklearn.linear_model.LinearRegression()

# Train the model
model.fit(X, y)

# Make a prediction for Cyprus
X_new = [[22587]]  # Cyprus's GDP per capita
print(model.predict(X_new)) # outputs [[ 5.96242338]]
```

Ví dụ 1.1. Huấn luyện và chạy mô hình tuyến tính bằng Scikit-Learn

Ghi chú

Nếu thay vào đó ta sử dụng thuật toán học dựa trên mẫu, ta sẽ nhận thấy rằng Slovenia có GDP đầu người gần giống Cộng hoà Síp nhất ($20,732), và vì dữ liệu OECD cho biết người Slovenia có mức độ hài lòng về cuộc sống là 5.7, ta có thể dự đoán rằng mức độ hài lòng của người Cộng hoà Síp là 5.7. Hai nước gần nhất tiếp theo là Bồ Đào Nha và Tây Ban Nha với mức độ hài lòng lần lượt là 5.1 và 6.5. Nếu lấy trung bình các giá trị trên, ta thu được 5.77, khá gần với dự đoán của thuật toán học dựa trên mô hình. Thuật toán đơn giản này được gọi là Hồi quy *k-Điểm Gần nhất – k-Nearest Neighbors regression* (trong ví dụ này thì $k = 3$).
Việc thay thế mô hình Hồi quy Tuyến tính bằng mô hình k-Điểm Gần nhất trong đoạn mã trên rất đơn giản, bạn chỉ cần thay hai dòng sau:

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

bằng hai dòng dưới đây:

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

Nếu không có vấn đề gì, mô hình sẽ đưa ra các dự đoán tốt. Còn nếu không, ta có thể cần thêm nhiều thuộc tính hơn (tỉ lệ có việc làm, sức khoẻ, ô nhiễm không khí, v.v.), thu thập thêm hoặc cải thiện chất lượng dữ liệu huấn luyện, hoặc có thể chọn một mô hình mạnh hơn (như Hồi quy Đa thức).

Tóm lại, ta đã:

- Nghiên cứu dữ liệu.
- Lựa chọn mô hình.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện (tức sử dụng thuật toán học để tìm kiếm các tham số mô hình sao cho hàm chi phí đạt giá trị nhỏ nhất).
- Và cuối cùng, áp dụng mô hình để đưa ra dự đoán trên dữ liệu mới (việc này được gọi là *suy luận – inference*), với hy vọng rằng mô hình này sẽ khái quát tốt.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Có thể nói đây là quy trình của một dự án Học Máy điển hình. Trong Chương 2 bạn sẽ được trải nghiệm thực tế với một dự án từ đầu tới cuối.

Cho tới giờ, chúng ta đã đề cập đến khá nhiều kiến thức nền tảng: giờ đây bạn đã biết Học Máy thật sự là gì, tại sao nó lại hữu ích, những loại hệ thống Học Máy phổ biến nhất, và quy trình làm việc của một dự án điển hình. Tiếp theo hãy xem xét những vấn đề có thể xảy ra trong quá trình học, gây ảnh hưởng đến độ chính xác của các dự đoán.

Những Thách thức Chính của Học Máy

Nói ngắn gọn, vì nhiệm vụ chính của ta là chọn và huấn luyện một thuật toán trên một tập dữ liệu, hai vấn đề có thể xảy ra là "thuật toán tệ" và "dữ liệu xấu". Hãy bắt đầu với các ví dụ của dữ liệu xấu.

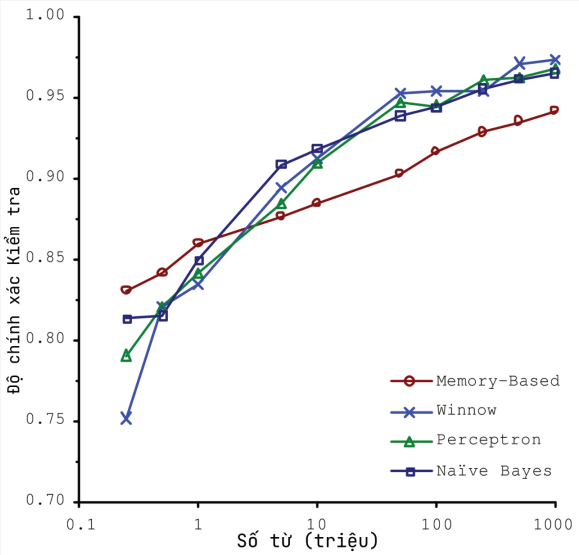
Không đủ Dữ liệu Huấn luyện

Để dạy một đứa trẻ quả táo là gì, ta chỉ cần chỉ vào quả táo và nói "quả táo" (có thể cần lặp lại quy trình này nhiều lần). Bây giờ đứa trẻ có thể nhận ra quả táo với đủ loại màu sắc và hình dạng. Quả thật xuất sắc.

Học Máy thì vẫn chưa đạt đến trình độ đó. Hầu hết các thuật toán Học Máy cần rất nhiều dữ liệu để có thể hoạt động hiệu quả. Ngay cả với những bài toán đơn giản, ta cũng cần đến hàng nghìn mẫu, và với những bài toán phức tạp như nhận diện ảnh hoặc giọng nói thì có thể lên đến hàng triệu mẫu (trừ khi ta có thể tận dụng một mô hình có sẵn).

Sự Hiệu quả Khó lý giải của Dữ liệu

Trong một bài báo nổi tiếng được xuất bản vào năm 2001, hai nhà nghiên cứu của Microsoft là Michele Banko và Eric Brill đã chỉ ra rằng các thuật toán Học Máy rất khác nhau, bao gồm cả những thuật toán khá đơn giản, hoạt động tốt gần như ngang nhau trong cùng một bài toán phức tạp là khử nhập nhằng ngôn ngữ tự nhiên (*natural language disambiguation*)[8] một khi chúng được cung cấp đủ dữ liệu (có thể thấy trong Hình 1.20).



Hình 1.20. Tầm quan trọng của dữ liệu so với thuật toán[9]

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

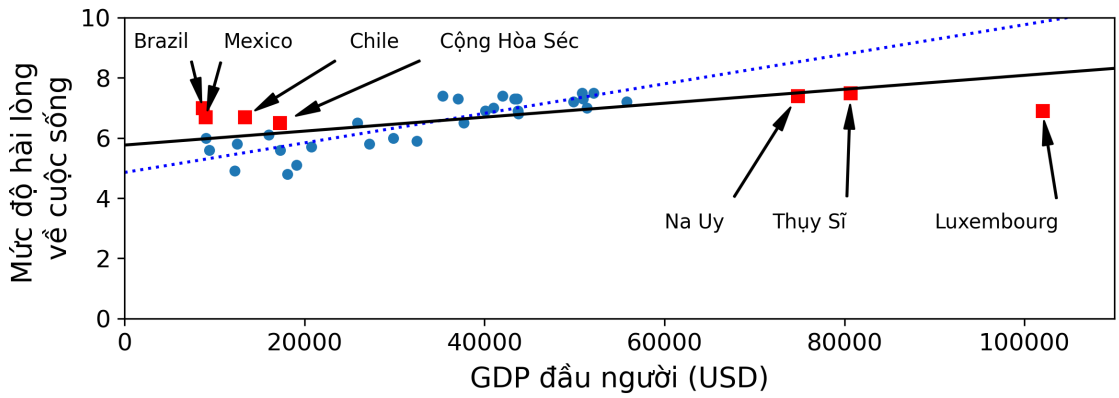
Theo lời của các tác giả, "những kết quả này cho thấy rằng chúng ta có thể cần xem xét lại sự đánh đổi giữa việc dành thời gian và tiền bạc để phát triển thuật toán và tập trung làm giàu kho ngữ liệu."

Ý tưởng về việc dữ liệu quan trọng hơn thuật toán trong các bài toán phức tạp đã được phổ biến hơn nữa bởi Peter Norvig và cộng sự trong bài báo với tiêu đề "The Unreasonable Effectiveness of Data", xuất bản vào năm 2009.[10] Tuy nhiên, cần lưu ý rằng các tập dữ liệu vừa và nhỏ vẫn rất phổ biến, và không phải lúc nào cũng có thể kiếm thêm dữ liệu huấn luyện một cách dễ dàng hoặc ít tốn kém. Vậy nên đừng vội bỏ rơi các thuật toán.

Dữ liệu Huấn luyện Không mang tính Đại diện

Để khái quát hóa tốt, điều quan trọng là dữ liệu huấn luyện phải có tính đại diện cho các trường hợp mới mà ta muốn khái quát hóa. Điều này đúng cho cả phương pháp học dựa trên mẫu hay học dựa trên mô hình.

Ví dụ, tập hợp các quốc gia mà lúc trước ta đã sử dụng để huấn luyện mô hình tuyến tính không có tính đại diện hoàn toàn, vì một vài quốc gia vẫn còn thiếu. Hình 1.21 minh họa dữ liệu sau khi các quốc gia còn thiếu đó được bổ sung.



Hình 1.21. Một mẫu huấn luyện có tính đại diện hơn

Nếu huấn luyện một mô hình tuyến tính trên tập dữ liệu mới này, ta sẽ thu được đường liền, trong khi mô hình cũ được biểu diễn bởi đường chấm. Có thể thấy rằng việc thêm một vài quốc gia bị thiếu không chỉ thay đổi đáng kể mô hình, mà còn giúp ta nhận ra rằng một mô hình tuyến tính đơn giản như vậy sẽ không bao giờ có thể hoạt động tốt. Có vẻ như các quốc gia rất giàu có không hạnh phúc hơn các quốc gia có thu nhập khá (thực chất họ còn có vẻ kém hạnh phúc hơn), và ngược lại, một số quốc gia nghèo lại có vẻ hạnh phúc hơn nhiều quốc gia giàu có.

Bằng cách huấn luyện trên một tập dữ liệu không có tính đại diện, các dự đoán của mô hình khó có thể chính xác, đặc biệt là đối với các nước rất nghèo và rất giàu.

Điều quan trọng ở đây là sử dụng tập huấn luyện có tính đại diện cho các trường hợp mà ta muốn khái quát hóa. Điều này nói dễ hơn làm: nếu lượng mẫu quá nhỏ, ta sẽ có *nhiễu do lấy*

[8] Ví dụ như việc lựa chọn giữa các từ "to,", "two,", hay "too," tùy thuộc vào ngữ cảnh.

[9] Hình ảnh được sao chép với sự cho phép của Michele Banko và Eric Brill, "Scaling to Very Very Large Corpora for Natural Language Disambiguation," *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics* (2001): 26–33.

[10] Peter Norvig và cộng sự, "The Unreasonable Effectiveness of Data," *IEEE Intelligent Systems* 24, no. 2 (2009): 8–12.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

mẫu (*sampling noise* – dữ liệu không mang tính đại diện do sự ngẫu nhiên), nhưng kể cả lượng mẫu lớn cũng có thể không có tính đại diện bởi sai sót trong phương pháp lấy mẫu. Hiện tượng thứ hai được gọi là *thiên kiến lấy mẫu* (*sampling bias*).

Các ví dụ về Thiên kiến lấy Mẫu

Có lẽ ví dụ nổi tiếng nhất về thiên kiến lấy mẫu xảy ra trong cuộc bầu cử tổng thống Mỹ năm 1936 giữa Landon và Roosevelt: tờ *Literary Digest* đã tiến hành một cuộc thăm dò khổng lồ bằng cách gửi thư cho khoảng 10 triệu người. Họ nhận được 2.4 triệu hồi âm và dựa vào đó để dự đoán với độ tin cậy cao rằng Landon sẽ nhận được 57% số phiếu bầu. Tuy nhiên, Roosevelt đã chiến thắng với 62% phiếu bầu. Lỗ hổng trong phương pháp lấy mẫu của *Literary Digest* là:

- Đầu tiên, để có được địa chỉ cho việc gửi phiếu thăm dò, tờ *Literary Digest* đã sử dụng danh bạ điện thoại, danh sách người đăng ký tạp chí, danh sách thành viên câu lạc bộ và những nguồn tương tự. Tất cả những danh sách này đều thiên về tầng lớp giàu có, những người khả năng cao sẽ bầu cho Đảng Cộng Hòa (mà Landon là người đại diện).
- Thứ hai, chỉ có ít hơn 25% số người được hỏi đã trả lời. Điều này lại gây ra thiên kiến lấy mẫu bằng cách loại trừ những người không quan tâm nhiều đến chính trị, những người không thích tờ *Literary Digest*, cũng như nhiều nhóm quan trọng khác. Đây là một dạng đặc biệt của thiên kiến lấy mẫu, thường được gọi là *thiên kiến không phản hồi* (*nonresponse bias*).

Một ví dụ khác: giả sử ta muốn xây dựng một hệ thống nhận diện các video nhạc funk. Một cách để xây dựng tập huấn luyện là tìm kiếm từ khóa "nhạc funk" trên Youtube và sử dụng các video thu được. Tuy nhiên, điều này giả định rằng công cụ tìm kiếm của Youtube trả về một tập hợp các video đại diện cho tất cả các video nhạc funk tồn tại trên nền tảng Youtube. Trên thực tế, kết quả tìm kiếm có thể thiên về những nghệ sĩ nổi tiếng (nếu bạn sống ở Brazil, đa số kết quả nhận được là các video "funk carioca", và chúng nghe hoàn toàn khác so với James Brown). Mặt khác, ta còn có thể làm gì hơn để thu được một tập huấn luyện lớn?

Dữ liệu Kém Chất lượng

Tất nhiên, nếu tập huấn luyện của bạn chứa đầy lỗi, điểm ngoại lai và nhiễu (chẳng hạn như do sai số đo lường), hệ thống sẽ gặp nhiều khó khăn trong việc phát hiện các khuôn mẫu ẩn, và ít có khả năng hoạt động tốt. Việc dành thời gian làm sạch dữ liệu huấn luyện thường rất cần thiết. Sự thật là đa số các nhà khoa học dữ liệu dành phần lớn thời gian của họ chỉ để làm việc đó. Những ví dụ sau đây là các trường hợp mà ta cần làm sạch dữ liệu huấn luyện:

- Nếu một số mẫu rõ ràng là ngoại lai, ta có thể đơn thuần loại bỏ chúng hoặc sửa lỗi một cách thủ công.
- Với những mẫu bị thiếu một số đặc trưng (ví dụ: 5% khách hàng của bạn không cung cấp tuổi của họ), ta cần quyết định giữa việc bỏ qua luôn thuộc tính này, bỏ qua những mẫu bị thiếu, điền vào các giá trị còn thiếu (ví dụ, bằng tuổi trung vị), hoặc huấn luyện hai mô hình: một mô hình với đặc trưng đó và một mô hình thì không.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các Đặc trưng Không liên quan

Có một câu nói là "rác vào, rác ra". Hệ thống chỉ có thể học được nếu dữ liệu huấn luyện chứa đủ các đặc trưng liên quan và không quá nhiều các đặc trưng không liên quan. Một phần quan trọng dẫn đến một dự án Học Máy thành công là xây dựng được một bộ đặc trưng tốt để huấn luyện. Quy trình này được gọi là *thiết kế đặc trưng* (*feature engineering*) và bao gồm các bước sau:

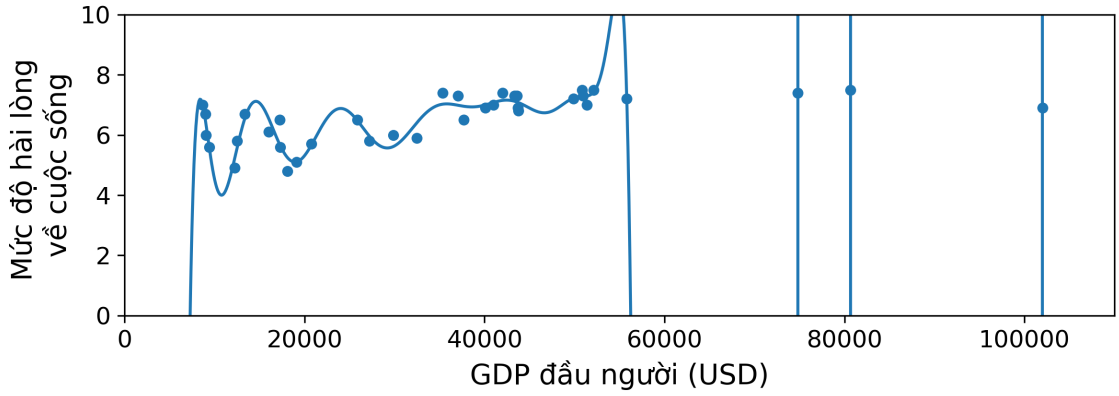
- *Lựa chọn đặc trưng* (lựa chọn những đặc trưng hữu ích nhất từ các đặc trưng sẵn có để huấn luyện)
- *Trích xuất đặc trưng* (kết hợp các đặc trưng sẵn có để tạo ra một đặc trưng hữu ích hơn, và như đã đề cập phía trên, các thuật toán giảm chiều có thể có ích)
- Tạo ra các đặc trưng mới bằng cách thu thập thêm dữ liệu

Ta đã xem xét nhiều ví dụ về dữ liệu xấu, giờ hãy cùng xem xét các ví dụ về thuật toán tệ.

Quá khớp Dữ liệu Huấn luyện

Giả sử bạn đang đi du lịch nước ngoài tham quan và bị tài xế taxi "chặt chém". Bạn có thể sẽ nghĩ rằng *tất cả* tài xế taxi ở đất nước đó đều là kẻ cắp. Con người thường có thói quen "vơ đũa cả nắm", và thật không may máy móc cũng có thể rơi vào cái bẫy tương tự nếu chúng ta không cẩn thận. Trong Học Máy, khái niệm này được gọi là *quá khớp* (*overfitting*), có nghĩa là mô hình hoạt động tốt trên dữ liệu huấn luyện nhưng lại không có tính khái quát hóa.

Hình 1.22 minh họa một mô hình đa thức bậc cao dự đoán mức độ hài lòng về cuộc sống đang quá khớp trên dữ liệu huấn luyện. Mặc dù nó hoạt động trên dữ liệu huấn luyện tốt hơn nhiều so với mô hình tuyến tính đơn giản, bạn có thật sự tin vào những dự đoán của nó không?



Hình 1.22. Quá khớp dữ liệu huấn luyện

Những mô hình phức tạp như mạng nơ-ron sâu có thể phát hiện được những quy luật không quá rõ ràng trong dữ liệu, nhưng nếu tập huấn luyện chứa nhiễu, hoặc nếu kích thước của tập này quá nhỏ (gây ra nhiễu do lấy mẫu), thì rất có thể mô hình sẽ lại phát hiện cả khuôn mẫu trong nhiễu. Rõ ràng những khuôn mẫu này sẽ không khái quát hóa cho các mẫu dữ liệu mới. Ví dụ, giả sử bạn huấn luyện mô hình dự đoán mức độ hài lòng về cuộc sống với nhiều đặc trưng hơn, bao gồm cả những đặc trưng không hữu ích như tên quốc gia. Trong trường hợp này, một mô hình phức tạp có thể phát hiện những quy luật như: tất cả các quốc gia trong tập huấn luyện

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

với tên tiếng Anh chứa chữ cái w đều có mức độ hài lòng về cuộc sống lớn hơn 7: New Zealand (7.3), Norway (Na Uy – 7.4), Sweden (Thụy Điển – 7.2), và Switzerland (Thụy Sĩ – 7.5). Bạn có chắc rằng "quy luật hài lòng w" này áp dụng cho Rwanda hoặc Zimbabwe không? Rõ ràng khuôn mẫu này chỉ xảy ra trong dữ liệu huấn luyện một cách tình cờ, nhưng mô hình không có cách nào để phân biệt được nó là thật hay chỉ đơn giản là nhiễu trong dữ liệu.

Lưu ý

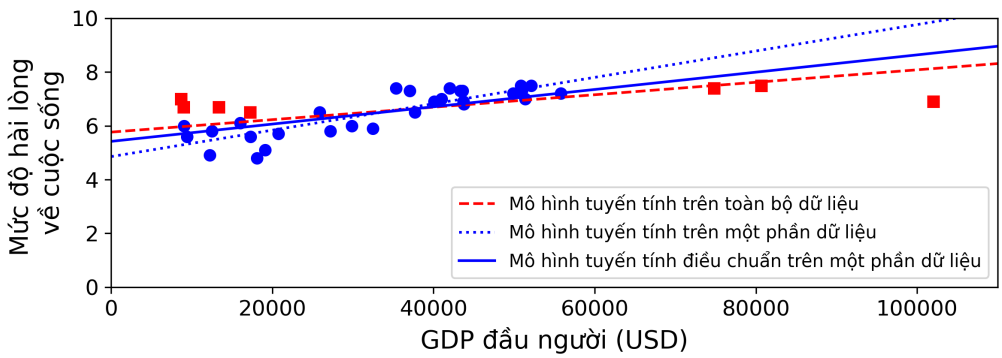
Quá khớp xảy ra khi mô hình quá phức tạp so với lượng mẫu và nhiễu của dữ liệu huấn luyện. Đây là những giải pháp để tránh vấn đề này:

- Đơn giản hóa mô hình bằng cách chọn mô hình có ít tham số hơn (ví dụ, một mô hình tuyến tính thay vì mô hình đa thức bậc cao), giảm số lượng thuộc tính trong dữ liệu huấn luyện, hoặc ràng buộc mô hình.
- Thu thập thêm dữ liệu huấn luyện.
- Giảm nhiễu trong dữ liệu huấn luyện (ví dụ như chỉnh sửa lỗi sai trong dữ liệu và loại bỏ những mẫu ngoại lai).

Việc ràng buộc một mô hình để làm cho nó đơn giản hơn và giảm nguy cơ quá khớp được gọi là *điều chuẩn* (*regularization*). Ví dụ, mô hình tuyến tính được định nghĩa ở trên có hai tham số là θ_0 và θ_1. Chúng cho phép thuật toán học với hai *mức độ tự do* (*degrees of freedom*) để thích ứng mô hình với dữ liệu huấn luyện: nó có thể điều chỉnh cả chiều cao (θ_0) và độ dốc (θ_1) của đường thẳng. Nếu chúng ta ép $\theta_1 = 0$, thuật toán chỉ còn một mức độ tự do và sẽ gặp khó khăn trong việc khớp dữ liệu: nó chỉ có thể di chuyển đường thẳng lên xuống sao cho càng gần với dữ liệu huấn luyện càng tốt, nên việc huấn luyện sẽ kết thúc gần điểm trung bình. Đây là một mô hình quá đơn giản! Nếu chúng ta cho phép thuật toán điều chỉnh θ_1 nhưng buộc nó có giá trị nhỏ thì thuật toán học sẽ có đâu đó từ một đến hai mức độ tự do. Kết quả là một mô hình đơn giản hơn mô hình có hai mức độ tự do, nhưng lại phức tạp hơn mô hình có một mức độ tự do. Chúng ta muốn tìm sự cân bằng giữa việc khớp tập huấn luyện một cách hoàn hảo và giữ mô hình đủ đơn giản để đảm bảo tính khái quát hóa.

Hình 1.23 biểu diễn ba mô hình. Đường chấm là mô hình ban đầu được huấn luyện chỉ trên các nước được đại diện bởi hình tròn (không bao gồm các nước đại diện bằng hình vuông), đường nét đứt là mô hình thứ hai được huấn luyện với tất cả các nước (hình tròn và hình vuông), và đường nét liền là mô hình được huấn luyện trên cùng dữ liệu với mô hình đầu tiên nhưng có điều chuẩn. Có thể thấy rằng việc điều chuẩn buộc mô hình có độ dốc nhỏ hơn: mô hình này không khớp với dữ liệu huấn luyện (hình tròn) tốt như mô hình đầu tiên, nhưng nó khái quát hóa tốt hơn trên các mẫu dữ liệu mới mà mô hình chưa thấy ở quá trình huấn luyện (hình vuông).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 1.23. Điều chuẩn giảm nguy cơ quá khớp

Mức độ điều chuẩn áp dụng trong quá trình học có thể được kiểm soát bởi một *siêu tham số* (*hyperparameter*). Một siêu tham số là một tham số của thuật toán chứ không phải của mô hình. Vì vậy, nó không bị ảnh hưởng bởi quá trình học. Giá trị của siêu tham số cần được đặt trước khi huấn luyện và sẽ được giữ nguyên trong suốt quá trình huấn luyện. Nếu ta đặt siêu tham số điều chuẩn quá cao, ta sẽ nhận được một mô hình gần như nằm ngang (độ dốc gần bằng 0). Khi đó, thuật toán gần như chắc chắn sẽ không quá khớp dữ liệu huấn luyện, nhưng sẽ khó để tìm được một mô hình tốt. Điều chỉnh siêu tham số là một phần quan trọng trong quá trình xây dựng một hệ thống Học Máy (ví dụ chi tiết sẽ được trình bày ở chương tiếp theo).

Dưới khớp Dữ liệu Huấn luyện

Như bạn có thể đoán được, *dưới khớp* (*underfitting*) ngược lại với quá khớp: nó xảy ra khi mô hình quá đơn giản để học được cấu trúc của dữ liệu. Ví dụ, một mô hình tuyến tính dự đoán mức độ hài lòng về cuộc sống sẽ dễ bị dưới khớp, đơn giản vì đời thực luôn phức tạp hơn mô hình. Vì vậy những dự đoán của nó chắc chắn sẽ không chính xác, kể cả với các mẫu huấn luyện.

Đây là những giải pháp chính để giải quyết vấn đề này:

- Chọn một mô hình mạnh hơn, với nhiều tham số hơn.
- Cung cấp đặc trưng tốt hơn cho thuật toán học (thiết kế đặc trưng).
- Giảm ràng buộc lên mô hình (ví dụ như giảm siêu tham số điều chuẩn).

Ôn tập

Cho tới giờ thì bạn đã biết khá nhiều về Học Máy. Tuy nhiên, việc học về quá nhiều khái niệm có thể khiến bạn hơi choáng ngợp, vì vậy hãy cùng lùi lại một chút và nhìn vào bức tranh tổng thể:

- Học Máy xoay quanh việc giúp máy móc thực hiện một số tác vụ tốt hơn bằng cách học từ dữ liệu thay vì phải lập trình các quy luật một cách tường minh.
- Có nhiều loại hệ thống ML: có hoặc không giám sát, học theo batch hay học trực tuyến, học dựa trên mẫu hay học dựa trên mô hình.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- Trong một dự án ML, ta thu thập dữ liệu huấn luyện và đưa chúng vào một thuật toán. Nếu thuật toán đó học dựa trên mô hình, nó sẽ điều chỉnh một vài tham số để khớp mô hình trên dữ liệu huấn luyện (tức để dự đoán tốt trên tập huấn luyện). Sau đó, ta hy vọng rằng mô hình cũng có thể dự đoán tốt trên dữ liệu mới. Nếu thuật toán đó học dựa trên mẫu, nó chỉ học thuộc lòng các mẫu dữ liệu và khái quát hóa cho các trường hợp mới bằng cách sử dụng một phép đo độ tương đồng để so sánh mẫu dữ liệu mới với mẫu dữ liệu đã học.
- Hệ thống sẽ hoạt động không hiệu quả nếu lượng dữ liệu huấn luyện quá nhỏ, hoặc nếu dữ liệu không mang tính đại diện, có nhiễu hay chứa các đặc trưng không liên quan ("rác vào, rác ra"). Cuối cùng, mô hình không nên quá đơn giản (gây dưới khớp) hoặc quá phức tạp (gây quá khớp).

Còn một chủ đề quan trọng nữa cần được đề cập: một khi mô hình đã được huấn luyện, ta không chỉ ngồi không và "hy vọng" nó sẽ khái quát cho các trường hợp mới. Ta cần phải đánh giá và tinh chỉnh mô hình nếu cần thiết. Hãy cùng xem xét cách để làm điều đó.

Kiểm tra và Đánh giá

Cách duy nhất để biết một mô hình khái quát hóa tốt đến đâu là thử nó với những mẫu dữ liệu mới. Một cách để thực hiện việc này là triển khai mô hình rồi giám sát chất lượng của nó. Cách này ổn, nhưng nếu mô hình hoạt động cực kỳ tệ, người dùng sẽ than phiền. Vì thế nên đây không phải là ý tưởng hay nhất.

Một lựa chọn tốt hơn là tách dữ liệu ra thành hai tập: *tập huấn luyện* (*training set*) và *tập kiểm tra* (*test set*). Như có thể đoán được từ cái tên, mô hình được huấn luyện trên tập huấn luyện và được kiểm tra trên tập kiểm tra. Tỉ lệ lỗi trên dữ liệu mới được gọi là *sai số khái quát – generalization error* (hoặc *sai số ngoài mẫu – out-of-sample error*) và bằng cách đánh giá mô hình trên tập kiểm tra, chúng ta sẽ ước lượng được giá trị sai số này. Giá trị này cho biết mô hình sẽ hoạt động tốt đến đâu trên dữ liệu mới.

Nếu sai số huấn luyện thấp (nghĩa là mô hình có ít lỗi trên tập huấn luyện) nhưng sai số khái quát lại cao, mô hình đã quá khớp dữ liệu huấn luyện.

Mẹo

Thông thường 80% dữ liệu được dùng để huấn luyện và 20% được *giữ lại* để kiểm tra. Tuy nhiên, việc này tuỳ thuộc vào kích cỡ của tập dữ liệu: nếu tập dữ liệu chứa 10 triệu mẫu, việc giữ lại 1% nghĩa là tập kiểm tra sẽ chứa 100,000 mẫu. Số lượng này có thể đã quá đủ để ước lượng tốt sai số khái quát.

Tinh Chỉnh Siêu Tham Số và Lựa Chọn Mô Hình

Việc đánh giá một mô hình khá đơn giản: chỉ cần dựa vào tập kiểm tra. Nhưng giả sử bạn đang lưỡng lự giữa hai loại mô hình (ví dụ, giữa một mô hình tuyến tính và một mô hình đa thức). Làm sao để đưa ra quyết định? Một lựa chọn là huấn luyện cả hai và so sánh khả năng khái quát của chúng trên tập kiểm tra.

Giờ hãy giả sử mô hình tuyến tính khái quát tốt hơn nhưng bạn muốn sử dụng thêm điều chuẩn để tránh quá khớp. Câu hỏi được đặt ra là ta chọn giá trị siêu tham số điều chuẩn như thế nào? Một giải pháp là huấn luyện 100 mô hình sử dụng 100 giá trị khác nhau cho siêu tham số này. Giả sử bạn đã tìm được siêu tham số tốt nhất sao cho mô hình có sai số khái quát thấp nhất,

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

chỉ ở mức 5%. Bạn triển khai mô hình này nhưng tiếc rằng nó không tốt như mong đợi với sai số ở mức 15%. Chuyện gì đã xảy ra?

Vấn đề là bạn đã tính toán sai số khái quát nhiều lần trên tập kiểm tra và cố gắng tìm các siêu tham số dẫn đến chất lượng tốt nhất *trên tập dữ liệu cố định đó*. Kết quả là mô hình khó có thể đạt được chất lượng tốt như mong đợi trên dữ liệu mới.

Giải pháp phổ biến cho vấn đề này là *kiểm định giữ lại* (*holdout validation*): ta đơn thuần giữ lại một phần của tập huấn luyện để đánh giá nhiều mô hình và chọn cái tốt nhất. Tập dữ liệu được giữ lại đó có tên là *tập kiểm định* (*validation set* – hoặc đôi khi được gọi là *tập phát triển* – *development set*/*dev set*). Cụ thể, ta huấn luyện nhiều mô hình với các siêu tham số khác nhau trên tập huấn luyện nhỏ hơn (do đã lấy ra tập kiểm định) và chọn mô hình hoạt động tốt nhất trên tập kiểm định. Sau khi hoàn tất việc kiểm định trên tập giữ lại, ta huấn luyện mô hình tốt nhất trên toàn bộ tập huấn luyện (bao gồm cả tập kiểm định) để thu được mô hình cuối. Cuối cùng, ta đánh giá mô hình này trên tập kiểm tra để ước lượng sai số khái quát.

Giải pháp này thường cho kết quả tốt. Tuy nhiên, nếu tập kiểm định quá nhỏ thì việc đánh giá mô hình sẽ không chính xác, và ta có thể vô tình chọn phải mô hình không tối ưu. Ngược lại, nếu tập kiểm định quá lớn, phần dữ liệu còn lại để huấn luyện sẽ nhỏ hơn rất nhiều so với tập huấn luyện đầy đủ. Vì sao điều này lại không tốt? Bởi vì mô hình cuối cùng sẽ được huấn luyện trên toàn bộ tập huấn luyện, việc so sánh giữa các mô hình được huấn luyện trên tập huấn luyện nhỏ hơn hẳn là không hợp lý. Để giải quyết vấn đề này, ta có thể dùng *kiểm định chéo* (*cross-validation*) với nhiều tập kiểm định nhỏ. Mỗi mô hình sẽ được đánh giá một lần trên mỗi tập kiểm định sau khi nó được huấn luyện trên phần dữ liệu còn lại. Bằng cách lấy trung bình các lần đánh giá, ta sẽ có một thước đo chính xác hơn nhiều cho chất lượng của mô hình. Tuy nhiên, phương pháp này có một hạn chế: thời gian huấn luyện sẽ tăng theo số tập kiểm định.

Dữ liệu không tương đồng

Trong một số trường hợp, ta có thể dễ dàng thu được một lượng lớn dữ liệu để huấn luyện, nhưng có lẽ chúng sẽ không đại diện hoàn toàn cho dữ liệu mà ta sẽ gặp phải trong thực tế. Giả sử bạn muốn tạo một ứng dụng điện thoại để chụp ảnh hoa và tự động xác định loài hoa. Bạn có thể dễ dàng tải xuống hàng triệu bức ảnh về hoa trên mạng, nhưng chúng sẽ không phải là đại diện hoàn hảo cho những bức ảnh sẽ được chụp bởi ứng dụng trên thiết bị di động. Có thể chỉ có 10,000 ảnh được chụp bằng ứng dụng đó. Trong trường hợp này, quy tắc quan trọng cần nhớ là tập kiểm định và tập kiểm tra phải mang tính đại diện cho dữ liệu trong thực tế càng nhiều càng tốt. Vì thế, hai tập này chỉ nên chứa các ảnh được chụp bằng ứng dụng: bạn có thể xáo trộn chúng và chia một nửa cho tập kiểm định, một nửa cho tập kiểm tra (đảm bảo rằng giữa hai tập không có mẫu nào bị trùng hoặc giống nhau). Nhưng sau khi huấn luyện mô hình bằng ảnh trên mạng, nếu bạn thấy chất lượng của mô hình trên tập kiểm định không tốt, bạn sẽ không biết được nguyên nhân là do mô hình đã quá khớp dữ liệu huấn luyện, hay chỉ là do sự không tương đồng giữa ảnh trên mạng và ảnh được chụp bằng ứng dụng di động. Một giải pháp là giữ lại một vài ảnh huấn luyện (ảnh trên mạng) cho một tập khác mà Andrew Ng gọi là *tập huấn luyện - phát triển* (*train-dev set*). Sau khi mô hình được huấn luyện (trên tập huấn luyện, *không phải* trên tập huấn luyện-phát triển), ta có thể đánh giá nó trên tập huấn luyện - phát triển. Nếu chất lượng tốt, ta biết được rằng mô hình không quá khớp tập huấn luyện. Nếu sau đó mô hình đạt chất lượng kém trên tập kiểm định, vấn đề chắc chắn đến từ việc dữ liệu không tương đồng. Bạn có thể thử giải quyết vấn đề này bằng cách tiền xử lý ảnh tải trên mạng để làm cho chúng giống với ảnh được chụp bởi ứng dụng và sau đó huấn luyện lại mô hình. Ngược lại, nếu mô hình hoạt động kém trên tập huấn luyện - phát triển, mô hình chắc hẳn đã quá khớp dữ liệu huấn luyện, vì vậy bạn nên đơn giản hóa hoặc điều chuẩn mô hình, thu thập thêm và làm sạch dữ liệu huấn luyện.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Định Lý Không Có Bữa Trưa Miễn Phí

Mô hình là một phiên bản được đơn giản hóa của các mẫu. Đơn giản hóa ở đây đồng nghĩa với việc loại bỏ các chi tiết thừa không có khả năng khái quát hóa cho các trường hợp mới. Để quyết định phần dữ liệu nào cần loại bỏ và phần dữ liệu nào cần giữ lại, bạn cần phải đặt ra các *giả định*. Ví dụ, một mô hình tuyến tính đưa ra giả định rằng dữ liệu có bản chất tuyến tính và khoảng cách giữa các mẫu và đường thẳng chỉ là nhiễu, và ta có thể bỏ qua khoảng cách đó mà không ảnh hưởng gì.

Trong một bài báo nổi tiếng năm 1996,[11] David Wolpert đã chứng minh rằng nếu bạn không đặt bất kỳ giả định nào về dữ liệu thì không có lý do gì để nói rằng mô hình này tốt hơn mô hình kia. Điều này được gọi là định lý *Không có Bữa trưa Miễn phí* (*No Free Lunch* – NFL). Với một số tập dữ liệu, mô hình tốt nhất là mô hình tuyến tính, trong khi với các tập dữ liệu khác, mô hình tốt nhất là một mạng nơ-ron. Không có mô hình nào được *tiên nghiệm* là sẽ hoạt động tốt hơn (do đó mà định lý có tên như trên). Cách duy nhất để biết chắc rằng mô hình nào tốt nhất là đánh giá tất cả các mô hình. Vì điều này là bất khả thi, trong thực tế, ta cần đưa ra một số giả định hợp lý về dữ liệu và chỉ đánh giá một số mô hình phù hợp. Ví dụ: đối với các tác vụ đơn giản, ta có thể đánh giá các mô hình tuyến tính với nhiều mức điều chuẩn khác nhau. Ngược lại, đối với một bài toán phức tạp, ta có thể đánh giá các mạng nơ-ron khác nhau.

Bài tập

Trong chương này, chúng tôi đã trình bày một số khái niệm quan trọng nhất trong Học Máy. Trong các chương tiếp theo, chúng ta sẽ đi sâu hơn và viết mã nhiều hơn, nhưng trước khi đi tiếp, hãy đảm bảo bạn biết câu trả lời cho các câu hỏi sau:

1. Định nghĩa của Học Máy là gì?
2. Bạn có thể liệt kê bốn loại bài toán mà Học Máy giải quyết tốt không?
3. Tập huấn luyện đã gán nhãn là gì?
4. Hai tác vụ học có giám sát phổ biến nhất là gì?
5. Bạn có thể liệt kê bốn tác vụ học không giám sát phổ biến không?
6. Bạn sẽ sử dụng loại thuật toán Học Máy nào để cho phép rô bốt đi lại trong các địa hình chưa biết?
7. Bạn sẽ sử dụng loại thuật toán nào để phân nhóm khách hàng thành nhiều nhóm?
8. Bạn sẽ đặt bài toán phát hiện thư rác là bài toán học có giám sát hay học không giám sát?
9. Hệ thống học trực tuyến là gì?
10. Thế nào là học ngoài bộ nhớ chính?
11. Loại thuật toán nào dựa vào phép đo độ tương đồng để đưa ra dự đoán?
12. Sự khác biệt giữa tham số mô hình và siêu tham số của thuật toán là gì?

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

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

13. Thuật toán học dựa trên mô hình đang tìm kiếm thứ gì? Chiến lược phổ biến nhất mà chúng sử dụng để thành công là gì? Chúng đưa ra dự đoán như thế nào?

14. Bạn có thể liệt kê bốn thách thức chính trong Học Máy không?

15. Nếu mô hình của bạn hoạt động tốt trên dữ liệu huấn luyện nhưng lại khái quát kém đối với dữ liệu mới, điều gì đang xảy ra? Bạn có thể liệt kê ba giải pháp khả thi cho vấn đề này không?

16. Tập kiểm tra là gì và tại sao bạn lại muốn sử dụng nó?

17. Mục đích của tập kiểm định là gì?

18. Tập huấn luyện - phát triển là gì, khi nào bạn cần sử dụng và làm thế nào để sử dụng nó?

19. Vấn đề gì có thể xảy ra nếu bạn tinh chỉnh siêu tham số bằng tập kiểm tra?

Lời giải cho những bài tập trên nằm ở Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 2

Dự án Học Máy từ Đầu tới Cuối

Trong chương này, ta đóng vai một nhà khoa học dữ liệu vừa được tuyển dụng bởi một công ty bất động sản, và sẽ làm việc trong một dự án mẫu từ đầu tới cuối.[1] Dưới đây là những bước chính mà ta sẽ thực hiện:

1. Nhìn vào bức tranh tổng thể.
2. Lấy dữ liệu.
3. Khám phá và trực quan hóa để hiểu dữ liệu.
4. Chuẩn bị dữ liệu cho các thuật toán Học Máy.
5. Lựa chọn và huấn luyện mô hình Học Máy.
6. Tinh chỉnh mô hình Học Máy.
7. Trình bày giải pháp.
8. Triển khai, giám sát và bảo trì hệ thống.

Làm việc với Dữ liệu Thực

Khi tìm hiểu về Học Máy, cách tốt nhất là thử nghiệm trên dữ liệu thực tế thay vì các tập dữ liệu nhân tạo. May mắn thay, hiện nay ta có thể sử dụng hàng ngàn tập dữ liệu mở trong hầu hết các lĩnh vực. Dưới đây là một số nguồn để lấy dữ liệu mà bạn có thể tham khảo:

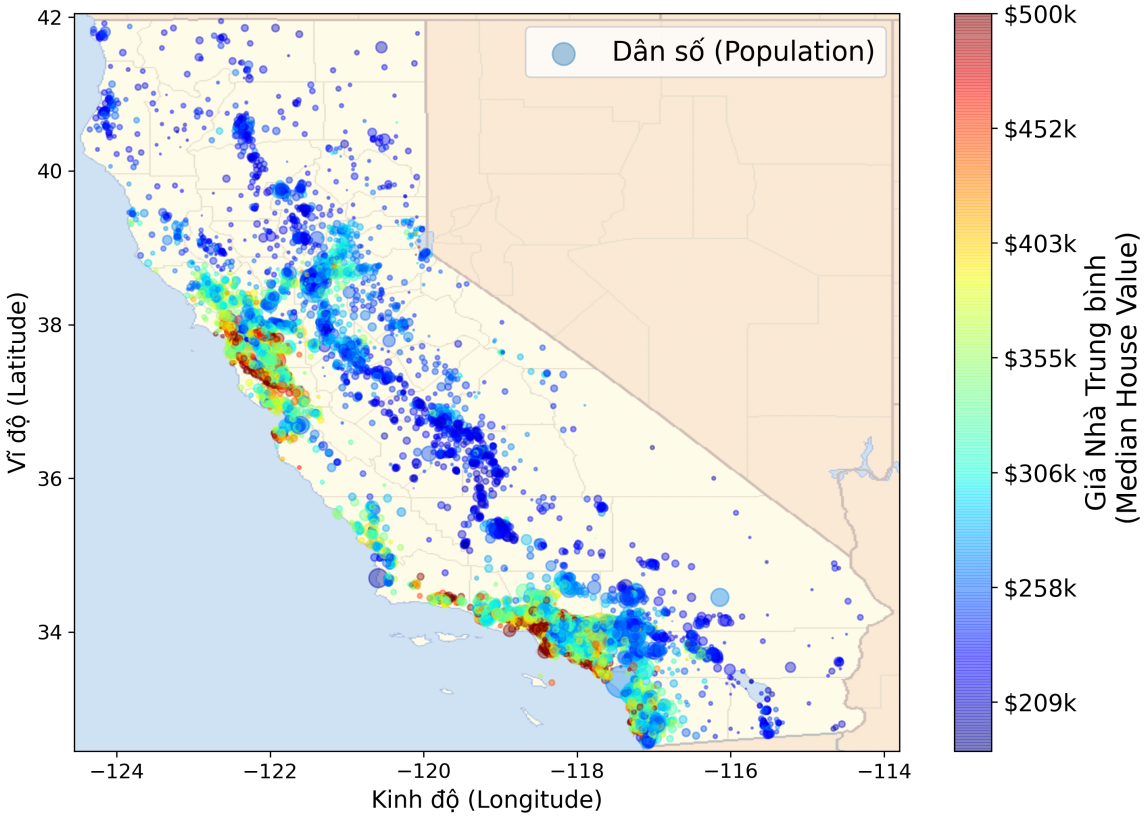
- Các kho dữ liệu mở phổ biến
  - Kho Dữ liệu Học Máy của trường UC Irvine
  - Các tập dữ liệu trên Kaggle
  - Các tập dữ liệu trên Amazon AWS
- Cổng thông tin các kho dữ liệu mở
  - Data Portals
  - OpenDataMonitor

[1] Dự án mẫu này là một ví dụ hư cấu với mục đích minh họa các bước chính trong một dự án Học Máy thực tế, không cung cấp kiến thức về bất động sản.

  - Quandl

- Các trang web khác
  - Danh sách các tập dữ liệu Học Máy trên Wikipedia
  - Quora.com
  - Subreddit về các tập dữ liệu

Trong chương này, chúng ta sẽ sử dụng tập dữ liệu Giá nhà ở California từ kho dữ liệu StatLib[2] (tham khảo Hình 2.1). Tập dữ liệu này dựa trên kết quả của cuộc điều tra dân số năm 1990 ở bang California. Nó không phản ánh đúng giá cả hiện tại (giá một ngôi nhà tốt ở Bay Area lúc đó vẫn còn phải chăng), nhưng lại sở hữu nhiều tính chất hỗ trợ việc học nên hãy cứ giả sử rằng tập dữ liệu này mới được thu thập gần đây. Để phù hợp với mục đích giảng dạy, chúng tôi đã thêm vào một vài thuộc tính hạng mục và loại bỏ một vài đặc trưng.



Hình 2.1. Giá các ngôi nhà ở California

Nhìn vào Bức tranh Tổng thể

Chào mừng bạn tới Tập đoàn Bất Động Sản Học Máy! Nhiệm vụ đầu tiên của bạn là sử dụng tập dữ liệu điều tra dân số của bang California để xây dựng một mô hình dự đoán giá nhà tại bang. Dữ liệu này bao gồm các số liệu như dân số, thu nhập trung vị và giá nhà trung vị cho từng *block* ở California. Block là đơn vị địa lý nhỏ nhất mà cục điều tra dân số Hoa Kỳ sử dụng

[2] Tập dữ liệu gốc lần đầu xuất hiện trong nghiên cứu của R. Kelley Pace và Ronald Barry, "Sparse Spatial Autoregressions," *Statistics & Probability Letters* 33, no. 3 (1997): 291–297.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

để công bố dữ liệu mẫu (một block thường có dân số từ 600 đến 3,000 người). Để thuận tiện, ta sẽ gọi các block là các "quận".

Mô hình của ta sẽ cần học từ tập dữ liệu này và dự đoán giá nhà trung vị cho một quận bất kỳ, dựa trên tất cả các số liệu đã cho.

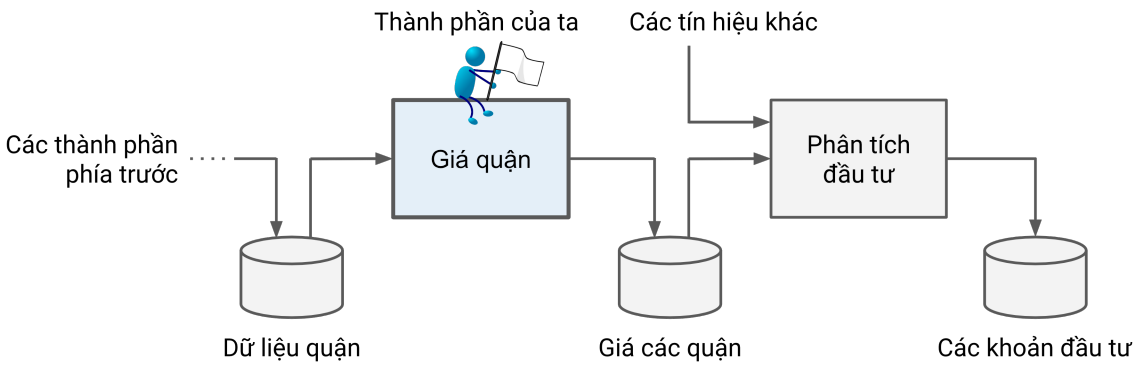
Mẹo

Là một nhà khoa học dữ liệu bài bản, việc đầu tiên bạn cần làm là tạo một danh mục công việc cho dự án Học Máy. Bạn có thể bắt đầu bằng cách tham khảo Phụ lục B, mô tả một khuôn mẫu phù hợp cho hầu hết các dự án Học Máy. Tuy nhiên, hãy điều chỉnh danh mục công việc theo nhu cầu cụ thể của bạn. Trong chương này, ta sẽ thực hiện nhiều việc trong danh mục, nhưng cũng sẽ bỏ qua một vài việc do chúng không cần giải thích gì thêm hoặc sẽ được giới thiệu trong các chương sau.

Phát biểu Bài toán

Trước hết, bạn nên hỏi sếp về mục tiêu của dự án. Xây dựng một mô hình có lẽ không phải là mục tiêu cuối cùng. Công ty sẽ sử dụng và thu lợi từ mô hình này như thế nào? Biết được mục tiêu rất quan trọng vì nó sẽ xác định cách bạn phát biểu bài toán, thuật toán được lựa chọn, phép đo chất lượng mà bạn sẽ sử dụng để đánh giá mô hình và công sức bạn cần bỏ ra để tinh chỉnh nó.

Sếp của bạn trả lời rằng đầu ra của mô hình (dự đoán giá nhà trung vị của một quận) sẽ được đưa tiếp vào một hệ thống Học Máy khác (tham khảo Hình 2.2), cùng với các tín hiệu khác.[3] Hệ thống này sẽ xác định liệu có đáng để đầu tư vào một khu vực hay không. Việc nắm được thông tin này rất quan trọng, vì nó ảnh hưởng trực tiếp tới doanh thu của công ty.



Hình 2.2. Một pipeline Học Máy cho đầu tư bất động sản

Pipeline

Một chuỗi các thành phần xử lý dữ liệu được gọi là một *pipeline* dữ liệu. Các pipeline này rất phổ biến trong các hệ thống Học Máy, vì ta cần áp dụng nhiều phép biến đổi và thao tác trên một lượng lớn dữ liệu.

Các thành phần này thường chạy một cách không đồng bộ. Mỗi thành phần lấy một lượng

[3] Mỗi thông tin được đưa vào một hệ thống Học Máy thường được gọi là một *tín hiệu* (*signal*), tham khảo theo lý thuyết thông tin của Claude Shannon, được ông nghiên cứu và phát triển tại phòng thí nghiệm Bell Labs để nâng cao chất lượng viễn thông. Lý thuyết của ông: ta muốn tỷ lệ tín hiệu trên nhiễu (signal-to-noise ratio) càng cao càng tốt.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

lớn dữ liệu, xử lý và đưa kết quả tới một kho dữ liệu khác. Một lát sau, thành phần tiếp theo trong pipeline sẽ lấy dữ liệu này ra và tiếp tục xử lý. Mỗi thành phần đều hoạt động một cách khép kín: các thành phần chỉ giao tiếp thông qua kho lưu trữ dữ liệu. Điều này giúp việc vận hành hệ thống đơn giản hơn (với sự trợ giúp của lưu đồ dữ liệu) và các nhóm khác nhau có thể tập trung vào các thành phần khác nhau. Hơn nữa, nếu xảy ra lỗi ở một thành phần, các thành phần phía sau vẫn có thể tiếp tục chạy bình thường (ít nhất là trong một khoảng thời gian) bằng cách sử dụng đầu ra cuối cùng từ thành phần bị lỗi. Điều này giúp cho kiến trúc hệ thống khá ổn định.

Mặt khác, thành phần bị lỗi có thể không được phát hiện một cách nhanh chóng nếu việc giám sát không được thực hiện đúng cách. Dữ liệu sẽ cũ dần và chất lượng tổng thể của hệ thống cũng sẽ giảm.

Câu hỏi tiếp theo cho sếp là về giải pháp hiện tại (nếu có). Tình hình hiện tại thường sẽ cung cấp một cách đối chiếu chất lượng, đồng thời giúp ta hiểu thêm về các phương pháp giải quyết vấn đề. Sếp của bạn trả lời rằng giá nhà ở của một quận hiện được các chuyên gia ước lượng một cách thủ công: một nhóm thu thập các thông tin mới nhất về quận đó và khi không có giá nhà trung vị, họ ước lượng nó bằng các quy tắc phức tạp.

Cách này rất tốn kém và mất thời gian, và các giá trị ước lượng thường không chính xác. Trong trường hợp tìm được giá nhà trung vị thực tế, họ nhận thấy rằng các ước lượng thường sai lệch hơn 20%. Do đó, công ty cho rằng sẽ rất hữu ích nếu huấn luyện được một mô hình dự đoán giá nhà trung vị của một quận, khi biết các dữ liệu khác về quận đó. Dữ liệu điều tra dân số được coi là một tập dữ liệu tuyệt vời để sử dụng cho mục đích này, vì nó bao gồm giá nhà trung vị của hàng nghìn quận, cũng như các dữ liệu khác.

Với tất cả những thông tin trên, bây giờ bạn đã sẵn sàng để bắt đầu thiết kế hệ thống của mình. Đầu tiên, bạn cần xác định bài toán: đây là bài toán học có giám sát, học không giám sát, hay học tăng cường? Đây là bài toán phân loại, hồi quy, hay một bài toán khác? Bạn nên sử dụng kỹ thuật học theo batch hay học trực tuyến? Trước khi đọc tiếp, bạn đọc hãy tạm dừng một chút và cố gắng tự trả lời những câu hỏi này.

Bạn đã tìm thấy câu trả lời chưa? Có thể thấy, rõ ràng đây là một bài toán học có giám sát điển hình, vì bạn có một tập dữ liệu huấn luyện đã được *gán nhãn* (mỗi mẫu có một giá trị nhãn tương ứng, trong trường hợp này là giá nhà ở trung vị của quận đó). Đây cũng là một bài toán hồi quy điển hình, vì bạn được yêu cầu dự đoán một giá trị. Cụ thể hơn, đây là bài toán *đa hồi quy* (*multiple regression*), vì hệ thống sẽ sử dụng nhiều đặc trưng để đưa ra dự đoán (như dân số của quận, trung vị thu nhập, v.v.). Đây cũng là bài toán *hồi quy đơn biến* (*univariate regression*), vì ta chỉ dự đoán một giá trị cho một quận. Nếu ta dự đoán nhiều giá trị cho một quận, thì nó sẽ là bài toán *hồi quy đa biến* (*multivariate regression*). Cuối cùng, do không có luồng dữ liệu liên tục tới hệ thống nên ta sẽ không cần phải điều chỉnh để thích nghi với dữ liệu một cách nhanh chóng. Đồng thời, dữ liệu đủ nhỏ để đưa vào bộ nhớ, nên ta có thể đơn thuần sử dụng phương pháp học theo batch.

Mẹo

Nếu lượng dữ liệu quá lớn, ta có thể chia nhỏ việc học theo batch ra các server (sử dụng kỹ thuật MapReduce) hoặc sử dụng phương pháp học trực tuyến.

Lựa chọn Phép đo Chất lượng

Bước tiếp theo là lựa chọn một phép đo chất lượng phù hợp. Một phép đo chất lượng điển hình trong bài toán hồi quy là Căn bậc hai Trung bình Bình phương Sai số (Root Mean Square Error

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

– RMSE). Phép đo này cho biết sai số mà hệ thống thường phạm phải khi đưa ra dự đoán, với trọng số lớn cho những sai số lớn. Phương trình 2.1 trình bày công thức toán học để tính giá trị RMSE.

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

Phương trình 2.1. Căn bậc hai Trung bình Bình phương Sai số (RMSE)

Ký hiệu

Phương trình trên chứa một vài ký hiệu phổ biến trong Học Máy mà ta sẽ sử dụng trong suốt cuốn sách này:

- m là số lượng mẫu trong tập dữ liệu đang được dùng để tính giá trị RMSE.
  - Ví dụ, nếu ta đang ước lượng giá trị RMSE trên tập đánh giá gồm 2,000 quận, khi đó $m = 2,000$.
- $\mathbf{x}^{(i)}$ là một vector chứa tất cả các giá trị đặc trưng (ngoại trừ nhãn) của mẫu thứ i trong tập dữ liệu, với $y^{(i)}$ là nhãn tương ứng (giá trị đầu ra tương ứng cho mẫu đó).
  - Ví dụ, nếu quận đầu tiên trong tập dữ liệu có kinh độ –118.29°, vĩ độ 33.91°, có 1,416 cư dân với thu nhập trung vị là 38,372 đô la và giá nhà ở trung vị là 156,400 đô la (tạm bỏ qua các đặc trưng khác), thì:

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1{,}416 \\ 38{,}372 \end{pmatrix}$$

    và:

$$y^{(1)} = 156{,}400$$

- $\mathbf{X}$ là một ma trận chứa tất cả các giá trị đặc trưng (ngoại trừ nhãn) của tất các mẫu trong tập dữ liệu. Mỗi hàng tương ứng với một mẫu, và hàng thứ i tương ứng với chuyển vị của $\mathbf{x}^{(i)}$, được ký hiệu là $(\mathbf{x}^{(i)})^\top$.[4]
  - Ví dụ, nếu quận đầu tiên được mô tả như trên thì ma trận $\mathbf{X}$ sẽ có dạng như sau:

$$\mathbf{X} = \begin{pmatrix} \left(\mathbf{x}^{(1)}\right)^\top \\ \left(\mathbf{x}^{(2)}\right)^\top \\ \vdots \\ \left(\mathbf{x}^{(1999)}\right)^\top \\ \left(\mathbf{x}^{(2000)}\right)^\top \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

- h là hàm dự đoán của hệ thống, cũng được gọi là một *giả thuyết* (*hypothesis*). Khi hệ thống nhận vector đặc trưng $\mathbf{x}^{(i)}$ của một mẫu, nó sẽ đưa ra giá trị dự đoán $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$ cho mẫu đó ($\hat{y}$ được đọc là "y-mũ").

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

  - Ví dụ, nếu hệ thống dự đoán giá nhà trung vị trong quận đầu tiên là 158,400 đô la, thì khi đó $\hat{y}^{(1)} = h(\mathbf{x}^{(1)}) = 158{,}000$. Sai số dự đoán cho quận này là $\hat{y}^{(1)} - y^{(1)} = 2{,}000$.

- RMSE$(\mathbf{X}, h)$ là hàm chi phí của giả thuyết h trên toàn bộ tập dữ liệu.

Chúng ta sẽ sử dụng phông chữ viết thường in nghiêng cho giá trị vô hướng (ví dụ m hay $y^{(i)}$) và tên hàm (ví dụ h), phông chữ viết thường in đậm cho vector (ví dụ $\mathbf{x}^{(i)}$), và phông chữ viết hoa in đậm cho ma trận (ví dụ $\mathbf{X}$).

Mặc dù RMSE nhìn chung khá phổ biến cho bài toán hồi quy, trong một vài trường hợp ta có thể muốn lựa chọn một phép đo khác. Ví dụ, giả sử tập dữ liệu có nhiều quận ngoại lai. Trong trường hợp này, bạn có thể cân nhắc sử dụng *trung bình sai số tuyệt đối* (*mean absolute error* – MAE, còn được gọi là trung bình độ lệch tuyệt đối; tham khảo Phương trình 2.2):

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^{m} \left| h\left(\mathbf{x}^{(i)}\right) - y^{(i)} \right|$$

Phương trình 2.2. Trung bình Sai số Tuyệt đối (MAE)

Cả RMSE và MAE đều được sử dụng để đo khoảng cách giữa hai vector: vector dự đoán và vector giá trị mục tiêu. Cũng có nhiều phép đo khoảng cách (hoặc các *chuẩn – norm*) khác:

- Việc tính căn bậc hai của tổng bình phương (RMSE) tương ứng với *chuẩn Euclid* (*Euclidean norm*): đây là chính là khái niệm quen thuộc về khoảng cách. Nó cũng được gọi là *chuẩn* ℓ_2, được ký hiệu là $\|\cdot\|_2$ (hay đơn giản là $\|\cdot\|$).
- Việc tính tổng các trị tuyệt đối (MAE) tương ứng với *chuẩn* ℓ_1, được ký hiệu là $\|\cdot\|_1$. Phép đo này đôi khi được gọi là *chuẩn Manhattan* (*Manhattan norm*) vì nó đo khoảng cách giữa hai điểm trong thành phố khi ta chỉ có thể di chuyển vuông góc dọc theo các lô nhà trong thành phố.
- Tổng quát hơn, *chuẩn* ℓ_k của một vector $\mathbf{v}$ chứa n phần tử được định nghĩa là

  $$\|\mathbf{v}\| = \left(|v_0|^k + |v_1|^k + \ldots + |v_n|^k\right)^{\frac{1}{k}}.$$

  ℓ_0 tính số lượng các giá trị khác không trong vector và ℓ_∞ trả về giá trị tuyệt đối lớn nhất trong vector.
- Chỉ số của chuẩn càng cao, chuẩn đó càng tập trung vào các giá trị lớn và bỏ qua những giá trị nhỏ. Đây là lý do tại sao phép đo RMSE lại nhạy với các điểm ngoại lai hơn so với phép đo MAE. Nhưng khi các điểm ngoại lai cực hiếm gặp (như trong đường cong hình chuông), phép đo RMSE lại tốt hơn và được sử dụng phổ biến hơn.

Kiểm tra các Giả định

Cuối cùng, việc liệt kê và kiểm tra các giả định đã được đưa ra (bởi bạn hoặc người khác) là một thói quen tốt. Việc này giúp ta sớm phát hiện các vấn đề nghiêm trọng. Ví dụ: giá nhà ở một quận mà hệ thống dự đoán sẽ được đưa vào một hệ thống Học Máy khác và ta giả định rằng những giá trị này vẫn được sử dụng ở định dạng số. Nhưng điều gì sẽ xảy ra nếu hệ thống

[4] Nhắc lại: phép chuyển vị biến một vector cột thành một vector hàng (và ngược lại).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

tiếp theo chuyển đổi giá nhà ở thành các hạng mục (ví dụ: "rẻ", "trung bình" hoặc "đắt") và sau đó sử dụng các hạng mục đó thay vì các con số ban đầu? Trong trường hợp này, việc dự đoán chính xác mức giá không còn quan trọng nữa, mà hệ thống của bạn chỉ cần chọn đúng hạng mục. Khi đó, bài toán nên được xem là một bài toán phân loại, chứ không phải là bài toán hồi quy. Ta không muốn phát hiện ra điều này quá muộn sau khi đã dành nhiều tháng để xây dựng một hệ thống hồi quy.

May thay, sau khi nói chuyện với nhóm phụ trách hệ thống tiếp theo, bạn biết chắc rằng họ thực sự cần giá nhà ở dạng số, thay vì các hạng mục. Tuyệt! Bạn đã được bật đèn xanh và có thể bắt đầu lập trình ngay bây giờ!

Thu thập Dữ liệu

Đã tới lúc bắt tay vào làm việc. Đừng chần chừ mở laptop và bắt đầu với các mã nguồn tham khảo trong Jupyter Notebook dưới đây. Bạn có thể xem toàn bộ Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Khởi tạo Môi trường Làm việc

Đầu tiên ta cần cài đặt Python. Nhiều khả năng Python đã được cài sẵn trong máy của bạn. Nếu chưa, bạn có thể tải Python tại *https://www.python.org/*.[5]

Tiếp theo, bạn cần tạo một thư mục làm việc dành cho mã nguồn dự án Học Máy và các tập dữ liệu. Hãy mở cửa sổ dòng lệnh và nhập các lệnh (sau ký tự $) sau:

```
$ export ML_PATH="$HOME/ml"      # You can change the path if you prefer
$ mkdir -p $ML_PATH
```

Ta sẽ cần một số mô-đun Python: Jupyter, Numpy, Pandas, Matplotlib, và Scikit-Learn. Nếu bạn đã cài đặt Jupyter cũng như các mô-đun còn lại, hãy tới thẳng phần Tải Dữ liệu. Nếu chưa, có nhiều cách để cài đặt chúng (cùng với các mô-đun phụ thuộc). Bạn có thể dùng hệ thống cài đặt của hệ điều hành (ví dụ, apt-get của Ubuntu, hoặc MacPorts hay Homebrew trên macOS), cài đặt một phần mềm Python cho khoa học như Anaconda và sử dụng hệ thống quản lý gói thư viện kèm theo, hoặc đơn thuần dùng hệ thống quản lý gói thư viện pip của Python, mặc định đi kèm khi cài đặt Python (từ phiên bản Python 2.7.9).[6] Ta có thể kiểm tra liệu pip đã được cài đặt bằng câu lệnh sau:

```
$ python3 -m pip --version
pip 19.3.1 from [...]/lib/python3.7/site-packages/pip (python 3.7)
```

Hãy đảm bảo rằng phiên bản mới nhất của pip đã được cài đặt. Để nâng cấp pip, hãy nhập câu lệnh sau (phiên bản được cập nhật có thể sẽ khác):[7]

[5] Chúng tôi khuyến nghị cài đặt phiên bản Python 3 mới nhất. Bạn cũng có thể sử dụng Python 2.7+, nhưng phiên bản này không còn được bảo trì và phát triển. Hầu hết các thư viện mã nguồn lớn đều sẽ ngừng việc hỗ trợ Python 2. Vì vậy, bạn nên chuyển sang phiên bản Python 3 càng sớm càng tốt.

[6] Chúng tôi sẽ hướng dẫn cài đặt bằng pip thông qua cửa sổ dòng lệnh trên Linux hoặc macOS. Các dòng lệnh sẽ thay đổi tùy từng hệ điều hành. Trên Windows, chúng tôi khuyến khích bạn sử dụng Anaconda.

[7] Để nâng cấp pip cho tất cả người dùng trên máy thay vì chỉ mình bạn, hãy bỏ tùy chọn `--user` và đảm bảo rằng bạn có quyền admin hệ thống (ví dụ, thêm `sudo` trước toàn bộ câu lệnh trên Linux hoặc macOS).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
$ python3 -m pip install --user -U pip
Collecting pip
[...]
Successfully installed pip-19.3.1
```

Tạo môi trường ảo (virtualenv)

Nếu bạn muốn làm việc trong một môi trường độc lập (cho phép cùng lúc làm việc trên nhiều dự án mà không tạo ra xung đột phiên bản của thư viện), hãy cài đặt virtualenv[8] bằng cách chạy dòng lệnh pip sau (nhắc lại, nếu bạn muốn cài đặt virtualenv cho toàn bộ người dùng, hãy bỏ tùy chọn `--user` và chạy dòng lệnh với quyền admin):

```
$ python3 -m pip install --user -U virtualenv
Collecting virtualenv
[...]
Successfully installed virtualenv-16.7.6
```

Giờ bạn có thể tạo một môi trường Python độc lập bằng cách nhập:

```
$ cd $ML_PATH
$ python3 -m virtualenv my_env
Using base prefix '[...]'
New python executable in [...]/ml/my_env/bin/python3
Also creating executable in [...]/ml/my_env/bin/python
Installing setuptools, pip, wheel...done.
```

Để kích hoạt môi trường này, mở cửa sổ dòng lệnh và nhập vào lệnh sau:

```
$ cd $ML_PATH
$ source my_env/bin/activate # on Linux or macOS
$ .\my_env\Scripts\activate  # on Windows
```

Để tắt môi trường này đi, hãy nhập **`deactivate`**. Khi môi trường đang được kích hoạt, bất kỳ gói thư viện nào được cài đặt thông qua pip sẽ được cài đặt trên môi trường độc lập này, và Python cũng chỉ có thể truy cập vào các gói thư viện trong môi trường này (nếu bạn cũng muốn truy cập vào các gói thư viện hệ thống, khi tạo môi trường hãy sử dụng tùy chọn `--system-site-packages` của virtualenv). Để biết thêm chi tiết, hãy tham khảo tài liệu của virtualenv.

Giờ ta đã có thể cài đặt tất cả các mô-đun cần thiết thông qua lệnh pip đơn giản sau (cần thêm tùy chọn `--user` hoặc quyền admin nếu không sử dụng virtualenv):

```
$ python3 -m pip install -U jupyter matplotlib numpy pandas scipy scikit-learn
Collecting jupyter
  Downloading https://[...]/jupyter-1.0.0-py2.py3-none-any.whl
```

[8] Các công cụ khác bao gồm venv (rất giống với virtualenv và nằm trong thư viện chuẩn của Python), virtualenvwrapper (cung cấp thêm một vài tính năng khác cho virtualenv), pyenv (dễ dàng chuyển đổi giữa các phiên bản Python), và pipenv (công cụ quản lý gói thư viện tuyệt vời được tạo bởi tác giả của thư viện `requests` nổi tiếng, được xây dựng trên pip và virtualenv).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
Collecting matplotlib
  [...]
```

Nếu đã tạo một virtualenv, ta sẽ cần khai báo cho Jupyter và đặt tên cho nó:

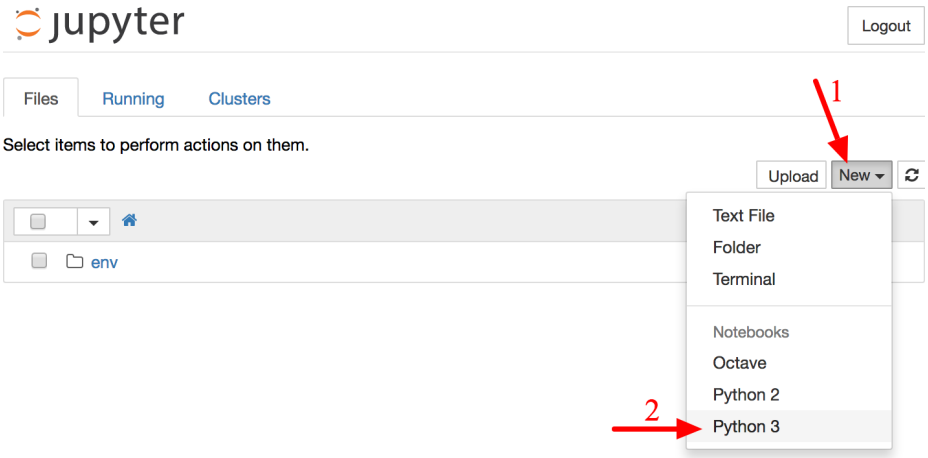
```
$ python3 -m ipykernel install --user --name=python3
```

Giờ ta có thể khởi động Jupyter bằng cách nhập câu lệnh sau:

```
$ Jupyter Notebook
[...] Serving notebooks from local directory: [...]/ml
[...] The Jupyter Notebook is running at:
[...] http://localhost:8888/?token=60995e108e44ac8d8865a[...]
[...]  or http://127.0.0.1:8889/?token=60995e108e44ac8d8865a[...]
[...] Use Control-C to stop this server and shut down all kernels [...]
```

Một máy chủ Jupyter sẽ bắt đầu chạy ở cửa sổ dòng lệnh, chờ đợi tại cổng 8888. Ta có thể truy cập vào máy chủ này bằng cách mở trình duyệt web và truy cập *http://localhost:8888/* (việc này thường được thực hiện tự động khi máy chủ được kích hoạt). Bạn sẽ thấy một không gian làm việc trống (chỉ chứa một thư mục *env* nếu bạn đã làm đúng theo hướng dẫn về virtualenv trước đó).

Giờ hãy tạo một notebook Python mới bằng cách nhấp vào nút New và chọn phiên bản Python phù hợp[9] (xem Hình 2.3). Kết quả là một tập tin notebook mới có tên *Untitled.ipynb* sẽ được tạo trong không gian làm việc, một kernel Python Jupyter sẽ khởi động để chạy notebook, và notebook sẽ được mở ở một tab mới. Ta sẽ bắt đầu bằng việc đặt lại tên cho notebook thành "Housing" (tập tin cũng sẽ được tự động đổi tên lại thành *Housing.ipynb*) bằng cách nhấp vào dòng chữ Untitled và gõ vào tên mới.



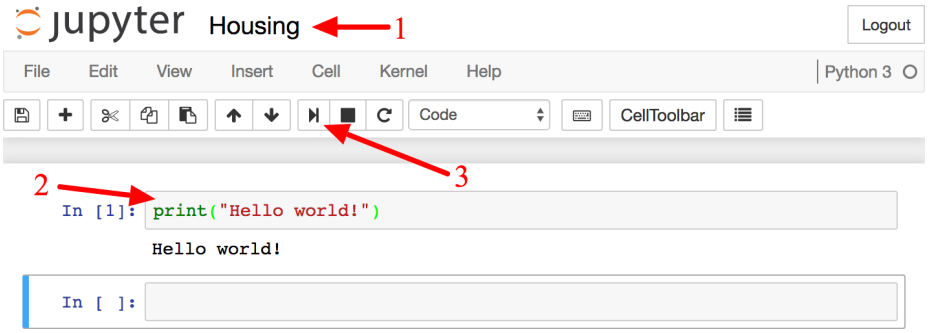
Hình 2.3. Không gian làm việc trong Jupyter

Một notebook chứa một danh sách các cell. Mỗi cell có thể chứa các mã thực thi hoặc văn bản có định dạng. Hiện tại, notebook chỉ chứa một cell mã nguồn trống được đánh nhãn "In [1]:". Hãy thử nhập **`print("Hello world!")`** vào cell và nhấn nút chạy (tham khảo Hình 2.4) hoặc

[9] Lưu ý rằng Jupyter có thể hoạt động với nhiều phiên bản Python khác nhau, và thậm chí nhiều ngôn ngữ lập trình như R hay Octave.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

nhấn tổ hợp Shift-Enter. Việc này sẽ gửi cell hiện tại đến kernel Python của notebook để chạy và nhận lại kết quả đầu ra. Kết quả sẽ được hiển thị bên dưới cell, và vì ta đã chạm đến phần kết thúc của notebook, một cell mới sẽ được tự động thêm vào. Để tìm hiểu căn bản về Jupyter Notebook, hãy xem thêm User Interface Tour tại danh mục Help của Jupyter.



Hình 2.4. In Hello World trong notebook Python

Tải Dữ liệu

Trong môi trường thông thường, dữ liệu sẽ được lưu trữ trong một cơ sở dữ liệu quan hệ (hoặc các kho chứa dữ liệu thông dụng khác) và có thể bao gồm nhiều bảng/tập tin/tài liệu. Để truy cập cơ sở dữ liệu này, ta cần có uỷ nhiệm (*credential*) và quyền truy cập[10], rồi làm quen với định dạng dữ liệu. Tuy nhiên, trong dự án này, mọi thứ đơn giản hơn rất nhiều: ta chỉ cần tải xuống một tập tin nén, *housing.tgz*, toàn bộ dữ liệu nằm trong tập tin *housing.csv*, với các giá trị được ngăn cách bởi dấu phẩy (*comma-separated values* hay CSV).

Bạn có thể sử dụng trình duyệt web để tải xuống tập tin đó, chạy `tar xzf housing.tgz` trên cửa sổ dòng lệnh để giải nén ra tập tin CSV. Tuy nhiên, lựa chọn phổ biến hơn là sử dụng một hàm nhỏ để làm việc này. Việc có một hàm tải dữ liệu về rất có lợi, đặc biệt là khi dữ liệu thay đổi thường xuyên. Ta có thể viết một đoạn mã ngắn dùng hàm này để lấy dữ liệu mới nhất (hoặc có thể tạo một tác vụ tự động lấy dữ liệu sau một khoảng thời gian nhất định). Việc tự động hóa quá trình lấy dữ liệu cũng cần thiết nếu ta cần cài đặt cùng một tập dữ liệu trên nhiều máy.

Dưới đây là hàm để lấy dữ liệu:[11]

```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/mlbvn/handson-ml2-vn/main/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    os.makedirs(housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
```

[10] Bạn có thể sẽ cần kiểm tra các ràng buộc pháp lý, ví dụ như các trường riêng tư mà không nên được lưu trữ tại các kho chứa không bảo mật

[11] Trong một dự án thực tế, ta nên lưu đoạn mã này vào một tập tin Python, nhưng giờ thì ta chỉ cần viết nó vào Jupyter Notebook

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

Việc gọi hàm `fetch_housing_data()` sẽ tạo thư mục *datasets/housing* trong không gian làm việc hiện tại, tải về tập tin *housing.tgz*, và giải nén tập tin *housing.csv* đến thư mục đó.

Giờ hãy sử dụng pandas để nạp dữ liệu. Tương tự, ta cũng nên viết một hàm nhỏ để làm điều này:

```
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)
```

Hàm này trả về một đối tượng DataFrame của pandas chứa toàn bộ dữ liệu.

Nhìn qua Cấu trúc Dữ liệu

Hãy cùng nhìn qua năm hàng đầu tiên bằng phương thức `head()` của DataFrame (xem Hình 2.5).

```
In [5]: housing = load_housing_data()
        housing.head()
```

Out[5]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populatio
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Hình 2.5. Năm hàng đầu tiên trong tập dữ liệu

Mỗi hàng biểu diễn một quận, và có 10 thuộc tính (chỉ 6 thuộc tính được minh họa trong hình): `longitude`, `latitude`, `housing_median_age`, `total_rooms`, `total_bedrooms`, `population`, `households`, `median_income`, `median_house_value`, và `ocean_proximity`.

Phương thức `info()` rất hữu dụng để xem các mô tả đơn giản về dữ liệu, cụ thể là tổng số hàng, kiểu dữ liệu của mỗi thuộc tính, và số lượng giá trị khác rỗng (xem Hình 2.6).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
In [6]: housing.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Hình 2.6. Thông tin tập dữ liệu nhà ở

Tập dữ liệu chứa tổng cộng 20,640 mẫu này khá nhỏ theo tiêu chuẩn Học Máy, nhưng lại hoàn hảo cho dự án đầu tiên. Lưu ý rằng thuộc tính `total_bedrooms` chỉ có 20,433 giá trị khác rỗng, đồng nghĩa với việc còn 207 quận thiếu đặc trưng đó và ta sẽ cần giải quyết vấn đề này.

Tất cả các thuộc tính là kiểu giá trị số, ngoại trừ thuộc tính `ocean_proximity`. Kiểu dữ liệu của thuộc tính này là `object`, tức nó có thể chứa bất kỳ đối tượng Python nào. Nhưng ta biết chắc rằng nó sẽ là một thuộc tính văn bản, vì dữ liệu này được lấy từ một tệp CSV. Khi nhìn vào năm hàng đầu tiên, ta thấy các giá trị trong cột `ocean_proximity` được lặp đi lặp lại, đồng nghĩa với việc nó có thể là một thuộc tính hạng mục. Ta có thể biết tất cả các hạng mục cũng như số quận thuộc mỗi hạng mục bằng cách sử dụng phương thức `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
```

Hãy xét thử các thuộc tính khác. Phương thức `describe()` cho biết thông tin tổng quan của các thuộc tính số (Hình 2.7).

```
In [8]: housing.describe()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedr
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553
std	2.003532	2.135952	12.585558	2181.615252	421.385070
min	-124.350000	32.540000	1.000000	2.000000	1.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.00000

Hình 2.7. Thông tin tổng quan của mỗi thuộc tính số

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ý nghĩa của các hàng `count`, `mean`, `min`, và `max` khá là rõ ràng. Lưu ý rằng các giá trị rỗng được bỏ qua (ví dụ, ta thấy giá trị `count` của `total_bedrooms` là 20,433 thay vì 20,640). Dòng `std` cho biết độ lệch chuẩn, một phép đo độ phân tán của giá trị.[12] Các hàng 25%, 50%, và 75% cho biết các *bách phân vị* (*percentile*): tương ứng với phần trăm số liệu có giá trị thấp hoặc cao hơn một mức ngưỡng cho trước. Ví dụ, `housing_median_age` của 25% các quận thấp hơn 18, trong khi đó 50% và 75% các quận có `housing_median_age` lần lượt thấp hơn 29 và 37. Chúng thường được gọi là bách phân vị thứ 25 (hay *tứ phân vị – quartile* thứ nhất), trung vị, và bách phân vị thứ 75 (hay tứ phân vị thứ ba).

Ngoài ra, ta có thể vẽ biểu đồ tần suất cho mỗi thuộc tính số để nhanh chóng biết được dạng dữ liệu đang xử lý. Biểu đồ tần suất cho biết số mẫu (trên trục tung) nằm trong một khoảng giá trị nhất định (trên trục hoành). Ta có thể vẽ biểu đồ tần suất theo từng thuộc tính một, hoặc sử dụng phương thức `hist()` trên toàn bộ tập dữ liệu (như trong đoạn mã bên dưới) để vẽ biểu đồ tần suất cho mỗi thuộc tính số (xem Hình 2.8):

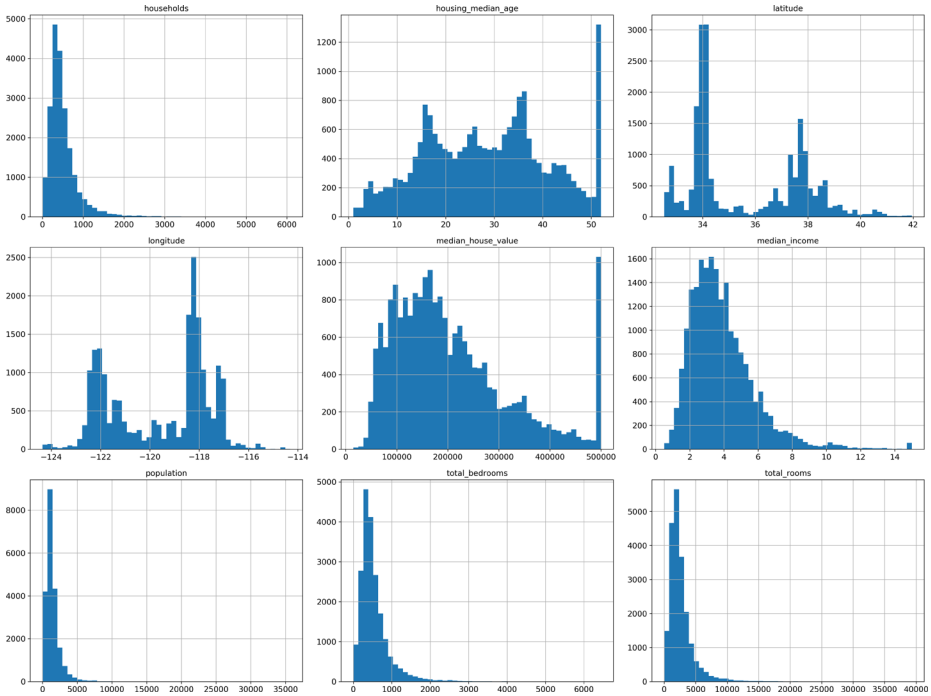
```
%matplotlib inline   # only in a Jupyter Notebook
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

Ghi chú

Phương thức `hist()` phụ thuộc vào thư viện Matplotlib. Thư viện này phụ thuộc vào backend đồ họa (*graphical backend*) mà người dùng chỉ định để vẽ kết quả. Vậy nên trước khi bắt đầu vẽ bất cứ thứ gì, ta cần chỉ định backend đồ họa cho Matplotlib. Cách đơn giản nhất là sử dụng lệnh `%matplotlib inline` trong Jupyter. Lệnh này yêu cầu Jupyter sử dụng backend của riêng nó cho Matplotlib, và biểu đồ sẽ được hiển thị ngay trong notebook. Lưu ý rằng trong Jupyter Notebook ta không cần phải gọi phương thức `show()` để hiển thị biểu đồ, do Jupyter sẽ tự động hiển thị các biểu đồ khi cell được kích hoạt.

[12] Độ lệch chuẩn thường được ký hiệu bằng σ (sigma trong bảng chữ cái Hy Lạp), và là căn bậc hai của *phương sai*, trong đó *phương sai* là trung bình bình phương sai số so với giá trị trung bình. Khi một đặc trưng có *phân phối chuẩn* theo hình chuông (hay còn được gọi là *phân phối Gauss*, một phân phối rất phổ biến), thì luật "68-95-99.7" được áp dụng: khoảng 68% giá trị nằm trong khoảng 1σ so với trung bình, 95% trong khoảng 2σ, và 99.7% nằm trong khoảng 3σ.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 2.8. Biểu đồ tần suất cho từng thuộc tính số

Ta có thể nhận thấy một vài điều từ các biểu đồ này:

1. Đầu tiên, dường như thuộc tính thu nhập trung vị không được biểu diễn bằng đơn vị đô-la Mỹ (USD). Sau khi hỏi lại nhóm thu thập dữ liệu, họ thông báo rằng dữ liệu đã được co giãn xuống với mức trần tại 15 (chính xác là 15.0001) đối với các thu nhập trung vị cao hơn, và mức sàn tại 0.5 (chính xác là 0.4999) đối với các thu nhập trung vị thấp hơn. Các con số được tính theo mười ngàn đô-la (ví dụ, 3 thực tế có nghĩa là khoảng 30,000 đô-la). Làm việc với các thuộc tính đã được tiền xử lý là việc khá phổ biến trong Học Máy. Mặc dù đây không hẳn là vấn đề, nhưng bạn nên hiểu cách mà dữ liệu được xử lý.

2. Tuổi đời trung vị và giá trị trung vị của ngôi nhà cũng được giới hạn. Việc giới hạn giá trị trung vị của ngôi nhà có thể là vấn đề nghiêm trọng vì đây là thuộc tính mục tiêu của bạn (nhãn muốn dự đoán). Thuật toán Học Máy có thể sẽ học được rằng giá cả sẽ không bao giờ vượt quá một giới hạn nhất định. Bạn cần phải kiểm tra với khách hàng (những người sẽ sử dụng hệ thống để dự đoán) để xem liệu đây có phải là một vấn đề không. Nếu khách hàng nói họ cần cả những dự đoán chính xác nằm ngoài khoảng $500,000, thì bạn có thể sử dụng hai cách sau:

   a. Thu thập nhãn thích hợp cho các quận có nhãn bị giới hạn.

   b. Loại bỏ các quận đó ra khỏi tập huấn luyện (cũng như tập kiểm tra vì hệ thống của bạn không nên bị đánh giá tệ mỗi khi nó dự đoán giá trị trên ngưỡng $500,000).

3. Các thuộc tính này có các khoảng giá trị rất khác nhau. Ta sẽ bàn luận về điều này ở phần sau của chương khi tìm hiểu về co giãn đặc trưng.

4. Cuối cùng, nhiều biểu đồ tần suất *nặng đuôi* (*tail-heavy*): biểu đồ trải dài về bên phải trung vị hơn bên trái. Điều này sẽ khiến việc phát hiện các khuôn mẫu trong một số thuật toán Học Máy gặp đôi chút khó khăn. Về sau, ta sẽ thử biến đổi phân phối của các thuộc tính này về dạng giống hình chuông hơn.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hy vọng rằng bây giờ bạn đã hiểu hơn về dạng dữ liệu mà mình đang xử lý.

Lưu ý

Chờ đã! Trước khi xem xét thêm về dữ liệu, bạn cần tạo một tập dữ liệu kiểm tra, bỏ nó qua một bên, và không bao giờ nhìn vào nó.

Tạo Tập Kiểm tra

Nghe có vẻ kỳ quặc nếu bây giờ ta tạm thời để một phần dữ liệu sang một bên. Cho tới hiện tại, ta chỉ mới nhìn qua dữ liệu, và chắc chắn rằng ta cần tìm hiểu thêm về dữ liệu trước khi quyết định sẽ sử dụng thuật toán nào. Tuy vậy, bộ não của chúng ta là một hệ thống phát hiện khuôn mẫu tuyệt vời, cũng có nghĩa là nó rất dễ quá khớp trên tập dữ liệu được quan sát: nếu nhìn vào tập kiểm tra, ta có thể gặp một vài khuôn mẫu thú vị trong tập kiểm tra, điều này khiến ta có thiên kiến lựa chọn một dạng mô hình Học Máy cụ thể. Khi đánh giá sai số khái quát trên tập kiểm tra, kết quả sẽ quá tốt. Do đó, ta sẽ đưa vào triển khai một hệ thống hoạt động tệ hơn mong đợi. Đây được gọi là thiên kiến *dòm ngó dữ liệu* (*data-snooping*).

Về lý thuyết, việc tạo một tập kiểm tra khá đơn giản: chọn ngẫu nhiên một vài mẫu, thường là 20% tập dữ liệu (hoặc ít hơn nếu tập dữ liệu có kích thước rất lớn), và để chúng sang một bên:

```
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]
```

Ta có thể sử dụng hàm này như sau:[13]

```
>>> train_set, test_set = split_train_test(housing, 0.2)
>>> len(train_set)
16512
>>> len(test_set)
4128
```

Cách này ổn, nhưng không hoàn hảo: nếu ta chạy lại chương trình, nó sẽ sinh ra một tập kiểm tra khác! Dần dần, bạn (hoặc thuật toán Học Máy) sẽ thấy được toàn bộ tập dữ liệu, và ta không muốn điều này xảy ra.

Một giải pháp là lưu lại tập kiểm tra trong lần chạy đầu tiên và nạp nó trong các lần chạy tiếp theo. Một lựa chọn khác là đặt seed cho bộ sinh số ngẫu nhiên (ví dụ, với `np.random.seed(42)`)[14] trước khi gọi hàm `np.random.permutation()` để nó luôn sinh ra cùng một cách hoán đổi.

[13] Trong cuốn sách này, khi một đoạn mã minh họa chứa cả dòng lệnh lẫn kết quả trả về, như trường hợp trên, thì nó được định dạng như khi làm việc trong trình thông dịch của Python nhằm dễ đọc hơn. Cụ thể, các đoạn mã được bắt đầu với `>>>` (hoặc `...` đối với các dòng lệnh được căn lề), còn kết quả của đoạn mã không có các ký tự bắt đầu này.

[14] Bạn sẽ thường xuyên thấy mọi người đặt giá trị này bằng 42. Con số này không có tính chất đặc biệt gì cả, ngoài việc nó là Đáp án cho Câu hỏi Tối thượng về Cuộc sống, Vũ trụ, và Vạn vật.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tuy nhiên cả hai cách trên sẽ không hoạt động trong trường hợp tập dữ liệu được cập nhật. Để có một cách chia ổn định ngay cả trong trường hợp này, một giải pháp thông dụng là sử dụng ID của mỗi mẫu để quyết định xem liệu nó có thuộc dữ liệu kiểm tra hay không (giả sử các mẫu có một ID duy nhất và bất biến). Ví dụ, bạn có thể tính toán hash của ID mỗi mẫu và đưa mẫu đó vào tập kiểm tra nếu hash thấp hơn hoặc bằng 20% giá trị hash lớn nhất. Việc này sẽ đảm bảo rằng tập kiểm tra trong các lần chạy luôn đồng nhất, kể cả khi bạn cập nhật tập dữ liệu. Tập kiểm tra mới sẽ chứa 20% mẫu mới, nhưng nó sẽ không chứa bất kỳ mẫu nào trước đó thuộc về tập huấn luyện.

Dưới đây là một cách lập trình khả thi:

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

Không may, tập dữ liệu nhà ở không có cột ID. Cách đơn giản nhất là sử dụng chỉ số hàng làm ID:

```
housing_with_id = housing.reset_index()   # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

Nếu sử dụng chỉ số hàng làm ID duy nhất, ta cần đảm bảo rằng dữ liệu mới được nối vào cuối tập dữ liệu và không hàng nào bị xóa sau này. Nếu cách này không khả thi, ta có thể thử sử dụng đặc trưng ổn định nhất để tạo ID độc nhất. Ví dụ, kinh độ và vĩ độ của một quận được đảm bảo ổn định đến vài triệu năm, nên ta có thể kết hợp chúng lại thành một ID:[15]

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

Scikit-Learn cung cấp một vài hàm để chia một tập dữ liệu thành các tập dữ liệu con theo nhiều cách. Hàm đơn giản nhất là `train_test_split()`, hoạt động gần giống với hàm `split_train_test()`, với một vài chức năng bổ sung. Đầu tiên là tham số `random_state` cho phép ta đặt seed cho bộ sinh số ngẫu nhiên. Thứ hai, ta có thể truyền vào nhiều tập dữ liệu có cùng số hàng, và hàm này sẽ chia các tập dữ liệu theo cùng một cách (điều này rất hữu ích, ví dụ như khi bạn có một DataFrame riêng cho nhãn):

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

[15] Thông tin vị trí trên thực tế khá thô, kết quả là nhiều quận có chung một ID, nên chúng sẽ luôn ở cùng trong một tập dữ liệu (huấn luyện hoặc kiểm tra). Điều này có thể dẫn tới một số thiên kiến không mong muốn khi lấy mẫu.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

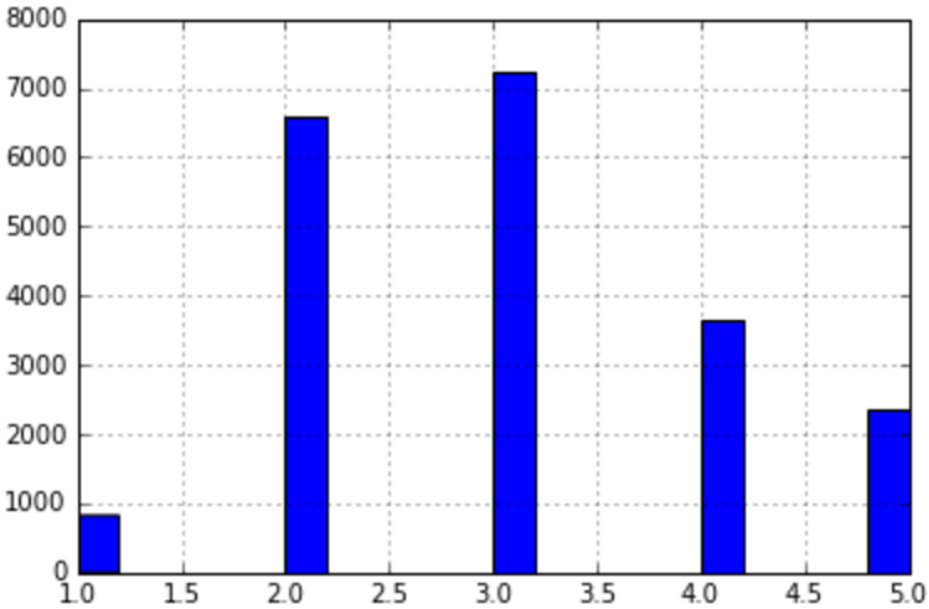
Cho tới hiện tại, ta mới chỉ đơn thuần xem xét các phương pháp lấy mẫu ngẫu nhiên. Nhìn chung, các phương pháp này hoạt động ổn nếu tập dữ liệu đủ lớn (đặc biệt là so với số lượng thuộc tính), nhưng nếu không, ta sẽ đối mặt với rủi ro đáng kể về thiên kiến lấy mẫu. Khi một công ty làm khảo sát trên 1,000 người, họ không chỉ lấy ngẫu nhiên 1,000 người trong danh bạ. Họ cố gắng đảm bảo 1,000 người này đại diện cho toàn bộ dân số. Ví dụ, dân số nước Mỹ có 51.3% nữ và 48.7% nam, nên một cuộc khảo sát tốt ở Mỹ sẽ cố gắng duy trì tỷ lệ này khi lấy mẫu: 513 nữ và 487 nam. Cách lấy mẫu này được gọi là *lấy mẫu stratified* (*stratified sampling*): toàn bộ tổng thể được chia thành các nhóm con đồng nhất gọi là *stratum*, và một số lượng mẫu xác định được lấy từ mỗi stratum để đảm bảo tập dữ liệu kiểm tra có tính đại diện cho toàn bộ tổng thể. Nếu người làm khảo sát chỉ đơn thuần lấy mẫu ngẫu nhiên, sẽ có 12% khả năng phân phối của tập dữ liệu kiểm tra bị lệch, tức hoặc ít hơn 49% nam hoặc nhiều hơn 54% nữ. Dù là trường hợp nào, kết quả khảo sát sẽ bị thiên kiến đáng kể.

Giả sử sau khi thảo luận với chuyên gia, ta biết rằng thu nhập trung vị là một thuộc tính rất quan trọng để dự đoán giá nhà trung vị. Ta cần đảm bảo rằng tập kiểm tra mang tính đại diện cho tất cả các mức thu nhập trong toàn bộ tập dữ liệu. Vì thu nhập trung vị là một thuộc tính liên tục, đầu tiên ta cần tạo thuộc tính hạng mục để biểu diễn các mức thu nhập khác nhau. Hãy quan sát biểu đồ tần suất kỹ hơn (trong Hình 2.8): phần lớn các giá trị thu nhập trung vị được phân bố trong khoảng từ 1.5 đến 6 (tức $15,000–$60,000), nhưng có vài giá trị vượt quá 6. Việc có đủ mẫu trong tập dữ liệu cho mỗi stratum là rất quan trọng. Nếu không, việc ước lượng độ quan trọng của một stratum sẽ bị thiên kiến. Nghĩa là, không nên có quá nhiều stratum, và mỗi stratum nên đủ lớn. Đoạn mã dưới đây sử dụng hàm `pd.cut()` để tạo thuộc tính mức thu nhập theo năm mức (gán nhãn từ 1 đến 5): mức 1 từ 0 đến 1.5 (tức dưới $15,000), mức 2 từ 1.5 đến 3, v.v.

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

Các mức thu nhập được biểu diễn trong Hình 2.9:

```
housing["income_cat"].hist()
```



Hình 2.9. Biểu đồ tần suất của các mức thu nhập

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bây giờ, ta đã sẵn sàng lấy mẫu stratified dựa trên các mức thu nhập. Để thực hiện việc này, ta có thể sử dụng lớp `StratifiedShuffleSplit` của 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]
```

Hãy kiểm tra xem đoạn mã có hoạt động đúng không, bắt đầu bằng việc quan sát tỷ lệ phần trăm các mức thu nhập trong tập kiểm tra.

```
>>> strat_test_set["income_cat"].value_counts() / len(strat_test_set)
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

Tương tự, ta có thể đo lường tỷ lệ phần trăm của các mức thu nhập trong toàn bộ tập dữ liệu. Hình 2.10 so sánh tỷ lệ phần trăm của các mức thu nhập trong toàn bộ dữ liệu và trong tập dữ liệu kiểm tra được tạo ra bởi lấy mẫu stratified và lấy mẫu ngẫu nhiên. Có thể thấy, lấy mẫu stratified tạo ra tập dữ liệu kiểm tra có tỷ lệ các mức thu nhập khá giống với tỷ lệ trong tập dữ liệu gốc, trong khi phân phối của tập dữ liệu lấy mẫu ngẫu nhiên bị lệch.

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

Hình 2.10. So sánh thiên kiến giữa lấy mẫu stratified và lấy mẫu ngẫu nhiên

Giờ ta có thể loại bỏ thuộc tính `income_cat` để dữ liệu trở về trạng thái ban đầu:

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

Việc ta dành khá nhiều thời gian để sinh tập dữ liệu kiểm tra là có lý do: điều này thường bị bỏ qua nhưng lại là phần cực kỳ quan trọng trong một dự án Học Máy. Hơn nữa, nhiều ý tưởng trình bày ở trên sẽ hữu ích khi thảo luận về kiểm định chéo (*cross-validation*). Giờ là lúc chuyển sang bước tiếp theo: khám phá dữ liệu.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Khám phá và trực quan hóa để hiểu Dữ liệu

Cho tới nay ta mới chỉ nhìn lướt qua dữ liệu để nắm được các đặc điểm chung của chúng. Bây giờ mục tiêu của ta là đi sâu hơn một chút.

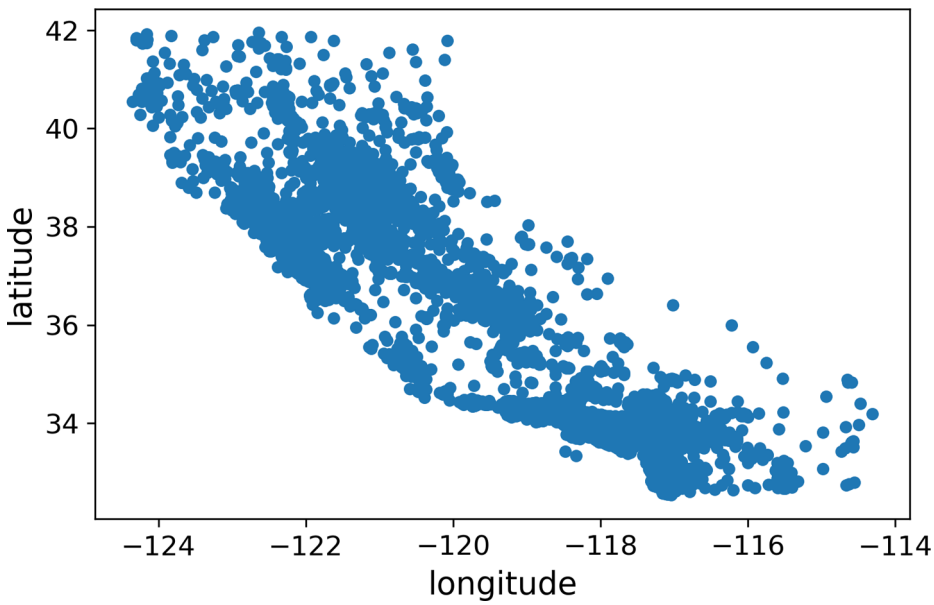
Đầu tiên, hãy đảm bảo rằng ta không động vào tập kiểm tra và chỉ sử dụng tập huấn luyện. Ngoài ra, nếu tập huấn luyện rất lớn, ta có thể chỉ lấy ra một tập nhỏ để việc thao tác dễ và nhanh hơn. Trong trường hợp này, tập huấn luyện tương đối nhỏ, nên ta có thể làm việc trực tiếp trên toàn bộ tập. Hãy tạo một bản sao để yên tâm rằng ta không làm ảnh hưởng đến tập huấn luyện gốc:

```
housing = strat_train_set.copy()
```

Trực quan hóa Dữ liệu Địa lý

Vì dữ liệu có các thông tin về địa lý (kinh độ và vĩ độ), ta nên tạo một biểu đồ phân tán (*scatterplot*) của tất cả các quận để trực quan hóa dữ liệu (Hình 2.11):

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

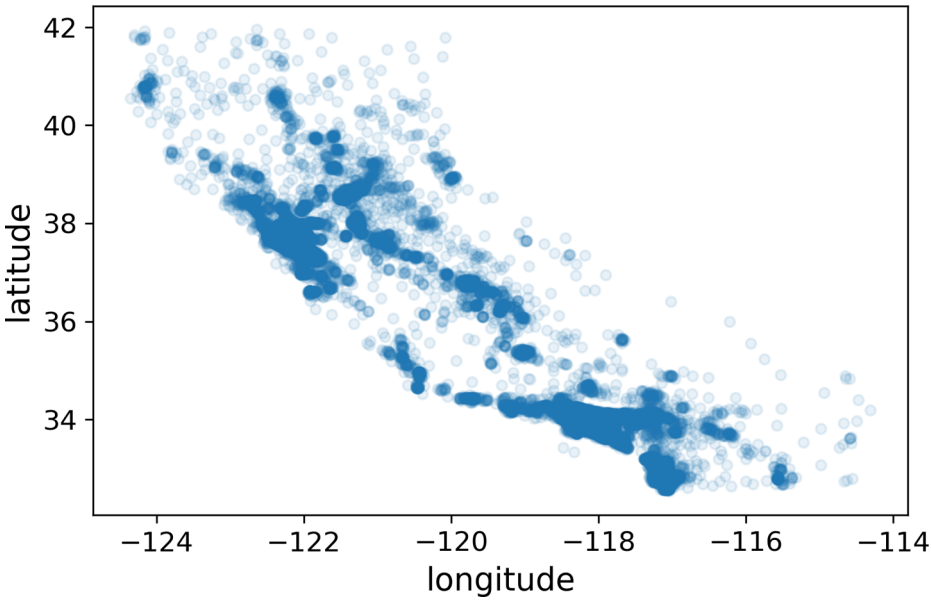


Hình 2.11. Biểu đồ phân tán theo địa lý của dữ liệu

Biểu đồ này nhìn khá giống hình dáng bang California, nhưng ngoài điều đó ra thì rất khó để thấy được khuôn mẫu cụ thể nào. Việc đặt `alpha` bằng 0.1 sẽ giúp minh họa tốt hơn các vùng có mật độ điểm dữ liệu lớn (Hình 2.12):

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

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 2.12. Một cách minh họa tốt hơn giúp các vùng có mật độ cao trở nên nổi bật

Lúc này ta đã có thể thấy rõ các khu vực đông dân, cụ thể là Bay Area, xung quanh Los Angeles và San Diego, cộng với một dải dài đông đúc ở Central Valley, đặc biệt là quanh Sacramento và Fresno.

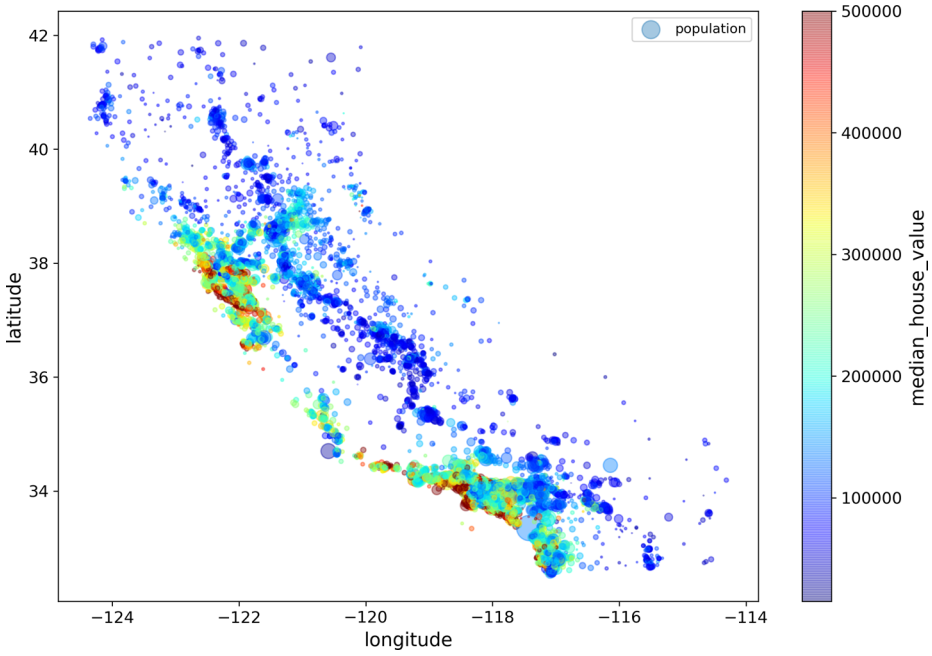
Não bộ con người phát hiện các khuôn mẫu trong hình ảnh rất tốt, nhưng ta có thể cần thử nghiệm thêm các tham số trực quan hóa để làm nổi bật các khuôn mẫu đó.

Giờ hãy nhìn vào giá nhà (Hình 2.13). Bán kính mỗi vòng tròn đại diện cho dân số của quận (đối số `s`), và màu sắc đại diện cho mức giá (đối số `c`). Ta sẽ sử dụng một bảng màu định nghĩa sẵn (đối số `cmap`) có tên là `jet`, có dải màu từ xanh (giá thấp) đến đỏ (giá cao):[16]

```
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()
```

[16] Nếu bạn đang đọc cuốn sách ở định dạng đen trắng, hãy lấy một chiếc bút đỏ rồi tô màu lên phần lớn vùng vịnh từ Bay Area xuống San Diego (như bạn có thể đã nhận định). Bạn cũng có thể tô thêm vài chấm màu vàng quanh Sacramento.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 2.13. Giá nhà ở California: đỏ là đắt, xanh là rẻ, vòng tròn càng lớn đại diện cho khu vực càng đông dân

Hình ảnh này cho thấy giá nhà liên quan mật thiết đến vị trí (như gần biển) và mật độ dân số, như bạn có thể đã biết. Một thuật toán phân cụm thường sẽ hữu ích trong việc phát hiện các cụm chính, từ đó giúp bổ sung thêm các đặc trưng mới là khoảng cách từ mẫu đến các tâm cụm. Khoảng cách đến biển cũng có thể là một thuộc tính hữu ích, tuy vậy ở Bắc California giá nhà ở các quận vùng vịnh lại không quá cao, vì thế đây không phải một quy tắc đơn giản.

Tìm sự Tương quan

Vì tập dữ liệu không quá lớn, ta có thể dễ dàng tính được *hệ số tương quan chuẩn* (*standard correlation coefficient*, còn được gọi là *hệ số tương quan Pearson r*) giữa các cặp thuộc tính bằng phương thức `corr()`:

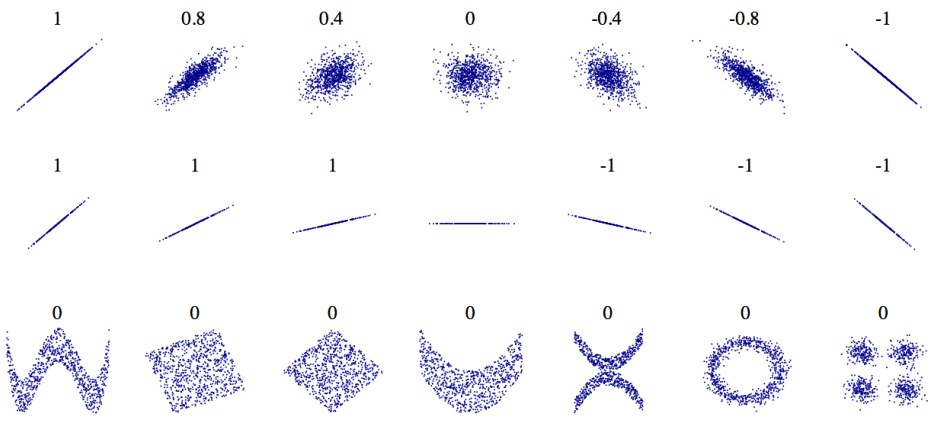
```
corr_matrix = housing.corr()
```

Giờ hãy cùng xem tính tương quan của từng thuộc tính dữ liệu với giá nhà trung vị:

```
>>> 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
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hệ số tương quan nằm trong khoảng từ –1 đến 1. Càng gần 1, mức tương quan càng dương; ví dụ, giá nhà trung vị có xu hướng tăng khi thu nhập trung vị tăng. Ngược lại, càng gần -1, mức tương quan càng âm; ta có thể thấy một sự tương quan âm nhỏ giữa vĩ độ và giá nhà trung vị (tức giá nhà hơi có xu hướng giảm đi khi đi lên phía Bắc). Cuối cùng, hệ số tương quan gần 0 nghĩa là không có sự tương quan tuyến tính nào. Hình 2.14 gồm các đồ thị biểu diễn hệ số tương quan giữa trục hoành và trục tung.



Hình 2.14. Hệ số tương quan chuẩn của nhiều tập dữ liệu (nguồn: Wikipedia; ảnh thuộc phạm vi công khai)

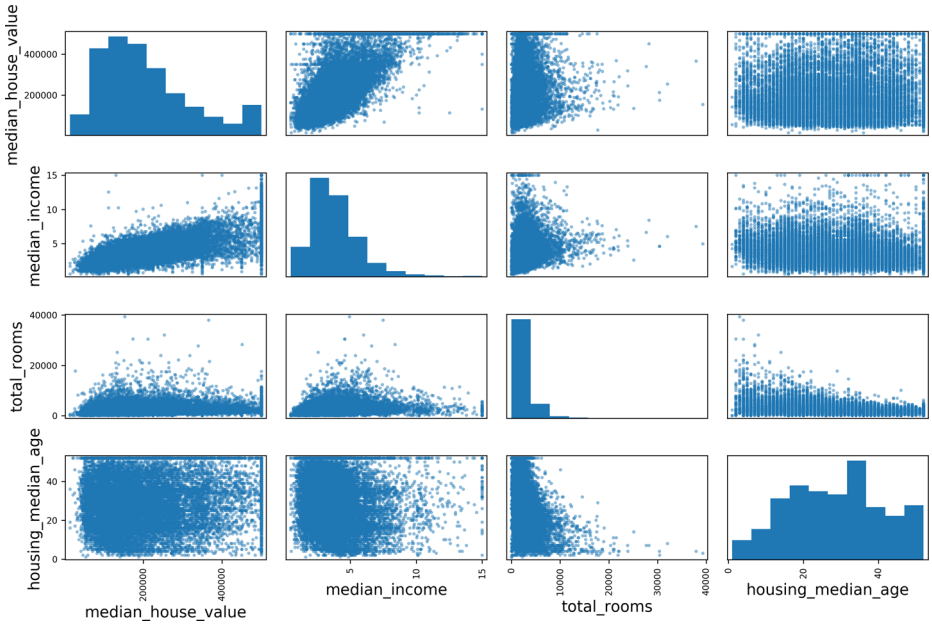
Lưu ý

Hệ số tương quan chỉ đo lường sự tương quan tuyến tính ("nếu x tăng thì y nhìn chung sẽ tăng hoặc giảm"). Nó có thể hoàn toàn bỏ qua các quan hệ phi tuyến trong dữ liệu (ví dụ, "nếu x gần 0 thì y tăng"). Hãy để ý rằng tất cả các đồ thị ở hàng dưới cùng đều có hệ số tương quan bằng 0, dù các trục toạ độ rõ ràng không độc lập với nhau: đây là các ví dụ về quan hệ phi tuyến. Thêm vào đó, trong hàng thứ hai, hệ số tương quan bằng 1 hoặc –1. Chú ý rằng điều này không liên quan gì đến độ dốc. Ví dụ, chiều cao của bạn tính bằng inch vẫn có hệ số tương quan bằng 1 với chiều cao của bạn tính bằng mét hoặc nano-mét.

Một cách khác để kiểm tra mối quan hệ tương quan giữa các thuộc tính là sử dụng hàm `scatter_matrix()` của pandas để vẽ đồ thị biểu diễn mối quan hệ giữa các thuộc tính số với nhau. Vì giờ có 11 thuộc tính số, nên ta sẽ có $11^2 = 121$ đồ thị. Vì chúng không nằm vừa trong một trang giấy nên hãy chỉ tập trung vào một vài thuộc tính hứa hẹn sẽ có tương quan cao với giá nhà trung vị (Hình 2.15):

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

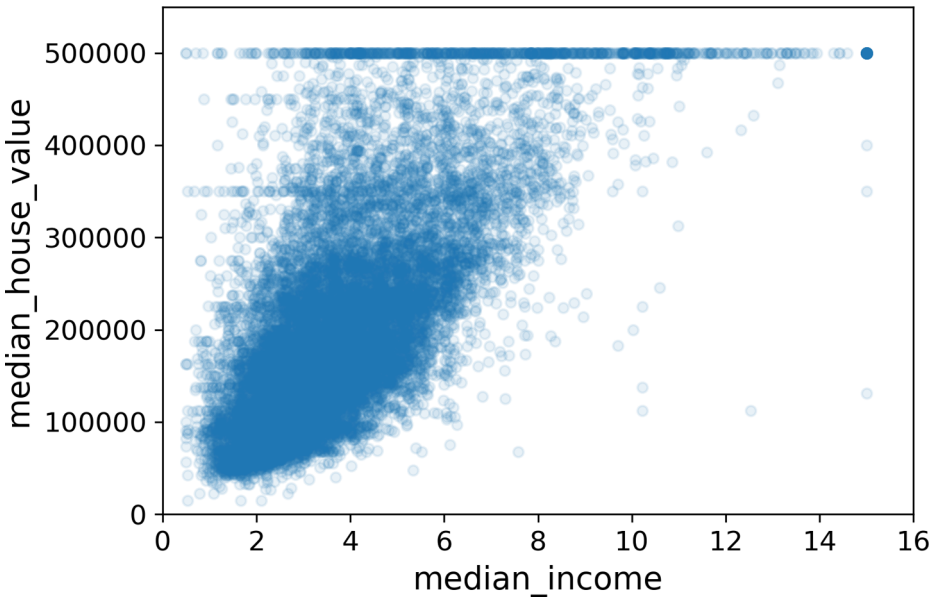


Hình 2.15. Ma trận đồ thị phân tán này biểu diễn mối quan hệ giữa các thuộc tính số với nhau, cộng với biểu đồ tần số của mỗi thuộc tính số

Đường chéo chính (từ góc trên bên trái đến góc dưới bên phải) sẽ đều là các đường thẳng nếu pandas vẽ quan hệ từng biến đối với chính biến đó, và điều này không hữu ích cho lắm. Nên thay vào đó, pandas vẽ biểu đồ tần số của từng thuộc tính (xem tài liệu hướng dẫn của pandas để biết thêm chi tiết về các lựa chọn khả dụng khác).

Thuộc tính có hứa hẹn nhất để dự đoán giá nhà trung vị là thu nhập trung vị, nên hãy phóng to đồ thị phân tán tương quan của chúng (Hình 2.16):

```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
```



Hình 2.16. Thu nhập trung vị với giá nhà trung vị

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Biểu đồ này nói lên vài điều. Đầu tiên, sự tương quan thật sự rất mạnh. Bạn có thể thấy rõ xu hướng đi lên trong khi các điểm dữ liệu thì không quá phân tán. Thứ hai, mức giá trần chúng ta thấy trước đó đã được hiển thị rõ ràng bằng một đường ngang tại $500,000. Tuy nhiên, biểu đồ này cũng tiết lộ một vài đường ngang khó nhận thấy ngay: một đường quanh $450,000, một đường khác ở khoảng $350,000, một đường khác ở khoảng $280,000, và một vài đường phía dưới nữa. Bạn có thể thử loại bỏ các quận tương ứng để ngăn thuật toán tái tạo lại những khuôn mẫu kì lạ này trong quá trình học.

Thí nghiệm Kết hợp các Thuộc tính

Hy vọng rằng các mục trước đã giúp bạn nắm được một vài cách thức khám phá và hiểu dữ liệu. Bạn đã nhận thấy vài điểm kì lạ trong dữ liệu mà bạn muốn loại bỏ trước khi đưa vào các thuật toán Học Máy, cũng như tìm thấy các mối tương quan thú vị giữa các thuộc tính, đặc biệt là với thuộc tính mục tiêu. Bạn cũng nhận ra một vài thuộc tính có phân phối nặng đuôi (*tail-heavy distribution*) và muốn biến đổi chúng (ví dụ như dùng hàm logarit). Tất nhiên, cách thức xử lý sẽ thay đổi nhiều tuỳ theo từng dự án, nhưng ý tưởng tổng quan là như nhau.

Điều cuối cùng ta có thể muốn làm trước khi chuẩn bị dữ liệu cho các thuật toán Học Máy là thử nghiệm hàng loạt các cách kết hợp thuộc tính. Ví dụ như việc biết thông tin tổng số phòng trong một quận sẽ không mấy có ích nếu ta không biết có bao nhiêu hộ gia đình trong quận đó. Thứ ta muốn là số phòng của mỗi hộ. Tương tự, bản thân thông tin tổng số phòng ngủ cũng không hữu dụng cho lắm: ta có thể muốn so sánh nó với tổng số phòng. Ngoài ra số người trong mỗi gia đình có thể cũng là một thuộc tính kết hợp thú vị. Hãy cùng tạo ra các thuộc tính mới này:

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

Và giờ hãy nhìn lại ma trận tương quan:

```
>>> corr_matrix = housing.corr()
>>> corr_matrix["median_house_value"].sort_values(ascending=False)
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
latitude                   -0.142724
bedrooms_per_room          -0.259984
Name: median_house_value, dtype: float64
```

Kết quả không tệ! Thuộc tính mới `bedrooms_per_room` có độ tương quan với giá nhà trung vị lớn hơn hẳn `total_rooms` hay `total_bedrooms`. Rõ ràng các ngôi nhà với tỉ lệ *phòng ngủ/số phòng* nhỏ hơn thường có xu hướng đắt hơn. Số phòng mỗi nhà `rooms_per_household` cũng mang nhiều thông tin hơn tổng số phòng `total_rooms` trong một quận, vì rõ ràng nhà càng lớn càng đắt.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Việc kết hợp thuộc tính khi khám phá dữ liệu lần đầu không cần phải tuyệt đối tỉ mỉ. Ta chỉ cần có một khởi đầu tốt và nhanh chóng hiểu dữ liệu, để từ đó phát triển nguyên mẫu đầu tiên tương đối tốt. Nhưng đây là một quá trình lặp lại: khi đã có một nguyên mẫu, ta có thể phân tích đầu ra của nó để có nhiều thông tin hơn và quay lại bước khám phá này.

Chuẩn bị Dữ liệu cho các Thuật toán Học Máy

Giờ là lúc chuẩn bị dữ liệu cho các thuật toán Học Máy. Thay vì thực hiện việc này thủ công, ta nên viết các hàm để làm điều đó, vì:

- Việc này cho phép ta áp dụng các phép biến đổi dễ dàng trên bất cứ tập dữ liệu nào (ví dụ khi ta có dữ liệu mới).
- Ta sẽ dần dần xây dựng một thư viện các hàm biến đổi để tái sử dụng trong tương lai.
- Ta có thể dùng các hàm này trong hệ thống thực để biến đổi dữ liệu mới trước khi đưa vào thuật toán.
- Ta có thể dễ dàng thử nhiều phép biến đổi và xem tổ hợp nào hoạt động tốt nhất.

Nhưng trước hết, hãy quay lại với tập huấn luyện ban đầu (bằng cách sao chép `strat_train_set` một lần nữa). Cùng với đó, hãy tách các thuộc tính và nhãn, vì ta không nhất thiết muốn áp dụng cùng các phép biến đổi lên cả hai (chú ý rằng phương thức `drop()` tạo một bản sao của dữ liệu và không làm ảnh hưởng tới `strat_train_set`):

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

Làm sạch Dữ liệu

Đa phần các thuật toán Học Máy không thể làm việc với các đặc trưng bị thiếu nên hãy viết một vài hàm xử lý vấn đề này. Trước đó ta đã thấy thuộc tính `total_bedrooms` có một vài giá trị bị thiếu, nên hãy xử lý nó. Ta có ba lựa chọn:

- Loại bỏ các quận tương ứng.
- Loại bỏ toàn bộ thuộc tính.
- Gán một giá trị nào đó (0, trung bình, trung vị, v.v.).

Ta có thể thực hiện những việc này dễ dàng bằng các phương thức `dropna()`, `drop()`, và `fillna()` của DataFrame:

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Nếu chọn cách thứ 3, ta nên tính giá trị trung vị trên tập huấn luyện và dùng nó để gán cho các giá trị bị thiếu trong tập huấn luyện. Đừng quên lưu giá trị trung vị này, vì ta sẽ cần đến nó khi đánh giá và triển khai hệ thống thực để thay thế các giá trị bị thiếu trong tập kiểm tra và trong dữ liệu mới.

Scikit-Learn cung cấp một lớp hữu ích để xử lý giá trị bị thiếu: `SimpleImputer`. Để sử dụng nó, đầu tiên ta cần tạo một thực thể của lớp `SimpleImputer` và chỉ định rằng ta muốn thay các giá trị thiếu bằng trung vị của thuộc tính:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
```

Vì trung vị chỉ có thể được tính cho các thuộc tính số, ta cần tạo bản sao của dữ liệu không chứa thuộc tính văn bản `ocean_proximity`:

```
housing_num = housing.drop("ocean_proximity", axis=1)
```

Giờ ta có thể khớp thực thể `imputer` với dữ liệu bằng phương thức `fit()`:

```
imputer.fit(housing_num)
```

Thực thể `imputer` chỉ đơn thuần tính trung vị của các thuộc tính và lưu kết quả vào biến thực thể `statistics_` của nó. Hiện chỉ duy nhất thuộc tính `total_bedrooms` có giá trị thiếu, nhưng ta không thể đảm bảo sẽ không có giá trị thiếu trong dữ liệu mới khi triển khai hệ thống, nên sẽ an toàn hơn nếu áp dụng `imputer` cho tất cả các thuộc tính số:

```
>>> imputer.statistics_
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
>>> housing_num.median().values
array([ -118.51 , 34.26 , 29. , 2119.5 , 433. , 1164. , 408. , 3.5409])
```

Giờ ta có thể dùng thực thể `imputer` "đã được huấn luyện" này để biến đổi tập huấn luyện bằng cách thay thế dữ liệu thiếu bằng trung vị đã tính được:

```
X = imputer.transform(housing_num)
```

Kết quả là một mảng Numpy thuần chứa các đặc trưng đã được biến đổi. Nếu ta muốn chuyển nó về dạng DataFrame của pandas, chỉ cần:

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

Thiết kế của Scikit-Learn

API của Scikit-Learn được thiết kế rất tốt. Dưới đây là các nguyên tắc thiết kế chính:[17]

Tính nhất quán
Tất cả các đối tượng có chung một giao diện nhất quán và tối giản:

Bộ ước lượng (Estimator)
Bất kỳ đối tượng nào có khả năng ước lượng các tham số dựa trên tập dữ liệu được gọi là một *bộ ước lượng* (ví dụ, một `imputer` là một bộ ước lượng). Bản thân việc ước lượng được

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

thực thi bởi phương thức `fit()`, và nó chỉ nhận một tập dữ liệu làm tham số đầu vào (hoặc hai tập với các thuật toán học có giám sát; tập thứ hai chứa nhãn). Bất kỳ tham số nào khác cần thiết cho việc điều chỉnh quá trình ước lượng được coi là một siêu tham số (ví dụ như `strategy` của `imputer`), và phải được đặt như một thuộc tính (thường thông qua một tham số của phương thức khởi tạo).

Bộ biến đổi (Transformer)
Một vài bộ ước lượng (ví dụ như `imputer`) cũng có thể được dùng để biến đổi tập dữ liệu, và chúng được gọi là *bộ biến đổi*. Cách sử dụng API cũng khá đơn giản: phép biến đổi được thực hiện bằng phương thức `transform()` với tham số đầu vào là tập dữ liệu cần biến đổi. Kết quả trả về là tập dữ liệu đã được biến đổi. Quá trình này thường phụ thuộc vào các tham số đã được học, tương tự trường hợp của `imputer`. Tất cả các bộ biến đổi đều có một phương thức khá tiện lợi là `fit_transform()`, tương đương với việc gọi lần lượt hai phương thức `fit()` và `transform()` (đôi khi phương thức `fit_transform()` được tối ưu hóa và thực thi nhanh hơn rất nhiều).

Bộ dự đoán (Predictor)
Cuối cùng, một số bộ ước lượng, với tập dữ liệu cho trước, có khả năng đưa ra các dự đoán; chúng được gọi là các *bộ dự đoán*. Ví dụ, mô hình `LinearRegression` ở chương trước là một bộ dự đoán: cho trước GDP đầu người của một quốc gia, mô hình này dự đoán mức độ hài lòng về cuộc sống của người dân quốc gia đó. Một bộ dự đoán có phương thức `predict()`: nhận một tập dữ liệu chứa các mẫu mới và trả về tập các kết quả dự đoán tương ứng. Nó cũng có phương thức `score()` để đo lường chất lượng dự đoán trên một tập kiểm tra cho trước (và các nhãn tương ứng trong trường hợp cần đánh giá các thuật toán học có giám sát).[18]

Kiểm tra (Inspection)
Tất cả các siêu tham số của bộ ước lượng đều có thể được truy cập thông qua các thuộc tính công khai của thực thể (ví dụ như `imputer.strategy`), và tất cả các tham số học được của bộ ước lượng đều có thể được truy cập thông quá các thuộc tính công khai với hậu tố là dấu gạch dưới (ví dụ như `imputer.statistics_`).

Hạn chế sử dụng lớp (class)
Các tập dữ liệu được biểu diễn dưới dạng mảng NumPy hoặc ma trận thưa SciPy thay vì các lớp tự định nghĩa. Siêu tham số chỉ đơn thuần là các xâu ký tự hoặc số trong Python.

Kết hợp (Composition)
Các thành phần được tái sử dụng một cách tối đa. Ví dụ, khá đơn giản để tạo một bộ ước lượng `Pipeline` từ một chuỗi các bộ biến đổi và kết thúc bằng một bộ ước lượng, như chúng ta sẽ thấy sau này.

Giá trị mặc định hợp lý
Scikit-Learn cung cấp các giá trị mặc định hợp lý cho hầu hết các tham số, từ đó giúp ta dễ dàng và nhanh chóng xây dựng được một hệ thống cơ bản khả dụng.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Xử lý các Thuộc tính Văn bản và Hạng mục

Cho đến nay chúng ta chỉ mới làm việc với các thuộc tính số, nên tiếp theo hãy cùng xem xét các thuộc tính văn bản. Trong tập dữ liệu này chỉ có một thuộc tính như vậy: `ocean_proximity`. Hãy nhìn qua các giá trị của 10 mẫu đầu tiên:

[17] Để biết thêm chi tiết về các nguyên tắc thiết kế, hãy tham khảo bài báo của Lars Buitinck cùng các cộng sự, "API Design for Machine Learning Software: Experiences from the Scikit-Learn Project", arXiv preprint arXiv:1309.0238 (2013).

[18] Một số bộ dự đoán cũng cung cấp các phương thức để đo lường độ tin cậy của các dự đoán.

```
>>> housing_cat = housing[["ocean_proximity"]]
>>> housing_cat.head(10)
      ocean_proximity
17606       <1H OCEAN
18632       <1H OCEAN
14650      NEAR OCEAN
3230           INLAND
3555        <1H OCEAN
19480          INLAND
8879        <1H OCEAN
13685          INLAND
4937        <1H OCEAN
4861        <1H OCEAN
```

Đây không phải là kiểu dữ liệu văn bản tùy ý: chỉ có một số lượng giới hạn các giá trị khả dụng, mỗi giá trị biểu diễn một hạng mục. Vì vậy thuộc tính này là thuộc tính hạng mục. Phần lớn các thuật toán Học Máy thích làm việc với các con số hơn, vậy nên hãy chuyển đổi các hạng mục này từ dạng văn bản sang dạng số. Để làm điều này, chúng ta có thể sử dụng lớp `OrdinalEncoder` của Scikit-Learn:[19]

```
>>> from sklearn.preprocessing import OrdinalEncoder
>>> ordinal_encoder = OrdinalEncoder()
>>> housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
>>> housing_cat_encoded[:10]
array([[0.],
       [0.],
       [4.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

Ta có thể trích xuất danh sách hạng mục bằng thuộc tính `categories_`. Đây là danh sách gồm các mảng một chiều chứa hạng mục tương ứng với mỗi thuộc tính hạng mục (trong trường hợp này danh sách chỉ chứa một mảng vì chỉ có một thuộc tính hạng mục).

```
>>> ordinal_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Một vấn đề với cách biểu diễn này là các thuật toán Học Máy sẽ giả định rằng hai giá trị gần nhau thì giống nhau hơn hai giá trị cách xa nhau. Có thể điều này sẽ không thành vấn đề trong một số trường hợp (ví dụ: với các hạng mục có thứ tự như "kém", "trung bình", "tốt", và "xuất sắc"), nhưng rõ ràng điều này không đúng với cột `ocean_proximity` (ví dụ: hạng mục 0 và 4 rõ ràng gần nhau hơn hạng mục 0 và 1). Để khắc phục vấn đề này, một giải pháp phổ biến là sử

[19] Lớp này có thể được sử dụng từ phiên bản Scikit-Learn 0.20 trở đi. Nếu bạn sử dụng phiên bản cũ hơn, vui lòng cân nhắc nâng cấp phiên bản mới hoặc sử dụng phương thức `Series.factorize()` của pandas.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

dụng thuộc tính nhị phân cho mỗi hạng mục: một thuộc tính bằng 1 khi hạng mục là "<1H OCEAN" (và trái lại bằng 0), một thuộc tính khác bằng 1 khi hạng mục là "INLAND" (và trái lại bằng 0), v.v. Phương pháp này được gọi là *biểu diễn one-hot* (*one-hot encoding*), vì chỉ có một thuộc tính bằng 1 (hot), trong khi các thuộc tính khác sẽ bằng 0 (cold). Các thuộc tính mới đôi khi được gọi là *thuộc tính giả* (*dummy attribute*). Scikit-Learn cung cấp lớp `OneHotEncoder` giúp biến đổi các giá trị hạng mục thành các vector one-hot:[20]

```
>>> from sklearn.preprocessing import OneHotEncoder
>>> cat_encoder = OneHotEncoder()
>>> housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
>>> housing_cat_1hot
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
  with 16512 stored elements in Compressed Sparse Row format>
```

Lưu ý rằng đầu ra ở đây là một *ma trận thưa* (*sparse matrix*) SciPy thay vì một mảng Numpy. Điều này rất hữu ích khi bạn có thuộc tính hạng mục với hàng nghìn danh mục. Sau khi biểu diễn thành dạng one-hot, ta có một ma trận với hàng nghìn cột chứa đầy giá trị 0 ngoại trừ giá trị 1 duy nhất ở mỗi hàng. Việc sử dụng một lượng bộ nhớ khổng lồ chỉ để lưu trữ các số 0 sẽ rất lãng phí, vì thế ta có thể sử dụng một ma trận thưa và chỉ lưu trữ vị trí của các phần tử khác 0. Ta có thể sử dụng nó gần giống như một mảng hai chiều thông thường,[21] tuy nhiên nếu bạn thực sự muốn chuyển đổi nó thành một mảng NumPy (dày đặc), chỉ cần gọi phương thức `toarray()` như sau:

```
>>> housing_cat_1hot.toarray()
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

Một lần nữa, ta có thể trích xuất danh sách các hạng mục bằng cách sử dụng thuộc tính `categories_` của thực thể encoder:

```
>>> cat_encoder.categories_
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

[20] Trước phiên bản Scikit-Learn 0.20, phương pháp này chỉ có thể mã hóa các giá trị hạng mục số nguyên, nhưng kể từ phiên bản 0.20 nó cũng có thể xử lý các loại đầu vào khác, bao gồm văn bản và hạng mục.

[21] Tham khảo tài liệu của SciPy để biết thêm chi tiết.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Mẹo

Nếu một thuộc tính hạng mục có số lượng hạng mục lớn (ví dụ: mã quốc gia, nghề nghiệp, giống loài), biểu diễn one-hot sẽ trả về rất nhiều đặc trưng đầu vào. Điều này có thể làm chậm quá trình huấn luyện và làm suy giảm chất lượng mô hình. Trong trường hợp này, ta có thể thay thế thuộc tính hạng mục bằng các đặc trưng số học có liên quan đến các hạng mục đó: ví dụ, có thể thay đặc trưng `ocean_proximity` bằng khoảng cách tới đại dương (tương tự, mã quốc gia có thể được thay bằng tổng dân số hoặc GDP đầu người). Ngoài ra, cũng có thể thay thế mỗi hạng mục bằng một vector ít chiều có được thông qua việc học, gọi là *embedding*. Biểu diễn của mỗi hạng mục sẽ được học trong quá trình huấn luyện. Đây là một ví dụ của *học biểu diễn* (*representation learning* – sẽ được đề cập trong Tập 2).

Bộ Biến đổi Tùy chỉnh

Mặc dù Scikit-Learn cung cấp khá nhiều bộ biến đổi hữu dụng, ta vẫn sẽ cần thiết kế một bộ biến đổi dành riêng cho các tác vụ như dọn dẹp hay kết hợp các thuộc tính cụ thể. Ta sẽ muốn bộ biến đổi này hoạt động trơn tru cùng với các thành phần khác của Scikit-Learn (như pipeline). Để làm được điều này, ta chỉ cần tạo một lớp và khai báo ba phương thức: `fit()` (trả về `self`), `transform()`, và `fit_transform()` bởi vì Scikit-Learn dựa trên *duck typing* (chứ không phải tính kế thừa).

Phương thức `fit_transform()` sẽ có sẵn nếu ta thêm `TransformerMixin` làm lớp cơ sở. Nếu thêm `BaseEstimator` làm lớp cơ sở (và tránh sử dụng `*args` and `**kargs` trong phương thức khởi tạo), ta sẽ có thêm hai phương thức rất hữu ích cho việc tự động tinh chỉnh các siêu tham số là `get_params()` và `set_params()`.

Ví dụ, đoạn mã sau lập trình một bộ biến đổi để ghép nối các thuộc tính kết hợp mà chúng ta đã thảo luận trước đó:

```
from sklearn.base import BaseEstimator, TransformerMixin

rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]

        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Trong ví dụ này, bộ biến đổi có một siêu tham số `add_bedrooms_per_room`, mặc định được gán là `True` (việc cung cấp các giá trị mặc định hợp lý thường khá hữu ích). Siêu tham số này sẽ cho phép ta dễ dàng biết được liệu việc thêm thuộc tính này có giúp ích cho thuật toán Học Máy hay không. Tổng quát hơn, ta có thể thêm một siêu tham số để bật/tắt một bước chuẩn bị dữ liệu bất kỳ nếu không chắc chắn 100% về nó. Càng tự động hóa những bước chuẩn bị dữ liệu này, ta có thể thử nghiệm càng nhiều tổ hợp khác nhau, từ đó tăng khả năng tìm được cách kết hợp hiệu quả nhất (và tiết kiệm rất nhiều thời gian).

Co giãn Đặc trưng

Một trong số những phép biến đổi quan trọng nhất mà ta cần thực hiện trên dữ liệu đó là *co giãn đặc trưng* (*feature scaling*). Ngoài một số ít trường hợp ngoại lệ, các thuật toán Học Máy không hoạt động tốt khi các đặc trưng đầu vào có khoảng giá trị khác nhau. Vấn đề này xảy ra trong dữ liệu nhà ở: tổng số phòng nằm trong khoảng từ 6 tới 39,320, trong khi thu nhập trung vị chỉ dao động từ 0 tới 15. Lưu ý rằng việc co giãn các giá trị mục tiêu (nhãn) thường không cần thiết.

Hai cách phổ biến nhất để đưa các thuộc tính về cùng một khoảng giá trị là: *co giãn min-max* (*min-max scaling*) và *chuẩn tắc hóa* (*standardization*).

Co giãn min-max (còn được gọi là *chuẩn hóa – normalization*) là phương pháp đơn giản nhất: các giá trị được dịch chuyển và co giãn sao cho chúng nằm trong khoảng từ 0 tới 1. Chúng ta thực hiện điều này bằng cách trừ đi giá trị nhỏ nhất và chia cho hiệu của giá trị lớn nhất và nhỏ nhất. Scikit-Learn cung cấp một bộ biến đổi có tên là `MinMaxScaler` để thực hiện phép biến đổi này. Bộ biến đổi này chứa siêu tham số `feature_range` cho phép thay đổi khoảng giới hạn trong trường hợp ta không muốn sử dụng khoảng $[0, 1]$.

Chuẩn tắc hóa thì khác: đầu tiên ta trừ đi giá trị trung bình (vì vậy các giá trị chuẩn tắc luôn có giá trị trung bình bằng 0), sau đó chia cho độ lệch chuẩn để phân phối thu được có phương sai đơn vị. Không giống như min-max scaling, chuẩn tắc hóa không ràng buộc các giá trị phải nằm trong một khoảng cụ thể. Điều này có thể trở thành vấn đề đối với một số thuật toán (ví dụ như mạng nơ-ron thường yêu cầu đầu vào nằm trong khoảng từ 0 tới 1). Tuy nhiên, chuẩn tắc hóa lại ít bị ảnh hưởng bởi các điểm ngoại lai. Ví dụ, giả sử một quận có thu nhập trung bình bằng 100 (do nhầm lẫn). Khi đó min-max scaling sẽ co tất cả các giá trị khác từ $[0, 15]$ xuống $[0, 0.15]$, còn chuẩn tắc hóa sẽ không bị ảnh hưởng đáng kể. Scikit-Learn cung cấp một bộ biến đổi gọi là `StandardScaler` cho việc chuẩn tắc hóa.

Lưu ý

Cùng với tất cả các bộ biến đổi khác, ta chỉ được khớp bộ co giãn cho dữ liệu huấn luyện, không phải toàn bộ dữ liệu (bao gồm tập kiểm tra). Sau đó ta mới sử dụng nó để biến đổi tập huấn luyện và tập kiểm tra (và cả dữ liệu mới).

Pipeline Biến đổi

Có thể thấy, có khá nhiều bước biến đổi dữ liệu cần được thực hiện theo đúng trình tự. May mắn thay, Scikit-Learn cung cấp lớp `Pipeline` nhằm thực hiện các chuỗi biến đổi như vậy. Dưới đây là một pipeline nhỏ dành cho các thuộc tính số:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Phương thức khởi tạo của `Pipeline` nhận một danh sách các cặp tên/bộ ước lượng để định nghĩa một chuỗi các bước. Tất cả các bộ ước lượng phải là bộ biến đổi (tức phải có phương thức `fit_transform()`), ngoại trừ bộ ước lượng cuối cùng. Tên gọi có thể được đặt một cách bất kỳ (miễn sao chúng độc nhất và không chứa dấu gạch dưới kép, `__`), và chúng sẽ trở nên hữu ích cho việc điều chỉnh siêu tham số sau này.

Khi ta gọi phương thức `fit()` của pipeline, nó sẽ lần lượt gọi phương thức `fit_transform()` của tất cả các bộ biến đổi, ngoại trừ bộ ước lượng cuối cùng sẽ gọi phương thức `fit()`. Đầu ra từ bộ biến đổi trước sẽ là tham số cho bộ biến đổi phía sau.

Pipeline này có các phương thức giống bộ ước lượng cuối cùng. Trong ví dụ này, bộ ước lượng cuối cùng là bộ biến đổi `StandardScaler`, vì thế phương thức `transform()` (và cả `fit_transform()` mà ta đã dùng) của pipeline này sẽ thực hiện tất cả các phép biến đổi trên tập dữ liệu theo đúng trình tự.

Cho đến nay, ta đã xử lý các cột dữ liệu hạng mục và các cột dữ liệu số một cách riêng biệt. Sẽ thuận tiện hơn nếu ta có một bộ biến đổi có thể xử lý tất cả các cột, và áp dụng những phép biến đổi phù hợp với từng cột đó. Trong phiên bản 0.20, Scikit-Learn đã giới thiệu `ColumnTransformer` cho mục đích này, và tin tốt đó là nó hoạt động rất tốt với DataFrame của pandas. Hãy sử dụng nó để thực hiện các phép biến đổi trên tập dữ liệu nhà ở:

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Đầu tiên ta khai báo lớp `ColumnTransformer`, lấy danh sách chứa tên các cột có giá trị số và danh sách chứa tên các cột có giá trị hạng mục, rồi khởi tạo một đối tượng `ColumnTransformer`. Phương thức khởi tạo đòi hỏi một danh sách các tuple, mỗi tuple chứa tên,[22] bộ biến đổi và danh sách tên (hoặc chỉ số) của các cột mà ta muốn áp dụng bộ biến đổi này. Ở ví dụ này, các cột chứa giá trị số sẽ được biến đổi bởi `num_pipeline` đã định nghĩa phía trên, và các cột chứa giá trị hạng mục sẽ được biến đổi bằng `OneHotEncoder`. Cuối cùng, ta áp dụng đối tượng `ColumnTransformer` này lên dữ liệu nhà ở: nó sẽ áp dụng mỗi phép biến đổi lên các cột phù hợp và ghép nối đầu ra theo cột (`axis=1`, các bộ biến đổi phải trả về số hàng giống nhau).

Chú ý rằng `OneHotEncoder` trả về một ma trận thưa, còn `num_pipeline` trả về một ma trận dày đặc. Khi kết quả trả về chứa cả ma trận thưa lẫn ma trận dày đặc, `ColumnTransformer` sẽ ước

[22] Cũng như pipeline, ta có thể chọn tên tùy ý, chỉ cần không chứa hai ký tự gạch dưới liên tiếp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

lượng mật độ của ma trận cuối cùng (tức tỉ lệ các phần tử khác không), rồi trả về một ma trận thưa nếu mật độ này nhỏ hơn một ngưỡng cho trước (mặc định thì `sparse_threshold=0.3`). Trong ví dụ này, nó trả về một ma trận dày đặc. Đã xong, giờ ta có một pipeline tiền xử lý có khả năng nhận toàn bộ dữ liệu giá nhà và áp dụng các phép biến đổi phù hợp lên từng cột.

Mẹo

Ngoài việc sử dụng một bộ biến đổi, ta cũng có thể sử dụng `"drop"` nếu muốn bỏ cột, hoặc `"passthrough"` nếu muốn giữ nguyên cột. Mặc định, những cột còn lại (tức những cột không xuất hiện trong các danh sách trên) sẽ bị loại bỏ, nhưng ta có thể đặt tham số `remainder` thành bất cứ bộ biến đổi nào (hoặc `"passthrough"`) nếu ta muốn xử lý chúng theo kiểu khác.

Nếu bạn đang sử dụng Scikit-Learn 0.19 hoặc cũ hơn, bạn có thể dùng một thư viện bên thứ ba như `sklearn-pandas`, hoặc tự lập trình một phép biến đổi hoạt động tương tự như `ColumnTransformer`. Một phương án khác là sử dụng lớp `FeatureUnion` để áp dụng các phép biến đổi khác nhau rồi ghép nối các kết quả đầu ra. Tuy nhiên ta sẽ không thể chỉ định các cột khác nhau cho mỗi phép biến đổi, mà chúng sẽ đều được áp dụng cho toàn bộ dữ liệu. Vấn đề này có thể được khắc phục bằng cách tự lập trình một phép biến đổi để chọn cột (ví dụ nằm trong Jupyter Notebook).

Chọn và Huấn luyện Mô hình

Ta đã định nghĩa xong bài toán, thu thập và khám phá dữ liệu, lấy mẫu tập huấn luyện và tập kiểm tra, rồi viết pipeline chứa các phép biến đổi để làm sạch và chuẩn bị dữ liệu cho các thuật toán Học Máy một cách tự động. Cuối cùng thì ta đã sẵn sàng để chọn và huấn luyện một mô hình Học Máy.

Huấn luyện và Đánh giá trên tập Huấn luyện

Tin tốt là nhờ có những bước phía trên, mọi thứ sẽ trở nên đơn giản hơn nhiều. Đầu tiên, hãy huấn luyện một mô hình Hồi quy Tuyến tính, như ta đã làm trong chương trước:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Rất tốt! Bây giờ ta đã có một mô hình Hồi quy Tuyến tính khả dụng. Hãy cùng thử nó với một vài mẫu dữ liệu từ tập huấn luyện:

```
>>> some_data = housing.iloc[:5]
>>> some_labels = housing_labels.iloc[:5]
>>> some_data_prepared = full_pipeline.transform(some_data)
>>> print("Predictions:", lin_reg.predict(some_data_prepared))
Predictions: [ 210644.6045  317768.8069  210956.4333  59218.9888  189747.5584]
>>> print("Labels:", list(some_labels))
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Nó có hoạt động, dù các dự đoán chưa được chính xác cho lắm (dự đoán đầu tiên bị lệch khoảng 40%!). Hãy đo RMSE của mô hình này trên toàn bộ tập huấn luyện bằng hàm `mean_squared_error()` có sẵn trong Scikit-Learn:

```
>>> from sklearn.metrics import mean_squared_error
>>> housing_predictions = lin_reg.predict(housing_prepared)
>>> lin_mse = mean_squared_error(housing_labels, housing_predictions)
>>> lin_rmse = np.sqrt(lin_mse)
>>> lin_rmse
68628.19819848922
```

Có vẫn hơn không, nhưng rõ ràng kết quả này không tốt: giá trị `median_housing_values` ở hầu hết các quận nằm trong khoảng $120,000 và $265,000, nên sai lệch khoảng $68,628 chưa thể làm ta hài lòng. Đây là một ví dụ cho việc mô hình dưới khớp dữ liệu huấn luyện. Khi điều này xảy ra, có thể các đặc trưng không cung cấp đủ thông tin để đưa ra dự đoán tốt, hoặc cũng có thể mô hình vẫn chưa đủ mạnh. Như ta đã thấy ở chương trước, những phương pháp chính để giải quyết vấn đề dưới khớp là chọn một mô hình mạnh hơn, cung cấp đặc trưng tốt hơn cho thuật toán hoặc giảm các điều kiện ràng buộc lên mô hình. Mô hình này chưa được điều chuẩn nên ta có thể loại trừ phương án cuối cùng. Ta có thể thử thêm các đặc trưng khác (ví dụ như log của dân số), nhưng trước tiên hãy thử sử dụng một mô hình phức tạp hơn để xem kết quả ra sao.

Hãy huấn luyện một `DecisionTreeRegressor`. Đây là một mô hình mạnh mẽ và có khả năng tìm ra các quan hệ phi tuyến trong dữ liệu (Cây Quyết định sẽ được mô tả chi tiết hơn trong Chương 6). Chắc hẳn giờ bạn đã thấy quen thuộc với đoạn mã bên dưới:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

Sau khi mô hình đã được huấn luyện xong, hãy đánh giá nó trên tập huấn luyện:

```
>>> housing_predictions = tree_reg.predict(housing_prepared)
>>> tree_mse = mean_squared_error(housing_labels, housing_predictions)
>>> tree_rmse = np.sqrt(tree_mse)
>>> tree_rmse
0.0
```

Gì thế này!? Không hề có sai lệch? Liệu có phải mô hình này hoàn hảo tuyệt đối không? Tất nhiên, khả năng cao là mô hình đã quá khớp dữ liệu một cách nặng nề. Làm thế nào để xác nhận điều này? Như đề cập ở trên, ta không muốn động vào tập kiểm tra cho đến khi sẵn sàng triển khai một mô hình đáng tin cậy. Vì vậy, ta cần phải dành riêng một phần của tập huấn luyện cho việc kiểm định mô hình và phần còn lại cho việc huấn luyện.

Kiểm định Chéo: Phương pháp Đánh giá tốt hơn

Một cách để đánh giá mô hình Cây Quyết định là sử dụng hàm `train_test_split()` để chia tập huấn luyện thành một tập huấn luyện nhỏ hơn và một tập kiểm định, rồi huấn luyện mô hình trên tập huấn luyện nhỏ và đánh giá nó trên tập kiểm định. Ta sẽ cần làm việc nhiều hơn một chút, nhưng cũng không có gì quá khó khăn và phương án này hoạt động khá tốt.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Một phương án tuyệt vời khác là sử dụng tính năng *kiểm định chéo K-fold* (*K-fold cross-validation*) của Scikit-Learn. Đoạn mã dưới đây chia ngẫu nhiên tập huấn luyện thành 10 tập con riêng biệt gọi là *fold*, rồi huấn luyện và đánh giá mô hình Cây Quyết định 10 lần, mỗi lần chọn một fold khác nhau để đánh giá và huấn luyện trên 9 fold còn lại. Kết quả là một mảng chứa 10 điểm số đánh giá:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

Lưu ý

Tính năng kiểm định chéo của Scikit-Learn làm việc với một hàm lợi ích (càng cao càng tốt) thay vì một hàm chi phí (càng thấp càng tốt), nên hàm tính điểm là hàm đối của MSE (tức có giá trị âm). Đây là lý do tại sao đoạn mã trên tính `-scores` trước khi lấy căn bậc hai.

Hãy cùng xem kết quả:

```
>>> def display_scores(scores):
...     print("Scores:", scores)
...     print("Mean:", scores.mean())
...     print("Standard deviation:", scores.std())
...
>>> display_scores(tree_rmse_scores)
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

Giờ thì Cây Quyết định không còn tốt như trước nữa. Thực chất, có vẻ nó hoạt động tệ hơn cả mô hình Hồi quy Tuyến tính! Chú ý rằng kiểm định chéo không chỉ giúp ta ước tính chất lượng của mô hình, mà nó còn đánh giá độ chính xác của ước tính này (tức độ lệch chuẩn). Cây Quyết định có điểm số khoảng 71,407 ± 2,439. Ta sẽ không có được thông tin này nếu chỉ dùng một tập kiểm định. Tuy nhiên, kiểm định chéo lại đi kèm với chi phí của việc huấn luyện mô hình nhiều lần, nên không phải lúc nào nó cũng khả thi.

Để chắc chắn hơn, hãy tính điểm số tương tự như trên cho mô hình Hồi quy Tuyến tính:

```
>>> lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
...                              scoring="neg_mean_squared_error", cv=10)
...
>>> lin_rmse_scores = np.sqrt(-lin_scores)
>>> display_scores(lin_rmse_scores)
Scores: [66782.73843989 66960.118071 70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Quả thật, mô hình Cây Quyết định quá khớp nặng tới mức nó hoạt động tệ hơn cả mô hình Hồi quy Tuyến tính.

Giờ hãy thử một mô hình cuối cùng: `RandomForestRegressor`. Như ta sẽ thấy trong Chương 7, Rừng Ngẫu nhiên hoạt động bằng cách huấn luyện nhiều Cây Quyết định với các tập con đặc trưng ngẫu nhiên, rồi lấy trung bình dự đoán của chúng. Xây dựng một mô hình dựa trên nhiều mô hình khác được gọi là *Học Ensemble*, và đây thường là một cách tốt để cải thiện hơn nữa các thuật toán ML. Mã nguồn tương tự như với các mô hình khác:

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> forest_reg = RandomForestRegressor()
>>> forest_reg.fit(housing_prepared, housing_labels)
>>> [...]
>>> forest_rmse
18603.515021376355
>>> display_scores(forest_rmse_scores)
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

Kết quả này tốt hơn rất nhiều. Rừng Ngẫu nhiên có vẻ rất hứa hẹn. Tuy nhiên, lưu ý rằng điểm số trên tập huấn luyện vẫn thấp hơn nhiều so với trên tập kiểm định, có nghĩa là mô hình vẫn quá khớp trên tập huấn luyện. Các giải pháp khả thi cho hiện tượng quá khớp là đơn giản hóa mô hình, ràng buộc mô hình (tức điều chuẩn), hoặc thu thập nhiều dữ liệu huấn luyện hơn nữa. Tuy nhiên, trước khi tìm hiểu sâu hơn về mô hình Rừng Ngẫu nhiên, bạn nên thử nhiều mô hình Học Máy khác nhau (ví dụ: thuật toán Máy Vector Hỗ trợ với các hạt nhân khác nhau, hay các mạng nơ-ron), chứ không nên dành quá nhiều thời gian để tinh chỉnh các siêu tham số. Mục tiêu là để chọn ra một vài mô hình triển vọng (từ hai đến năm mô hình).

Mẹo

Bạn nên lưu mọi mô hình đã thử nghiệm để có thể dễ dàng quay lại làm việc với bất kỳ mô hình nào. Hãy đảm bảo rằng bạn lưu cả siêu tham số và tham số đã được huấn luyện, điểm số kiểm định chéo và có thể cả các dự đoán thực tế. Điều này sẽ cho phép bạn dễ dàng so sánh điểm số giữa các loại mô hình và so sánh các loại lỗi mà chúng mắc phải. Bạn có thể dễ dàng lưu các mô hình trong Scikit-Learn bằng cách sử dụng mô-đun `pickle` của Python hoặc sử dụng `joblib`, một thư viện hiệu quả hơn trong việc chuỗi hóa các mảng Numpy lớn (bạn có thể cài đặt thư viện này bằng cách sử dụng pip):

```
import joblib

joblib.dump(my_model, "my_model.pkl")
# and later...
my_model_loaded = joblib.load("my_model.pkl")
```

Tinh chỉnh Mô hình

Giả sử ta có một danh sách những mô hình triển vọng và cần được tinh chỉnh. Hãy xem qua một vài cách để thực hiện việc này.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tìm kiếm dạng Lưới

Một lựa chọn là tinh chỉnh thủ công đến khi tìm được một bộ các giá trị siêu tham số ưng ý. Cách này thực sự rất nhàm chán và ta có thể không có đủ thời gian để khám phá nhiều cách kết hợp khác nhau.

Thay vào đó, hãy sử dụng `GridSearchCV` trong Scikit-Learn để tìm kiếm một bộ siêu tham số phù hợp. Ta chỉ cần chỉ rõ những siêu tham số muốn tinh chỉnh và các giá trị nào cần thử, rồi `GridSearchCV` sẽ sử dụng phương pháp kiểm định chéo để đánh giá tất cả các cách kết hợp khả thi của các giá trị siêu tham số. Ví dụ, đoạn mã dưới đây sẽ tìm kiếm bộ giá trị tốt nhất cho các siêu tham số của `RandomForestRegressor`:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```

Mẹo

Khi không biết giá trị nào sẽ phù hợp với một siêu tham số, một cách tiếp cận đơn giản là thử các lũy thừa liên tiếp của 10 (hoặc một số nhỏ hơn nếu bạn muốn tìm kiếm chi tiết hơn, như với siêu tham số `n_estimators` trong ví dụ này).

Đối số `param_grid` trên yêu cầu Scikit-Learn phải: đầu tiên đánh giá tất cả $3 \times 4 = 12$ cách kết hợp của `n_estimators` và `max_features` được mô tả trong `dict` đầu tiên (hiện tại bạn không cần quan tâm về ý nghĩa của các siêu tham số này; chúng sẽ được giải thích trong Chương 7), sau đó thử tất cả $2 \times 3 = 6$ cách kết hợp của hai siêu tham số đó trong `dict` thứ hai, nhưng lần này với `bootstrap` được đặt là `False` thay vì giá trị mặc định `True`.

Tìm kiếm dạng lưới sẽ khám phá $12 + 6 = 18$ cách kết hợp của các giá trị siêu tham số trong `RandomForestRegressor` và huấn luyện mỗi mô hình 5 lần (vì dùng phương pháp kiểm định chéo 5 fold). Nói cách khác, sẽ có tổng cộng $18 \times 5 = 90$ lần huấn luyện! Việc này có thể sẽ khá tốn thời gian, nhưng sau khi hoàn thành, bạn có thể tìm được bộ giá trị tốt nhất cho các siêu tham số như sau:

```
>>> grid_search.best_params_
{'max_features': 8, 'n_estimators': 30}
```

Mẹo

Vì 8 và 30 là giá trị lớn nhất được kiểm định, bạn nên thử tìm kiếm lại với giá trị lớn hơn; điểm số có thể sẽ tiếp tục được cải thiện.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bạn cũng có thể thu được bộ ước lượng tốt nhất một cách trực tiếp như sau:

```
>>> grid_search.best_estimator_
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=30, n_jobs=None, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

Ghi chú

Nếu `GridSearchCV` được khởi tạo với `refit=True` (là giá trị mặc định), thì ngay sau khi tìm được bộ ước lượng tốt nhất theo kiểm định chéo, nó sẽ huấn luyện lại mô hình trên toàn bộ tập dữ liệu huấn luyện một lần nữa. Đây thường là một ý tưởng hay vì đưa thêm nhiều dữ liệu hơn nữa sẽ có khả năng cải thiện chất lượng của mô hình.

Và điểm số đánh giá cũng sẽ được trả về:

```
>>> cvres = grid_search.cv_results_
>>> for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
...     print(np.sqrt(-mean_score), params)
...
63669.05791727153 {'max_features': 2, 'n_estimators': 3}
55627.16171305252 {'max_features': 2, 'n_estimators': 10}
53384.57867637289 {'max_features': 2, 'n_estimators': 30}
60965.99185930139 {'max_features': 4, 'n_estimators': 3}
52740.98248528835 {'max_features': 4, 'n_estimators': 10}
50377.344409590376 {'max_features': 4, 'n_estimators': 30}
58663.84733372485 {'max_features': 6, 'n_estimators': 3}
52006.15355973719 {'max_features': 6, 'n_estimators': 10}
50146.465964159885 {'max_features': 6, 'n_estimators': 30}
57869.25504027614 {'max_features': 8, 'n_estimators': 3}
51711.09443660957 {'max_features': 8, 'n_estimators': 10}
49682.25345942335 {'max_features': 8, 'n_estimators': 30}
62895.088889905004 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.14484390074 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.399594730654 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52725.01091081235 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.612956065226 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.51445842374 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Trong ví dụ này, chúng ta tìm được mô hình tốt nhất bằng cách đặt giá trị siêu tham số `max_features` là 8 và `n_estimators` là 30. Điểm RMSE cho cách kết hợp này là 49,682, tốt hơn một chút so với các giá trị mặc định của siêu tham số trước đó (là 50,182). Chúc mừng, bạn đã tinh chỉnh thành công mô hình tốt nhất của mình!

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Mẹo

Đừng quên rằng bạn có thể coi một số bước chuẩn bị dữ liệu là siêu tham số. Ví dụ, tìm kiếm dạng lưới sẽ tự động kiểm tra xem có nên thêm một đặc trưng hay không (ví dụ: sử dụng siêu tham số `add_bedroom_per_room` trong bộ chuyển đổi `CombineAttributesAdder`). Tương tự, bạn cũng có thể sử dụng tìm kiếm dạng lưới để tự động tìm cách xử lý tốt nhất các mẫu ngoại lai, các đặc trưng bị khuyết, lựa chọn đặc trưng, v.v.

Tìm kiếm Ngẫu nhiên

Tìm kiếm dạng lưới có thể được dùng để khám phá tương đối ít các cách kết hợp như trong ví dụ trước, nhưng khi không gian tìm kiếm siêu tham số lớn hơn thì ta nên ưu tiên dùng `RandomizedSearchCV`. Lớp này có thể được sử dụng tương tự như lớp `GridSearchCV`, nhưng thay vì thử tất cả các cách kết hợp khả thi, nó sẽ đánh giá một số cách kết hợp ngẫu nhiên nhất định bằng cách chọn một giá trị ngẫu nhiên cho mỗi siêu tham số tại mỗi lần lặp. Phương pháp này có hai lợi ích chính sau đây:

- Nếu ta để thuật toán tìm kiếm ngẫu nhiên chạy 1,000 lần lặp, nó sẽ khám phá 1,000 giá trị khác nhau cho mỗi siêu tham số (thay vì chỉ một vài giá trị trên mỗi siêu tham số như trong tìm kiếm dạng lưới).
- Chỉ cần nhập số lần lặp, ta sẽ kiểm soát tốt hơn tài nguyên tính toán được phân bổ cho việc tìm kiếm siêu tham số.

Phương pháp Ensemble

Một cách nữa để tinh chỉnh hệ thống là thử kết hợp các mô hình hoạt động tốt nhất. Nhóm (hoặc "ensemble") các mô hình riêng lẻ này thường sẽ hoạt động tốt hơn so với một mô hình riêng lẻ tốt nhất (giống như Rừng Ngẫu nhiên hoạt động tốt hơn Cây Quyết định riêng lẻ), đặc biệt nếu các mô hình riêng lẻ mắc các lỗi rất khác nhau. Chúng ta sẽ trình bày chi tiết hơn về chủ đề này trong Chương 7.

Phân tích các Mô hình Tốt nhất và Lỗi của Chúng

Ta thường có thêm những hiểu biết sâu hơn về bài toán bằng cách kiểm tra các mô hình tốt nhất. Ví dụ: `RandomForestRegressor` có thể tính được độ quan trọng tương đối của mỗi thuộc tính để đưa ra dự đoán chính xác:

```
>>> feature_importances = grid_search.best_estimator_.feature_importances_
>>> feature_importances
array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
       1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
       5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
       1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])
```

Hãy in ra các điểm số thể hiện độ quan trọng cùng với tên các thuộc tính tương ứng:

```
>>> extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
>>> cat_encoder = full_pipeline.named_transformers_["cat"]
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> cat_one_hot_attribs = list(cat_encoder.categories_[0])
>>> attributes = num_attribs + extra_attribs + cat_one_hot_attribs
>>> sorted(zip(feature_importances, attributes), reverse=True)
[(0.3661589806181342, 'median_income'),
 (0.1647809935615905, 'INLAND'),
 (0.10879295677551573, 'pop_per_hhold'),
 (0.07334423551601242, 'longitude'),
 (0.0629090704826203, 'latitude'),
 (0.05641917918195401, 'rooms_per_hhold'),
 (0.05335107734767581, 'bedrooms_per_room'),
 (0.041143798478729635, 'housing_median_age'),
 (0.014874280890402767, 'population'),
 (0.014672685420543237, 'total_rooms'),
 (0.014257599323407807, 'households'),
 (0.014106483453584102, 'total_bedrooms'),
 (0.010311488326303787, '<1H OCEAN'),
 (0.002856474637320158, 'NEAR OCEAN'),
 (0.00196041559947807, 'NEAR BAY'),
 (6.028038672736599e-05, 'ISLAND')]
```

Với thông tin này, bạn có thể thử loại bỏ một số đặc trưng ít hữu dụng (ví dụ: dường như chỉ có đặc trưng `ocean_proximity` là hữu dụng, nên có thể thử loại bỏ các đặc trưng khác).

Ta cũng nên xem xét các lỗi cụ thể mà hệ thống mắc phải, từ đó cố gắng hiểu nguyên nhân và tìm ra cách khắc phục vấn đề (như thêm các đặc trưng bổ sung hoặc loại bỏ các đặc trưng không có giá trị, loại bỏ các mẫu ngoại lai, v.v.).

Đánh giá Hệ thống trên Tập Kiểm tra

Sau khi tinh chỉnh xong các mô hình, ta đã có một hệ thống hoạt động đủ tốt. Bây giờ là lúc để đánh giá mô hình cuối cùng trên tập dữ liệu kiểm tra. Không có gì đặc biệt về quá trình này: chỉ cần lấy các đặc trưng và nhãn từ tập kiểm tra, chạy `full_pipeline` để biến đổi dữ liệu (gọi hàm `transform()` thay vì `fit_transform()`, bởi ta không muốn khớp trên tập kiểm tra!), và đánh giá mô hình cuối cùng trên tập kiểm tra:

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

X_test_prepared = full_pipeline.transform(X_test)

final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)   # => evaluates to 47,730.2
```

Trong một số trường hợp, ước lượng điểm cho sai số khái quát sẽ không đủ thuyết phục để triển khai mô hình: nếu mô hình này chỉ tốt hơn 0.1% so với mô hình đang được triển khai thì sao? Bạn có thể muốn biết mức độ chính xác của ước lượng này. Để làm vậy, ta có thể tính toán *khoảng tin cậy* 95% cho lỗi khái quát bằng cách sử dụng `scipy.stats.t.interval()`:

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> from scipy import stats
>>> confidence = 0.95
>>> squared_errors = (final_predictions - y_test) ** 2
>>> np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
...                          loc=squared_errors.mean(),
...                          scale=stats.sem(squared_errors)))
...
array([45685.10470776, 49691.25001878])
```

Nếu ta đã tinh chỉnh siêu tham số nhiều lần, chất lượng mô hình thường sẽ kém hơn một chút so với kết quả thu được từ kiểm định chéo (vì hệ thống lúc này đã được tinh chỉnh để hoạt động tốt trên dữ liệu kiểm định và nhiều khả năng sẽ không hoạt động tốt như vậy trên dữ liệu chưa biết). Việc này không xảy ra trong ví dụ trên, nhưng khi gặp trường hợp này, ta phải hạn chế việc tinh chỉnh siêu tham số để đạt kết quả cao hơn trên tập kiểm tra. Rất có thể mức cải thiện này sẽ không khái quát tốt trên dữ liệu mới.

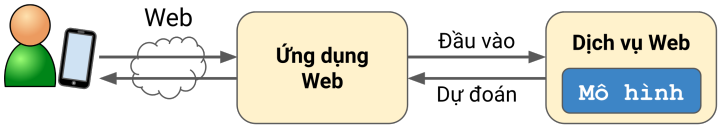
Giờ đã đến giai đoạn tiền triển khai dự án: ta cần trình bày giải pháp của mình (nhấn mạnh những gì đã học được, những gì hiệu quả và không hiệu quả, các giả định và hạn chế của hệ thống), viết tài liệu chi tiết và chuẩn bị thuyết trình với minh họa rõ ràng cùng các ngôn từ dễ nhớ (ví dụ: "thu nhập trung vị là đặc trưng số một để dự đoán giá nhà ở"). Trong ví dụ về giá nhà ở California, chất lượng cuối cùng của hệ thống không tốt hơn so với kết quả của các chuyên gia (thường lệch khoảng 20%), nhưng có lẽ ta vẫn nên triển khai hệ thống, đặc biệt nếu điều này giúp các chuyên gia tiết kiệm thời gian để họ có thể thực hiện các tác vụ thú vị và hiệu quả hơn.

Triển khai, Theo dõi, và Bảo trì Hệ thống

Thật tuyệt! Hệ thống đã được chấp thuận để triển khai! Bây giờ ta cần chỉnh trang lại mã nguồn, viết tài liệu và kiểm thử, v.v., để chuẩn bị sẵn sàng vận hành. Sau đó, ta có thể triển khai mô hình vào môi trường vận hành (*production environment*). Để làm điều này, ta có thể lưu mô hình Scikit-Learn đã được huấn luyện (ví dụ: sử dụng `joblib`), bao gồm toàn bộ pipeline tiền xử lý và dự đoán, sau đó nạp mô hình này vào môi trường vận hành và sử dụng chúng để dự đoán bằng cách gọi phương thức `predict()`. Ví dụ, có thể mô hình sẽ được sử dụng trong một trang web: người dùng sẽ nhập dữ liệu về một quận mới và nhấp vào nút Dự đoán Giá. Thao tác này sẽ gửi một truy vấn chứa dữ liệu đến máy chủ của trang web, máy chủ sẽ chuyển tiếp đến ứng dụng web và gọi phương thức `predict()` của mô hình (nên nạp mô hình ngay khi khởi động máy chủ, thay vì nạp mỗi lần muốn sử dụng mô hình). Ngoài ra, ta có thể đóng gói mô hình trong một dịch vụ web chuyên dụng mà ứng dụng của bạn có thể truy vấn thông qua REST API [23] (xem thêm Hình 2.17). Điều này giúp ta dễ dàng nâng cấp mô hình lên các phiên bản mới mà không làm gián đoạn ứng dụng chính. Nó cũng đơn giản hóa việc mở rộng quy mô, vì ta có thể khởi tạo nhiều dịch vụ web nếu cần và cân bằng tải (*load-balance*) các truy vấn đến từ ứng dụng cho các dịch vụ này. Hơn nữa, nó cho phép ứng dụng web sử dụng bất kỳ ngôn ngữ nào, không chỉ Python.

[23] Nói ngắn gọn, REST (hay RESTful) API là một API dựa trên chuẩn HTTP, tuân theo một số quy ước, chẳng hạn như sử dụng cú pháp chuẩn HTTP để đọc, cập nhật, tạo, hoặc xóa các tài nguyên (GET, POST, PUT, và DELETE) và sử dụng JSON cho đầu vào và đầu ra.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 2.17. Một mô hình được triển khai dưới dạng web service và được sử dụng bởi ứng dụng web

Một chiến lược phổ biến khác là triển khai mô hình trên điện toán đám mây, chẳng hạn như Google Cloud AI Platform (tên gọi trước là Google Cloud ML Engine): chỉ cần lưu mô hình bằng `joblib` và tải lên Google Cloud Storage (GCS), sau đó tạo phiên bản mô hình mới tại Google Cloud AI Platform và trỏ nó tới tệp GCS. Chỉ đơn giản vậy thôi! Cách này cung cấp một dịch vụ web đơn giản đảm nhận việc cân bằng tải và mở rộng quy mô. Dịch vụ web trên sẽ nhận các yêu cầu dạng JSON chứa dữ liệu đầu vào (ví dụ: của một quận) và trả về các phản hồi JSON có chứa các dự đoán. Sau đó, ta có thể sử dụng dịch vụ web này trong trang web của mình (hoặc bất kỳ môi trường vận hành nào đang được sử dụng). Việc triển khai những mô hình TensorFlow trên nền tảng AI (sẽ được đề cập trong Tập 2) không khác nhiều so với việc triển khai mô hình Scikit-Learn.

Nhưng triển khai sản phẩm không có nghĩa là đã hoàn thành công việc. Ta cũng cần viết mã giám sát để kiểm tra định kỳ chất lượng của hệ thống và kích hoạt cảnh báo khi nó giảm đi. Đây có thể là một sự sụt giảm nghiêm trọng, khả năng cao là do một thành phần bị hỏng trong cơ sở hạ tầng, nhưng hãy lưu ý rằng nó cũng có thể là do một sự suy giảm nhẹ không được chú ý một thời gian dài. Điều này khá phổ biến vì các mô hình có xu hướng "suy giảm chất lượng" theo thời gian: thật vậy, thế giới luôn thay đổi, vì vậy mô hình được huấn luyện trên dữ liệu của năm ngoái có thể sẽ không thích ứng được với dữ liệu hiện nay.

Lưu ý

Ngay cả một mô hình đơn giản như phân loại ảnh mèo và chó cũng có thể cần phải huấn luyện lại thường xuyên. Lý do không phải vì chó và mèo sẽ biến đổi trong một sớm một chiều mà vì máy ảnh liên tục thay đổi, cùng với định dạng, độ sắc nét, độ sáng và tỷ lệ kích thước ảnh. Hơn nữa, mọi người có thể yêu thích các giống chó khác vào năm tới, hoặc giả có thể đội cho vật nuôi của mình những chiếc mũ nhỏ xinh – không ai có thể biết trước được.

Vì vậy, ta luôn cần theo dõi chất lượng hiện tại của mô hình. Nhưng phải làm điều đó như thế nào? Thật ra nó phụ thuộc vào nhiều yếu tố. Trong một số trường hợp, chất lượng của mô hình có thể được suy ra từ các phép đo mục tiêu. Ví dụ, nếu mô hình của bạn là một phần của hệ thống đề xuất và nó đề xuất các sản phẩm mà người dùng có thể quan tâm thì thật dễ dàng để theo dõi số lượng sản phẩm được đề xuất đã bán mỗi ngày. Nếu con số này giảm xuống (so với các sản phẩm không được đề xuất), thì khả năng cao là do mô hình. Điều này có thể là do pipeline dữ liệu bị hỏng, hoặc có lẽ mô hình cần được huấn luyện lại trên dữ liệu mới (như chúng ta sẽ thảo luận ngay sau đây).

Tuy nhiên, không phải lúc nào bạn cũng có thể xác định được chất lượng của mô hình mà không cần bất kỳ phân tích nào của con người. Ví dụ, giả sử bạn đã huấn luyện một mô hình phân loại hình ảnh (tham khảo Chương 3) để phát hiện lỗi sản phẩm trên dây chuyền sản xuất. Làm thế nào để nhận được cảnh báo nếu chất lượng hoạt động của mô hình giảm xuống, trước khi hàng nghìn sản phẩm bị lỗi được chuyển đến khách hàng? Một giải pháp là gửi cho người đánh giá tất cả các bức ảnh mà mô hình đã phân loại (đặc biệt là những bức ảnh mà mô hình không chắc chắn lắm). Tùy từng tác vụ mà người đánh giá cần phải là chuyên gia, hoặc họ cũng có thể

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

là người thường, chẳng hạn như công nhân trên nền tảng cung ứng cộng đồng (ví dụ: Amazon Mechanical Turk). Trong một số ứng dụng, người đánh giá thậm chí có thể là chính người dùng, ví dụ như thông qua khảo sát hoặc captcha.[24]

Dù bằng cách nào, ta cũng cần đặt một hệ thống giám sát (có hoặc không có người đánh giá để đánh giá trực tiếp mô hình), cũng như tất cả các quy trình có liên quan để xác định những gì cần làm trong trường hợp thất bại và làm sao để chuẩn bị ứng phó. Thật không may, thực tế thì việc này thường tốn nhiều công sức hơn việc xây dựng và huấn luyện một mô hình.

Nếu dữ liệu tiếp tục thay đổi, ta sẽ cần cập nhật tập dữ liệu và huấn luyện lại mô hình thường xuyên, do đó ta nên tự động hóa toàn bộ quá trình càng nhiều càng tốt. Dưới đây là một số việc có thể được tự động hóa:

- Thường xuyên thu thập dữ liệu mới và gán nhãn (ví dụ: sử dụng người đánh giá).
- Viết mã để huấn luyện mô hình và tinh chỉnh siêu tham số một cách tự động. Mã này có thể chạy tự động, ví dụ như sau mỗi ngày hoặc mỗi tuần, tuỳ thuộc vào nhu cầu của bạn.
- Viết mã để đánh giá cả mô hình mới và mô hình cũ trên tập kiểm tra được cập nhật, và triển khai mô hình nếu chất lượng của nó không bị giảm sút (nếu giảm thì hãy tìm nguyên nhân).

Ta cũng nên đảm bảo rằng ta đã đánh giá chất lượng dữ liệu đầu vào của mô hình. Thỉnh thoảng chất lượng mô hình sẽ giảm nhẹ do chất lượng tín hiệu kém (ví dụ, một cảm biến bị trục trặc gửi đi các giá trị ngẫu nhiên, hoặc đầu ra công việc của một nhóm khác chưa được cập nhật), nhưng có thể mất một lúc trước khi chất lượng giảm sút đủ để kích hoạt cảnh báo. Nếu kiểm soát đầu vào của mô hình, ta có thể nắm bắt điều này sớm hơn. Ví dụ, ta có thể kích hoạt cảnh báo nếu có nhiều đầu vào bị thiếu đặc trưng, hoặc nếu giá trị trung bình và độ lệch chuẩn khác quá nhiều so với tập huấn luyện, hoặc một đặc trưng hạng mục bắt đầu chứa những hạng mục mới.

Cuối cùng, hãy giữ bản sao lưu cho mỗi mô hình và có quy trình cũng như công cụ để quay trở lại mô hình trước đó một cách nhanh chóng nếu mô hình mới bắt đầu hoạt động kém hẳn đi vì lý do nào đó. Bản sao lưu cũng giúp ta dễ dàng so sánh mô hình mới với các mô hình trước đó. Tương tự, ta nên giữ bản sao lưu cho mỗi phiên bản dữ liệu để có thể quay trở lại tập dữ liệu trước đó nếu dữ liệu mới có vấn đề (ví dụ, nếu dữ liệu mới chứa nhiều điểm ngoại lai). Bản sao lưu dữ liệu cũng cho phép ta đánh giá bất kỳ mô hình nào trên bất kỳ tập dữ liệu nào trước đó.

Mẹo

Bạn có thể muốn tạo một số tập con từ tập kiểm tra để đánh giá chất lượng của mô hình trên một phần cụ thể của dữ liệu. Ví dụ, bạn muốn có một tập con chỉ chứa dữ liệu gần đây nhất, hoặc một tập kiểm tra cho những loại đầu vào cụ thể (ví dụ, quận nằm trong đất liền với quận ven biển). Việc này giúp bạn hiểu sâu hơn về điểm mạnh và điểm yếu của mô hình.

Có thể thấy, Học Máy cần khá nhiều cơ sở hạ tầng, vì vậy không có gì bất ngờ nếu dự án ML đầu tiên của bạn hao tổn nhiều công sức và thời gian để xây dựng và triển khai. May mắn thay, một khi tất cả các cơ sở hạ tầng đã ổn định, chặng đường từ ý tưởng đến sản phẩm thực tế sẽ ngắn hơn rất nhiều.

[24] Captcha là một bài kiểm tra để đảm bảo người dùng không phải là robot. Những bài kiểm tra này thường được sử dụng như một cách ít tốn kém để gán nhãn dữ liệu huấn luyện.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hãy Thực Hành!

Hy vọng rằng chương này đã giúp bạn hiểu về một dự án Học Máy cũng như cung cấp những công cụ bạn có thể sử dụng để huấn luyện một hệ thống hiệu quả. Có thể thấy, đa số công việc tập trung vào bước chuẩn bị dữ liệu: xây dựng công cụ kiểm soát, thiết lập quy trình đánh giá bằng con người, và tự động hóa việc huấn luyện mô hình một cách thường xuyên. Các thuật toán Học Máy dĩ nhiên cũng đóng vai trò quan trọng, nhưng bạn nên hiểu rõ về quy trình tổng thể và biết thấu đáo ba hoặc bốn thuật toán thay vì dành trọn thời gian để khám phá những thuật toán nâng cao.

Nếu bạn chưa làm điều đó, vậy thì bây giờ đã đến lúc để bạn mở laptop, chọn một tập dữ liệu mà bạn quan tâm, cố gắng thực hiện quy trình từ đầu đến cuối. Một nơi tốt để bắt đầu là một website dành cho các cuộc thi như *http://kaggle.com/*: bạn sẽ có một tập dữ liệu, một mục tiêu rõ ràng, và nhiều người để chia sẻ kinh nghiệm. Chúc bạn thực hành vui vẻ!

Bài tập

Những bài tập sau đây đều dựa trên dữ liệu nhà ở của chương này:

1. Hãy thử sử dụng một bộ hồi quy SVM (`sklearn.svm.SVR`) với một vài siêu tham số như `kernel="linear"` (với những giá trị khác nhau cho `C`) hoặc `kernel="rbf"` (với các giá trị khác nhau cho `C` và `gamma`). Đừng vội quan tâm đến những siêu tham số này là gì. Chất lượng của mô hình `SVR` tốt nhất như thế nào?
2. Hãy thử thay `GridSearchCV` bằng `RandomizedSearchCV`.
3. Hãy thêm một bộ biến đổi vào pipeline để chỉ chọn ra những thuộc tính quan trọng nhất.
4. Thử tạo một pipeline đơn giản có thể thực hiện từ việc chuẩn bị dữ liệu đến việc đưa ra dự đoán cuối cùng.
5. Tự động khám phá một số tuỳ chọn trong việc chuẩn bị dữ liệu bằng cách sử dụng `GridSearchCV`.

Lời giải cho phần bài tập nằm ở các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 3

Bài toán Phân loại

Chương 1 đã nhắc đến việc đa số các tác vụ phổ biến của học có giám sát là hồi quy (dự đoán giá trị) và phân loại (dự đoán các hạng mục). Trong Chương 2 chúng ta đã cùng nhau khám phá một tác vụ hồi quy là dự đoán giá trị nhà ở, sử dụng một số thuật toán như Hồi quy Tuyến tính, Cây Quyết định và Rừng Ngẫu nhiên (sẽ được giải thích cụ thể trong những chương sau). Bây giờ chúng ta sẽ tìm hiểu về các hệ thống phân loại.

MNIST

Chương này sẽ sử dụng tập dữ liệu MNIST. Đây là một tập gồm 70,000 ảnh kích thước nhỏ chứa chữ số được viết tay bởi học sinh trung học phổ thông và nhân viên của Cục Điều Tra Dân Số Hoa Kỳ (US Census Bureau). Mỗi ảnh được gán nhãn với chữ số mà nó biểu diễn. Tập dữ liệu này được sử dụng nhiều tới mức nó được xem như "hello world" trong Học Máy: các thuật toán phân loại mới thường sẽ được kiểm tra thử trên tập MNIST. Với những ai đang tìm hiểu về Học Máy, thì làm việc với tập dữ liệu này chỉ là chuyện sớm hay muộn.

Scikit-Learn cung cấp nhiều hàm hỗ trợ để tải những tập dữ liệu phổ biến, bao gồm cả MNIST. Đoạn mã bên dưới tải về tập dữ liệu MNIST:[1]

```
>>> from sklearn.datasets import fetch_openml
>>> mnist = fetch_openml('mnist_784', version=1)
>>> mnist.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details',
           'categories', 'url'])
```

Những tập dữ liệu được nạp bởi Scikit-Learn thường có cấu trúc từ điển tương tự nhau, bao gồm các từ khóa:

- `DESCR` chứa mô tả về tập dữ liệu
- `data` chứa một mảng hai chiều, mỗi hàng ứng với một mẫu dữ liệu và mỗi cột ứng với một đặc trưng
- `target` chứa một mảng các nhãn

Hãy xem qua các mảng này:

[1] Theo mặc định Scikit-Learn lưu các tập dữ liệu đã tải xuống trong thư mục *$HOME/scikit_learn_data*.

```
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

Tập dữ liệu MNIST chứa 70,000 ảnh với 784 đặc trưng trên từng ảnh. Lí do là mỗi ảnh chứa 28×28 điểm ảnh, và mỗi đặc trưng đại diện cho cường độ của một điểm ảnh với giá trị nằm trong khoảng từ 0 (trắng) đến 255 (đen). Hãy xem thử một mẫu từ tập dữ liệu. Tất cả những gì ta cần làm là lấy vector đặc trưng của một mẫu, chuyển kích thước thành một mảng 28×28 và dùng hàm `imshow()` của Matplotlib để hiển thị bức ảnh:

```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = X[0]
some_digit_image = some_digit.reshape(28, 28)

plt.imshow(some_digit_image, cmap="binary")
plt.axis("off")
plt.show()
```

Đây giống như bức ảnh của một số 5 và thật vậy, nhãn của nó là 5:

```
>>> y[0]
'5'
```

Chú ý rằng các nhãn đang được lưu trữ dưới dạng chuỗi ký tự. Hầu hết các thuật toán Học Máy làm việc với kiểu dữ liệu số, vì vậy hãy chuyển `y` thành dạng số nguyên:

```
>>> y = y.astype(np.uint8)
```

Để cảm nhận rõ ràng hơn độ phức tạp của tác vụ phân loại, hãy quan sát thêm một số ảnh từ tập dữ liệu MNIST trong Hình 3.1.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hình 3.1. Các chữ số từ tập dữ liệu MNIST

Chờ đã! Ta phải tạo tập kiểm tra và để sang một bên trước khi phân tích sâu hơn. Thực chất, tập dữ liệu MNIST đã được chia sẵn thành tập huấn luyện (60,000 ảnh đầu tiên) và tập kiểm tra (10,000 ảnh còn lại):

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Tập huấn luyện đã được xáo trộn sẵn, điều này là cần thiết vì nó đảm bảo tất cả các fold trong kiểm định chéo sẽ tương tự nhau (ta không muốn có fold nào đó bị thiếu một vài chữ số). Hơn nữa, một vài thuật toán khá nhạy cảm với thứ tự mẫu huấn luyện và sẽ hoạt động kém nếu liên tục nhận được nhiều mẫu có cùng nhãn. Việc xáo trộn tập dữ liệu giúp đảm bảo điều này không xảy ra.[2]

Huấn luyện một Bộ Phân loại Nhị phân

Hãy đơn giản hóa vấn đề thành bài toán xác định chỉ một chữ số – ví dụ như số 5. "Bộ nhận diện số 5" này là một ví dụ tiêu biểu cho *bộ phân loại nhị phân* (*binary classifier*), chỉ có thể phân biệt giữa hai lớp: 5 và không phải 5. Hãy tạo các vector mục tiêu cho tác vụ phân loại này:

[2] Xáo trộn dữ liệu có thể là một ý kiến tồi trong một số trường hợp, ví dụ như khi làm việc với dữ liệu chuỗi thời gian (giá thị trường cổ phiếu hay điều kiện thời tiết). Ta sẽ tìm hiểu về vấn đề này trong những chương tiếp theo.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
y_train_5 = (y_train == 5)  # True for all 5s, False for all other digits
y_test_5 = (y_test == 5)
```

Bây giờ hãy chọn và huấn luyện một bộ phân loại. Ta có thể bắt đầu với bộ phân loại *Hạ Gradient Ngẫu nhiên* (*Stochastic Gradient Descent* - SGD), sử dụng lớp `SGDClassifier` của Scikit-Learn. Ưu điểm của bộ phân loại này là khả năng xử lý những tập dữ liệu rất lớn một cách hiệu quả. Một phần nguyên nhân là bởi SGD xử lý từng mẫu dữ liệu huấn luyện một cách độc lập (giúp SGD rất phù hợp cho học trực tuyến). Ta sẽ thấy điều này trong các phần sau. Hãy tạo một `SGDClassifier` và huấn luyện nó trên toàn bộ tập huấn luyện.

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

Mẹo

`SGDClassifier` phụ thuộc vào sự ngẫu nhiên trong quá trình huấn luyện (đây là lý do tại sao tên của thuật toán chứa từ "stochastic – ngẫu nhiên"). Nếu muốn tái hiện lại kết quả, bạn hãy đặt giá trị cho tham số `random_state`.

Bây giờ chúng ta có thể sử dụng bộ phân loại để tìm ra các ảnh chứa số 5:

```
>>> sgd_clf.predict([some_digit])
array([ True])
```

Bộ phân loại dự đoán ảnh này chứa số 5 (`True`). Có vẻ như đây là dự đoán đúng cho trường hợp này! Bây giờ, hãy thực hiện đánh giá chất lượng của mô hình thu được.

Phép đo Chất lượng

Việc đánh giá một bộ phân loại thường sẽ khó hơn việc đánh giá một bộ hồi quy, vì thế ta sẽ dành phần lớn thời lượng của chương để thảo luận về chủ đề này. Có rất nhiều phép đo chất lượng khả dụng, do đó hãy sẵn sàng cho những khái niệm và thuật ngữ mới!

Đánh giá Độ Chính xác bằng Kiểm định chéo

Một cách hiệu quả để đánh giá một mô hình là sử dụng kiểm định chéo (*cross-validation*), như ta đã làm trong Chương 2.

Lập trình kiểm định chéo

Đôi lúc, ta sẽ cần điều khiển quá trình kiểm định chéo nhiều hơn những gì Scikit-Learn cung cấp sẵn. Trong trường hợp đó, ta có thể tự lập trình kiểm định chéo. Đoạn mã dưới đây có chức năng tương tự hàm `cross_val_score()` của Scikit-Learn, và in ra cùng một kết quả:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
skfolds = StratifiedKFold(n_splits=3, random_state=42)

for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred))  # prints 0.9502, 0.96565, and 0.96495
```

Lớp `StratifiedKFold` thực hiện lấy mẫu stratified (như đã giải thích ở Chương 2) để tạo ra các fold chứa tỉ lệ đại diện của mỗi lớp. Tại mỗi vòng lặp, đoạn mã tạo một bản sao của bộ phân loại, huấn luyện bản sao đó trên các fold huấn luyện và dự đoán trên fold kiểm tra, rồi sau đó đếm số lượng và tính tỉ lệ dự đoán đúng.

Hãy sử dụng hàm `cross_val_score()` để đánh giá mô hình `SGDClassifier`, thiết lập kiểm định chéo K-fold với 3 fold. Nhắc lại rằng kiểm định chéo K-fold chia tập huấn luyện thành K fold (trong trường hợp này là 3), sau đó dự đoán và đánh giá trên từng fold bằng mô hình được huấn luyện trên các fold còn lại (tham khảo Chương 2):

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.96355, 0.93795, 0.95615])
```

Thật bất ngờ! Ta đạt được độ chính xác 93% trên tất cả các fold kiểm định chéo sao? Điều này hẳn là rất tuyệt đúng không? Đừng vội mừng, hãy xem thử độ chính xác của một bộ phân loại cực kỳ đơn giản là gán cho tất cả các ảnh cùng một nhãn "không phải 5":

```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        return self
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

Bạn có thể đoán được độ chính xác của mô hình này không?

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.91125, 0.90855, 0.90915])
```

Mô hình này cũng có độ chính xác trên 90%. Điều này đơn thuần là do chỉ có khoảng 10% số ảnh là 5, nên nếu ta luôn dự đoán một ảnh *không phải* 5, ta sẽ đúng khoảng 90% số lần. Thắng luôn cả Nostradamus.

Điều này minh hoạ lý do vì sao việc độ chính xác nhìn chung không phải phép đo chất lượng được ưa dùng cho các bộ phân loại, đặc biệt là khi ta đang làm việc với các *tập dữ liệu lệch* (*skewed dataset*), tức là khi một vài lớp có nhiều dữ liệu hơn các lớp khác.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ma trận Nhầm lẫn

Một cách hiệu quả hơn để đánh giá chất lượng của một bộ phân loại là nhìn vào *ma trận nhầm lẫn* (*confusion matrix*). Ý tưởng chính là tính số lần một mẫu của lớp A bị phân loại nhầm thành lớp B. Ví dụ, để biết số lần bộ phân loại gán nhầm các ảnh 5 thành 3, ta nhìn vào hàng 5 cột 3 của ma trận nhầm lẫn.

Để tính ma trận này, đầu tiên ta cần có một tập các dự đoán để so sánh với nhãn mục tiêu. Ta có thể sử dụng tập kiểm tra, nhưng tạm thời hãy khoan động đến nó (nhớ rằng ta chỉ muốn sử dụng tập kiểm tra vào cuối dự án, một khi đã có một bộ phân loại sẵn sàng để triển khai). Thay vào đó, ta có thể sử dụng hàm `cross_val_predict()`:

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

Cũng giống hàm `cross_val_score()`, `cross_val_predict()` thực hiện kiểm định chéo K-fold, nhưng hàm này trả về các dự đoán trên mỗi fold kiểm tra thay vì điểm số đánh giá. Nhờ đó, ta có được một dự đoán sạch cho mỗi mẫu trong tập huấn luyện ("sạch" ý nói là mô hình đưa ra dự đoán cho mẫu dữ liệu không xuất hiện trong quá trình huấn luyện).

Giờ ta đã sẵn sàng tính ma trận nhầm lẫn bằng hàm `confusion_matrix()`. Chỉ cần đưa vào các nhãn mục tiêu (`y_train_5`) và các nhãn dự đoán (`y_train_pred`):

```
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
array([[53057,  1522],
       [ 1325,  4096]])
```

Mỗi hàng trong ma trận nhầm lẫn biểu diễn một *lớp thật*, trong khi mỗi cột là một *lớp dự đoán*. Hàng đầu tiên của ma trận xét các ảnh của lớp "không phải 5" (*lớp âm*): 53,057 ảnh được phân loại đúng cho lớp này (gọi là *âm tính thật – true negative*), trong khi 1,522 ảnh còn lại bị phân loại sai là 5 (*dương tính giả – false positive*). Hàng thứ hai xét các ảnh là 5 (*lớp dương*): 1,325 ảnh bị phân loại sai là không phải 5 (*âm tính giả – false negative*), còn lại 4,096 ảnh được phân loại đúng là 5 (*dương tính thật – true positive*). Một bộ phân loại hoàn hảo sẽ chỉ có dương tính thật và âm tính thật, nên ma trận nhầm lẫn của nó sẽ chỉ có các giá trị khác không nằm trên đường chéo chính (từ góc trên bên trái đến góc dưới bên phải):

```
>>> y_train_perfect_predictions = y_train_5  # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579,     0],
       [    0,  5421]])
```

Ma trận nhầm lẫn cho ta rất nhiều thông tin, nhưng đôi lúc ta có thể sẽ cần một phép đo súc tích hơn. Một phép đo đáng cân nhắc là độ chính xác của các dự đoán cho lớp dương, được gọi là *precision* của bộ phân loại (Phương trình 3.1).

$$\text{precision} = \frac{TP}{TP + FP}$$

Phương trình 3.1. Precision

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Trong đó TP là số mẫu dương tính thật, FP là số mẫu dương tính giả.

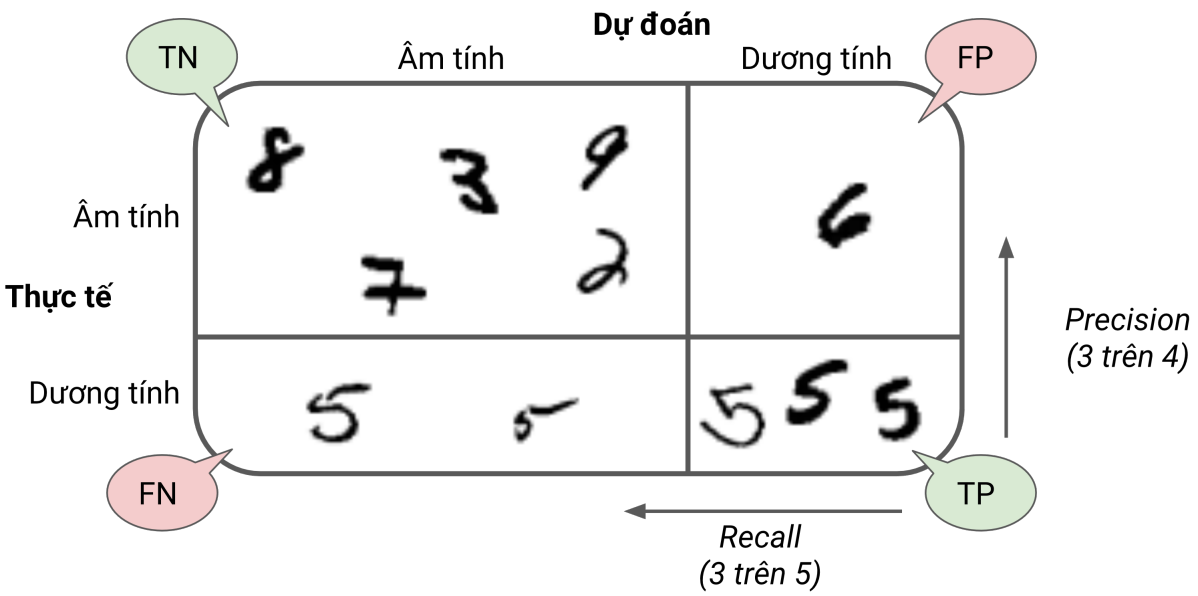
Một cách đơn giản để có precision tuyệt đối là chỉ đưa ra một dự đoán dương và đảm bảo đó là dự đoán đúng (precision $= 1/1 = 100\%$). Nhưng cách làm này sẽ không mang nhiều ý nghĩa, vì bộ phân loại sẽ chỉ dự đoán một mẫu là dương và không cân nhắc những mẫu còn lại. Vì vậy, precision thường được sử dụng cùng với một phép đo khác gọi là *recall*, còn được gọi là *độ nhạy* (*sensitivity*) hoặc *tỉ lệ dương tính thật* (*true positive rate* – *TPR*): là tỉ lệ mẫu dương được phát hiện đúng bởi bộ phân loại (Phương trình 3.2).

$$\text{recall} = \frac{TP}{TP + FN}$$

Phương trình 3.2. Recall

Với FN là số mẫu âm tính giả.

Nếu bạn vẫn còn khúc mắc về ma trận nhầm lẫn, hãy tham khảo Hình 3.2.



Hình 3.2. Minh hoạ ma trận nhầm lẫn với các mẫu âm tính thật (trên trái), dương tính giả (trên phải), âm tính giả (dưới trái), và dương tính thật (dưới phải)

Precision và Recall

Scikit-Learn cung cấp nhiều hàm để tính các phép đo chất lượng cho một bộ phân loại, bao gồm precision và recall:

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1522)
0.7290850836596654
>>> recall_score(y_train_5, y_train_pred) # == 4096 / (4096 + 1325)
0.7555801512636044
```

Giờ thì bộ nhận diện số 5 không còn quá hào nhoáng như khi được đánh giá bằng độ chính xác. Trong tổng số lần bộ phân loại gán nhãn một ảnh là 5, nó chỉ đúng 72.9% số lần. Hơn nữa, trên tổng số ảnh số 5, nó chỉ phát hiện được 75.6% số ảnh.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thông thường, sẽ tiện hơn khi gộp precision và recall lại thành một phép đo gọi là *chỉ số* F_1 (F_1*–score*), đặc biệt là khi ta cần một cách đơn giản để so sánh hai bộ phân loại. Chỉ số F_1 là *trung bình điều hoà* (*harmonic mean*) của precision và recall (Phương trình 3.3). Với phép trung bình thông thường thì các giá trị được coi là như nhau, còn trung bình điều hoà đánh trọng số lớn hơn cho các giá trị nhỏ. Do đó, một bộ phân loại chỉ có thể đạt chỉ số F_1 cao nếu cả precision và recall đều cao.

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

Phương trình 3.3. F_1

Để tính chỉ số F_1, chỉ cần gọi hàm `f1_score()`:

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7420962043663375
```

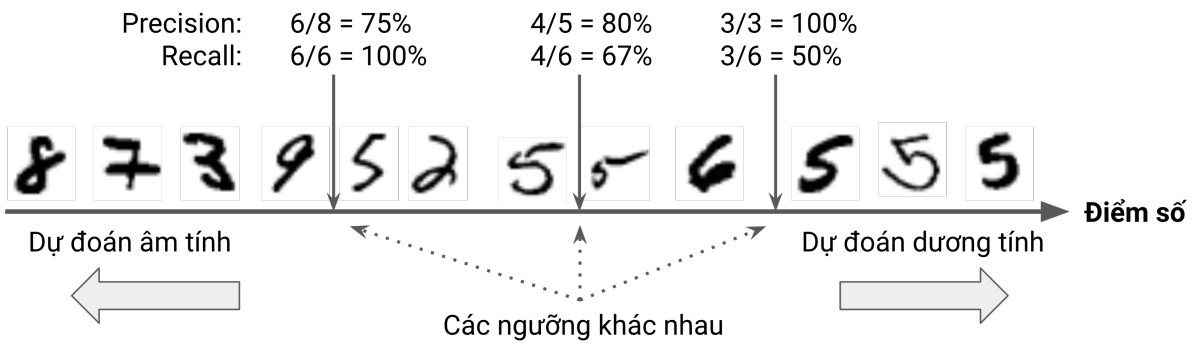
Chỉ số F_1 đánh giá cao các bộ phân loại có precision và recall không quá chênh lệch. Tuy nhiên, không phải lúc nào đây cũng là điều ta mong muốn: trong vài ngữ cảnh, ta hầu như chỉ cần quan tâm đến precision thay vì recall, và ngược lại. Ví dụ, nếu ta đang huấn luyện một bộ phân loại để nhận diện các video có nội dung phù hợp cho trẻ em, ta sẽ ưu tiên một bộ phân loại có thể bỏ lỡ nhiều video tốt (recall thấp) nhưng cần đảm bảo các video được giữ lại đều an toàn (precision cao), hơn là một bộ phân loại có recall cao ngất ngưởng nhưng để lọt một vài video độc hại (trong các trường hợp đó, ta có thể sẽ cần thêm cả sức người để kiểm tra các video được chọn bởi bộ phân loại). Mặt khác, giả sử ta đang huấn luyện một bộ phân loại để nhận diện ăn cắp vặt từ camera giám sát: sẽ không có vấn đề gì nếu bộ phân loại chỉ đạt 30% precision nhưng có 99% recall (nhân viên bảo vệ sẽ nhận được vài báo động giả, nhưng gần như tất cả kẻ cắp đều sẽ bị phát hiện).

Không may, ta không thể vẹn cả đôi đường: tăng precision sẽ làm giảm recall và ngược lại. Đây được gọi là *đánh đổi precision/recall* (*precision/recall trade-off*).

Đánh đổi Precision/Recall

Để hiểu rõ sự đánh đổi này, hãy xem xét cách `SGDClassifier` phân loại. Với mỗi mẫu, bộ phân lớp tính điểm dựa trên một *hàm quyết định* (*decision function*). Nếu số điểm đó lớn hơn một ngưỡng, nó sẽ gán nhãn dương cho mẫu đó, còn ngược lại nó sẽ gán nhãn âm. Hình 3.3 minh hoạ một vài chữ số, với điểm dự đoán tăng dần từ trái qua phải. Giả sử *ngưỡng quyết định* (*decision threshold*) nằm ở vị trí mũi tên chính giữa (giữa hai chữ số 5): ta sẽ có 4 mẫu dương tính thật (nhãn đúng là 5) bên phải ngưỡng đó, và 1 mẫu dương tính giả (nhãn thực là 6). Do đó, với ngưỡng trên, precision là 80% (4 trên 5). Tuy nhiên, trong 6 giá trị có nhãn 5, bộ phân loại chỉ phát hiện được 4, nên recall là 67% (4 trên 6). Nếu tăng mức ngưỡng (dịch sang mũi tên bên phải), mẫu dương tính giả (chữ số 6) sẽ trở thành âm tính thật, nên precision tăng lên (đến 100% trong trường hợp này), nhưng một mẫu dương tính thật sẽ biến thành âm tính giả khiến recall giảm còn 50%. Ngược lại, việc giảm mức ngưỡng sẽ tăng recall và giảm precision.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 3.3. Minh hoạ đánh đổi precision/recall, các bức ảnh được sắp xếp theo điểm phân loại, những bức nào có điểm lớn hơn mức ngưỡng được xem là các mẫu dương. Mức ngưỡng càng cao, recall càng nhỏ, nhưng (nhìn chung) precision sẽ càng cao

Scikit-Learn không cho ta đặt mức ngưỡng một cách trực tiếp, nhưng cho phép ta lấy các điểm số mà mô hình dùng để đưa ra dự đoán. Thay vì gọi phương thức `predict()` của bộ phân loại, ta có thể gọi `decision_function()` để trả về điểm số của mỗi mẫu, rồi sử dụng bất cứ ngưỡng nào ta muốn trên các điểm số đó để dự đoán:

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2412.53175101])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])
```

Đoạn mã trên đang dùng mức ngưỡng bằng 0, nên trả về cùng giá trị với phương thức `predict()` (là `True`). Hãy thử tăng mức ngưỡng này:

```
>>> threshold = 8000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

Kết quả này xác nhận rằng tăng mức ngưỡng sẽ làm giảm recall. Bức ảnh được kiểm tra là chữ số 5, bộ phân loại đã dự đoán đúng khi mức ngưỡng bằng 0, nhưng lại thất bại khi mức ngưỡng tăng lên 8,000.

Vậy làm sao để xác định mức ngưỡng phù hợp? Đầu tiên, ta sử dụng hàm `cross_val_predict()` để dự đoán cho mọi mẫu trong tập huấn luyện, nhưng thiết lập lại để trả về điểm quyết định thay vì dự đoán:

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

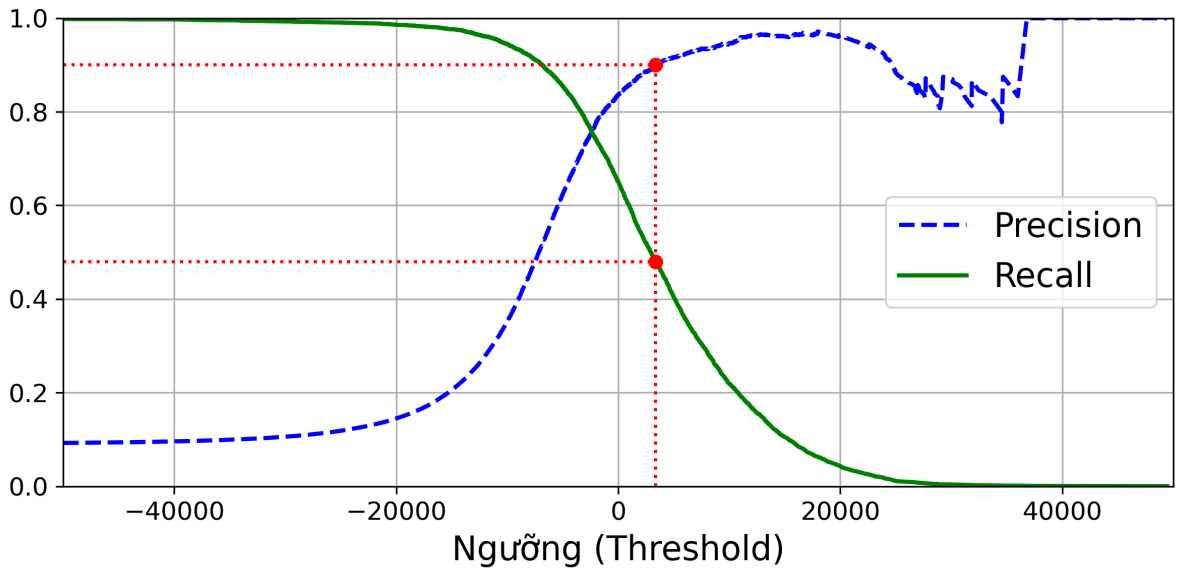
Với những điểm số này, dùng `precision_recall_curve()` để tính precision và recall cho tất cả các mức ngưỡng có thể:

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Cuối cùng, sử dụng Matplotlib để vẽ đồ thị của precision và recall theo mức ngưỡng (Hình 3.4):

```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    [...] # highlight the threshold and add the legend, axis label, and grid

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



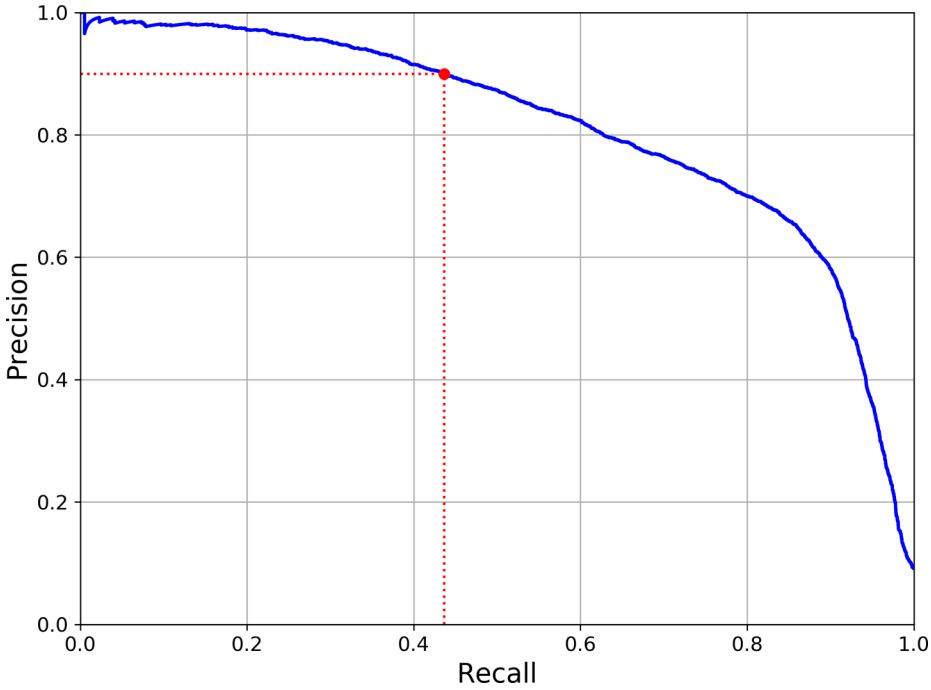
Hình 3.4. Precision và Recall với từng ngưỡng quyết định

Ghi chú

Bạn có thể đang thắc mắc tại sao đường cong precision lại gồ ghề hơn đường cong recall trong Hình 3.4. Lý do là precision đôi khi có thể giảm khi bạn nâng mức ngưỡng (mặc dù nhìn chung nó sẽ tăng). Để hiểu tại sao, hãy nhìn lại Hình 3.3 và để ý chuyện gì xảy ra khi ta bắt đầu từ mức ngưỡng trung tâm và dịch sang bên phải một chữ số: precision giảm từ 4/5 (80%) xuống 3/4 (75%). Mặt khác, recall chỉ có thể giảm khi mức ngưỡng tăng lên, điều này lý giải cho việc đường cong của nó mượt hơn.

Một cách khác để chọn một mức đánh đổi precision/recall phù hợp là vẽ đồ thị giữa precision và recall, như trong Hình 3.5 (với cùng một mức ngưỡng được đánh dấu).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 3.5. Precision với recall

Có thể thấy precision bắt đầu giảm nhanh thực sự ở khoảng recall 80%. Ta sẽ muốn chọn một mức đánh đổi precision/recall ngay trước đó - như quanh khoảng recall 60% chẳng hạn. Nhưng tất nhiên, cách chọn mức ngưỡng sẽ phụ thuộc vào dự án cụ thể.

Giả sử ta muốn đạt được precision 90%. Nhìn vào biểu đồ đầu tiên, ta thấy rằng mức ngưỡng vào khoảng 8000 là phù hợp. Để chính xác hơn nữa, ta có thể tìm mức ngưỡng nhỏ nhất đem lại precision ít nhất là 90% (hàm `np.argmax()` sẽ cho ta chỉ số đầu tiên của giá trị lớn nhất, trong trường hợp này là giá trị `True` đầu tiên):

```
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)] # ~7816
```

Để dự đoán (tạm thời trên tập huấn luyện), thay vì gọi phương thức `predict()` của bộ phân loại, ta có thể chạy đoạn mã sau:

```
y_train_pred_90 = (y_scores >= threshold_90_precision)
```

Hãy kiểm tra precision và recall trên các dự đoán đó:

```
>>> precision_score(y_train_5, y_train_pred_90)
0.9000380083618396
>>> recall_score(y_train_5, y_train_pred_90)
0.4368197749492714
```

Tuyệt vời, ta đã có một bộ phân loại với precision 90%! Như đã thấy, để có một bộ phân loại với mức precision bất kì khá đơn giản: chỉ cần chọn một mức ngưỡng đủ cao là xong. Tuy nhiên, một bộ phân loại có precision cao sẽ không quá hữu dụng nếu recall của nó quá thấp!

Mẹo

Nếu ai đó nói, "Hãy nhắm tới mức precision 99%," bạn nên hỏi lại, "Ở mức recall bao nhiêu?"

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Đường cong ROC

Đường cong ROC (*receiver operating characteristic*) là một công cụ thông dụng khác được sử dụng với các bộ phân loại nhị phân. Nó rất giống đường cong precision/recall, nhưng thay vì vẽ precision theo recall, đường cong ROC vẽ *tỉ lệ dương tính thật* (*true positive rate*), tên gọi khác của recall, theo *tỉ lệ dương tính giả* (*false positive rate* - FPR). FPR là tỉ lệ các mẫu âm tính bị phân loại sai thành dương tính, và bằng $1 - TNR$ (*true negative rate*). TNR là tỉ lệ các mẫu âm tính được phân loại đúng, còn được gọi là *specificity*. Do đó, đường cong ROC là biểu đồ thể hiện *sensitivity* (recall) theo $1 -$ *specificity*.

Để vẽ đường cong ROC, đầu tiên ta dùng hàm `roc_curve()` để tính TPR và FPR cho nhiều giá trị mức ngưỡng khác nhau:

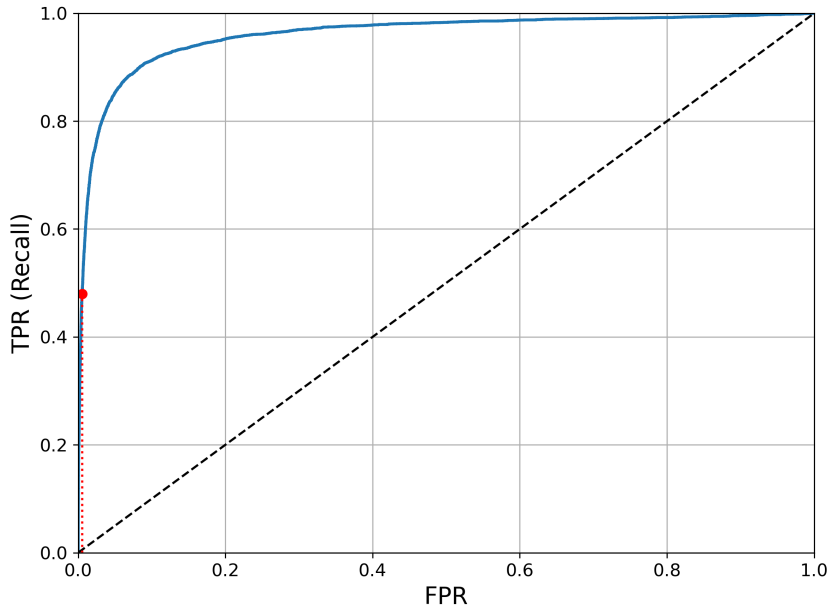
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

Sau đó ta có thể sử dụng Matplotlib để vẽ TPR theo FPR. Đoạn mã sau cho ra kết quả là đồ thị Hình 3.6:

```
def plot_roc_curve(fpr, tpr, label=None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal
    [...] # Add axis labels and grid

plot_roc_curve(fpr, tpr)
plt.show()
```

Một lần nữa ta thấy có một sự đánh đổi: recall (TPR) càng cao, tỉ lệ dương tính giả (FPR) càng lớn. Đường nét đứt biểu diễn đường cong ROC của một bộ phân loại hoàn toàn ngẫu nhiên. Một bộ phân loại tốt sẽ có đường cong ROC càng xa đường nét đứt càng tốt (về phía góc trên bên trái).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 3.6. Đường cong ROC vẽ tỉ lệ dương tính giả theo tỉ lệ dương tính thật cho tất cả các mức ngưỡng khả dĩ. Hình tròn đỏ đánh dấu mức ngưỡng được chọn (có recall 43.68%)

Một cách để so sánh các bộ phân loại là tính *diện tích dưới đường cong* (*area under the curve* - AUC). Một bộ phân loại hoàn hảo sẽ có ROC AUC bằng 1, trong khi một bộ phân loại hoàn toàn ngẫu nhiên sẽ có ROC AUC bằng 0.5. Scikit-Learn cung cấp một hàm tính ROC AUC:

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9611778893101814
```

Mẹo

Vì đường cong ROC rất giống đường cong precision/recall (PR), có thể bạn đang băn khoăn về việc lựa chọn đường cong nào để sử dụng. Quy tắc vàng là nên chọn đường cong PR bất cứ khi nào số lượng mẫu dương ít hoặc khi bạn quan tâm đến dương tính giả nhiều hơn âm tính giả. Nếu không, hãy sử dụng đường cong ROC. Ví dụ, sau khi quan sát đường cong ROC ở phía trên (và cả điểm ROC AUC), bạn có thể nghĩ bộ phân loại này rất tốt. Nhưng điều này phần lớn là vì có ít mẫu dương (các số 5) hơn so với số mẫu âm (không phải 5). Ngược lại, từ đường cong PR ta có thể thấy rõ rằng bộ phân loại này vẫn có thể được cải thiện (đường cong có thể gần góc trên bên phải hơn nữa).

Giờ hãy huấn luyện một `RandomForestClassifier` và so sánh đường cong ROC và điểm ROC AUC với `SGDClassifier`. Đầu tiên, ta cần tính điểm cho mỗi mẫu trong tập huấn luyện. Tuy nhiên, do cách hoạt động của Rừng Ngẫu nhiên (tham khảo Chương 7), lớp `RandomForestClassifier` không có phương thức `decision_function()` mà chỉ có `predict_proba()`. Các bộ phân loại của Scikit-Learn thường sẽ có một trong hai phương thức, hoặc cả hai. Phương thức `predict_proba()` trả về một ma trận có số hàng là số mẫu và số cột là số lớp, mỗi phần tử là xác suất từng mẫu thuộc về từng lớp (ví dụ, xác suất một ảnh là 5 rơi vào khoảng 70%).

```
from sklearn.ensemble import RandomForestClassifier
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

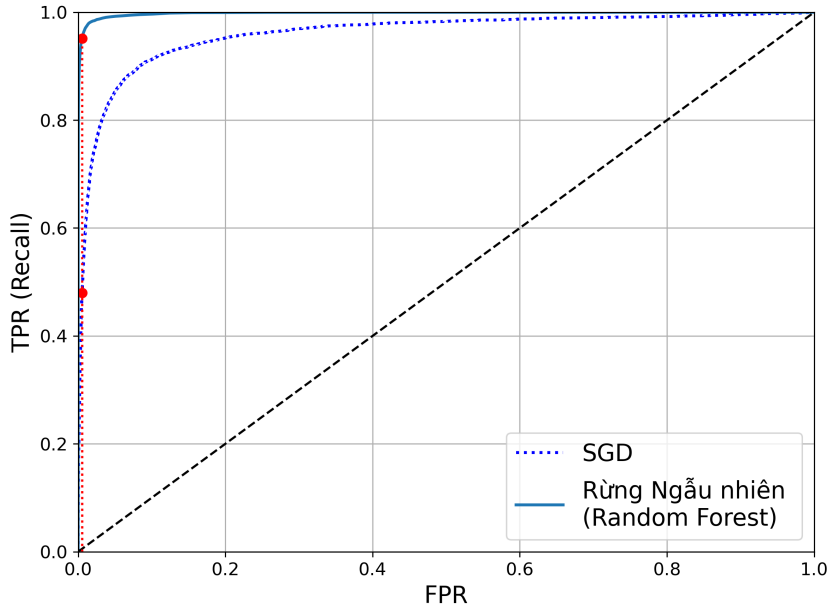
```
forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

Hàm `roc_curve()` nhận đầu vào là nhãn và điểm dự đoán, nhưng thay vì điểm ta có thể truyền vào xác suất của các lớp. Hãy sử dụng xác suất của lớp dương làm điểm:

```
y_scores_forest = y_probas_forest[:, 1]   # score = proba of positive class
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
```

Giờ ta đã có thể vẽ đường cong ROC cho Rừng Ngẫu nhiên. Ta cũng nên vẽ cả đường cong ROC của SGD trước đó để so sánh (Hình 3.7):

```
plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "Random Forest")
plt.legend(loc="lower right")
plt.show()
```



Hình 3.7. So sánh hai đường cong ROC: bộ phân loại Rừng Ngẫu nhiên tốt hơn SGD vì có đường cong ROC gần góc trên bên trái hơn và có AUC lớn hơn

Như có thể thấy trong Hình 3.7, đường cong ROC của `RandomForestClassifier` tốt hơn nhiều so với của `SGDClassifier`: nó gần góc trên bên trái hơn. Do đó, điểm ROC AUC của `RandomForestClassifier` cũng cao hơn đáng kể:

```
>>> roc_auc_score(y_train_5, y_scores_forest)
0.9983436731328145
```

Thử tính precision và recall, ta sẽ có precision 99.0% và recall 86.6%. Không quá tệ!

Đến bây giờ, bạn đã biết cách huấn luyện các bộ phân loại nhị phân, chọn phép đo chất lượng phù hợp với tác vụ, đánh giá bộ phân loại bằng kiểm định chéo, chọn mức đánh đổi precision/recall phù hợp, và dùng đường cong ROC cũng như điểm ROC AUC để so sánh các mô hình. Tiếp đến ta sẽ thử phân loại nhiều hơn một lớp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phân loại Đa lớp

Trong khi bộ phân loại nhị phân chỉ phân biệt hai lớp, *bộ phân loại đa lớp* (*multiclass classifier*, còn được gọi là *bộ phân loại đa thức – multinomial classifier*) có thể phân biệt nhiều hơn hai lớp.

Một vài bộ phân loại (như SGD, Rừng Ngẫu nhiên và Naive Bayes) có khả năng làm việc với nhiều lớp một cách tự nhiên. Những thuật toán khác (như Hồi quy Logistic hay Máy Vector Hỗ trợ) hoàn toàn là các bộ phân loại nhị phân. Tuy nhiên, có rất nhiều chiến lược cho phép ta sử dụng nhiều bộ phân loại nhị phân cho bài toán phân loại đa lớp.

Một cách để xây dựng hệ thống phân loại với 10 lớp ảnh các chữ số (từ 0 đến 9) là huấn luyện 10 bộ phân loại nhị phân, một bộ cho mỗi chữ số (phát hiện 0, phát hiện 1, phát hiện 2, v.v.). Sau đó khi muốn phân loại một ảnh, ta tính điểm của mỗi bộ phân loại cho ảnh đó và chọn lớp có điểm cao nhất. Đây được gọi là chiến lược *một-còn lại – one-versus-the-rest* (OvR; còn được gọi là *một-toàn bộ – one-versus-all*).

Một chiến lược khác là huấn luyện một bộ phân loại nhị phân cho mỗi cặp chữ số: phân biệt 0 và 1, phân biệt 0 và 2, 1 và 2, v.v. Chiến lược này được gọi là *một-một* (*one-versus-one* - OvO). Nếu có N lớp, ta cần huấn luyện $N \times (N - 1) / 2$ bộ phân loại. Như vậy với bài toán MNIST, sẽ cần tới 45 bộ phân loại cần được huấn luyện! Khi muốn phân loại một ảnh, ta phải đưa ảnh qua tất cả các bộ phân loại và xem lớp nào thắng nhiều nhất. Ưu điểm chính của OvO là mỗi bộ phân loại chỉ cần được huấn luyện trên một phần nhỏ của tập huấn luyện chứa hai lớp cần phân biệt.

Một vài thuật toán (như Máy Vector Hỗ trợ) hoạt động chậm khi tăng kích thước tập huấn luyện. Với các thuật toán đó, OvO là chiến lược phù hợp vì sẽ nhanh hơn khi huấn luyện nhiều bộ phân loại trên các tập huấn luyện nhỏ thay vì huấn luyện một số ít bộ phân loại trên các tập huấn luyện lớn. Tuy nhiên, với phần lớn thuật toán phân loại nhị phân, chiến lược OvR được ưa dùng hơn.

Scikit-Learn có thể phát hiện khi nào ta sử dụng một bộ phân loại nhị phân cho tác vụ phân loại đa lớp, và tự động chạy OvR hoặc OvO, tuỳ vào thuật toán. Hãy thử việc này với Máy Vector Hỗ trợ (tham khảo Chương 5), sử dụng lớp `sklearn.svm.SVC`:

```
>>> from sklearn.svm import SVC
>>> svm_clf = SVC()
>>> svm_clf.fit(X_train, y_train) # y_train, not y_train_5
>>> svm_clf.predict([some_digit])
array([5], dtype=uint8)
```

Rất dễ! Đoạn mã trên huấn luyện bộ phân loại `SVC` trên tập huấn luyện sử dụng các lớp mục tiêu ban đầu từ 0 đến 9 (`y_train`), thay vì các lớp mục tiêu 5-versus-the-rest (`y_train_5`). Sau đó, đoạn mã thực hiện dự đoán (dự đoán đúng trong trường hợp này). Ẩn dưới mã nguồn, thực chất Scikit-Learn đã sử dụng chiến lược OvO: huấn luyện 45 bộ phân loại nhị phân, tính điểm của ảnh cần dự đoán, và chọn lớp thắng nhiều nhất.

Khi gọi phương thức `decision_function()`, ta sẽ thấy nó trả về 10 điểm số cho mỗi mẫu (thay vì chỉ 1). Các điểm số đó tương ứng với các lớp:

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])
>>> some_digit_scores
array([[ 2.92492871,  7.02307409,  3.93648529,  0.90117363,  5.96945908,
         9.5       ,  1.90718593,  8.02755089, -0.13202708,  4.94216947]])
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Quả thực, điểm số cao nhất ứng với lớp 5:

```
>>> np.argmax(some_digit_scores)
5
>>> svm_clf.classes_
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
>>> svm_clf.classes_[5]
5
```

Lưu ý

Khi một bộ phân loại được huấn luyện xong, nó lưu danh sách các lớp mục tiêu trong thuộc tính `classes_`, sắp xếp theo giá trị. Trong trường hợp này, chỉ số của mỗi lớp trong mảng `classes_` cũng chính là lớp đó (ví dụ lớp 5 có chỉ số là 5), nhưng thường thì ta sẽ không may mắn như vậy.

Nếu muốn buộc Scikit-Learn sử dụng chiến lược một-một hoặc một-còn lại, ta có thể sử dụng lớp `OneVsOneClassifier` hoặc `OneVsRestClassifier`. Ta chỉ cần đưa một bộ phân loại (không nhất thiết phải là nhị phân) vào hàm khởi tạo của một trong hai lớp trên. Ví dụ, đoạn mã sau tạo một bộ phân loại đa lớp sử dụng chiến lược OvR, dựa trên `SVC`:

```
>>> from sklearn.multiclass import OneVsRestClassifier
>>> ovr_clf = OneVsRestClassifier(SVC())
>>> ovr_clf.fit(X_train, y_train)
>>> ovr_clf.predict([some_digit])
array([5], dtype=uint8)
>>> len(ovr_clf.estimators_)
10
```

Việc huấn luyện `SGDClassifier` (hoặc `RandomForestClassifier`) cũng rất đơn giản:

```
>>> sgd_clf.fit(X_train, y_train)
>>> sgd_clf.predict([some_digit])
array([5], dtype=uint8)
```

Ở đây Scikit-Learn không phải chạy OvR hoặc OvO vì bộ phân loại SGD có thể trực tiếp phân loại mẫu vào nhiều lớp. Phương thức `decision_function()` giờ trả về một giá trị mỗi lớp. Hãy nhìn vào điểm số mà SGD gán cho mỗi lớp:

```
>>> sgd_clf.decision_function([some_digit])
array([[-15955.22628, -38080.96296, -13326.66695,   573.52692, -17680.68466,
          2412.53175, -25526.86498, -12290.15705, -7946.05205, -10631.35889]])
```

Có thể thấy bộ phân loại khá tự tin vào dự đoán của nó: hầu hết mọi giá trị đều có điểm âm rất lớn, trừ lớp 5 có điểm 2412.5. Mô hình có chút do dự ở lớp 3 với điểm số là 573.5. Bây giờ ta sẽ cần đánh giá bộ phân loại này. Như mọi khi, ta có thể thực hiện kiểm định chéo bằng cách sử dụng hàm `cross_val_score()` để đánh giá độ chính xác của `SGDClassifier`:

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.8489802 , 0.87129356, 0.86988048])
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các fold kiểm tra đều có kết quả hơn 84%. Nếu sử dụng một bộ phân loại ngẫu nhiên, độ chính xác sẽ chỉ là 10%. Vậy nên đây không phải là một kết quả tồi, nhưng ta vẫn có thể làm tốt hơn nữa. Chỉ cần co giãn đầu vào (như đề cập trong Chương 2), độ chính xác sẽ tăng lên trên 89%:

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
>>> cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
array([0.89707059, 0.8960948 , 0.90693604])
```

Phân tích Lỗi

Nếu đây là một dự án thực tế, ta sẽ cần thực hiện những bước tiếp theo trong danh sách công việc cho dự án Học Máy (tham khảo Phụ lục B). Ta sẽ khám phá các lựa chọn trong việc chuẩn bị dữ liệu, thử thêm một vài mô hình (tập trung vào các mô hình hoạt động tốt nhất và tinh chỉnh siêu tham số bằng `GridSearchCV`), và tự động hóa chúng càng nhiều càng tốt. Tuy nhiên, ở đây giả sử ta đã tìm thấy một mô hình triển vọng và muốn tìm cách cải thiện nó. Một cách để làm việc này là phân tích các lỗi mô hình phạm phải.

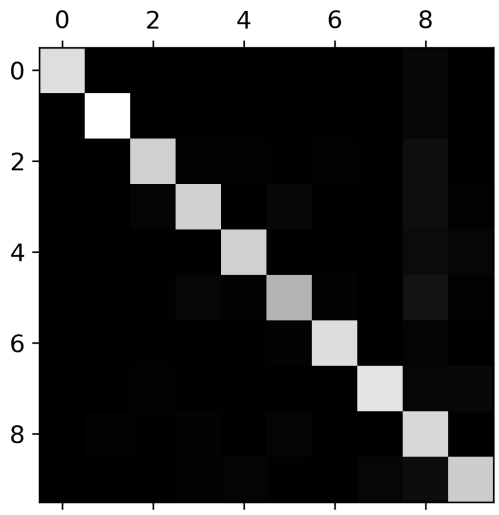
Trước tiên hãy xem xét ma trận nhầm lẫn. Như lần trước, ta cần sử dụng `cross_val_predict()` để dự đoán, rồi dùng hàm `confusion_matrix()`:

```
>>> y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
>>> conf_mx = confusion_matrix(y_train, y_train_pred)
>>> conf_mx
array([[5578,    0,   22,    7,    8,   45,   35,    5,  222,    1],
       [   0, 6410,   35,   26,    4,   44,    4,    8,  198,   13],
       [  28,   27, 5232,  100,   74,   27,   68,   37,  354,   11],
       [  23,   18,  115, 5254,    2,  209,   26,   38,  373,   73],
       [  11,   14,   45,   12, 5219,   11,   33,   26,  299,  172],
       [  26,   16,   31,  173,   54, 4484,   76,   14,  482,   65],
       [  31,   17,   45,    2,   42,   98, 5556,    3,  123,    1],
       [  20,   10,   53,   27,   50,   13,    3, 5696,  173,  220],
       [  17,   64,   47,   91,    3,  125,   24,   11, 5421,   48],
       [  24,   18,   29,   67,  116,   39,    1,  174,  329, 5152]])
```

Khá khó để nắm bắt nhanh được đầy đủ tình hình khi có quá nhiều con số. Một cách thuận tiện hơn là nhìn vào hình ảnh biểu diễn của ma trận này, sử dụng hàm `matshow()` của Matplotlib:

```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show()
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



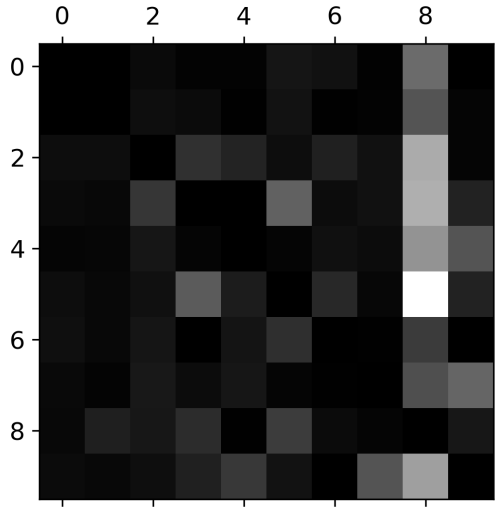
Ma trận nhầm lẫn trông khá ổn, bởi đa số tấm ảnh đều nằm trên đường chéo chính, đồng nghĩa với việc chúng được phân loại chính xác. Các ảnh của lớp `5` có phần tối hơn các lớp còn lại, có thể do số 5 có ít ảnh hơn trong tập dữ liệu hoặc bộ phân loại hoạt động trên lớp này kém hơn những lớp còn lại. Thực tế, ta có thể kiểm chứng rằng cả hai đều đúng.

Hãy tập trung vào việc minh họa các lỗi. Đầu tiên, ta cần chia mỗi giá trị trong ma trận nhầm lẫn cho số ảnh của lớp tương ứng để so sánh tỷ lệ lỗi thay vì số lượng lỗi (điều này vô tình khiến các lớp có nhiều ảnh trông có vẻ tệ hơn):

```
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
```

Đưa các phần tử trên đường chéo về 0 nhằm chỉ giữ lại các lỗi và vẽ biểu đồ kết quả:

```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```



Ta có thể thấy rõ các dạng lỗi mà bộ phân loại phạm phải. Nhắc lại rằng các hàng biểu thị lớp thực và các cột biểu thị lớp dự đoán. Cột của lớp 8 khá sáng, đồng nghĩa với việc có nhiều ảnh bị phân loại nhầm thành 8. Tuy nhiên, hàng của lớp 8 thì không tệ lắm, cho thấy hầu hết các

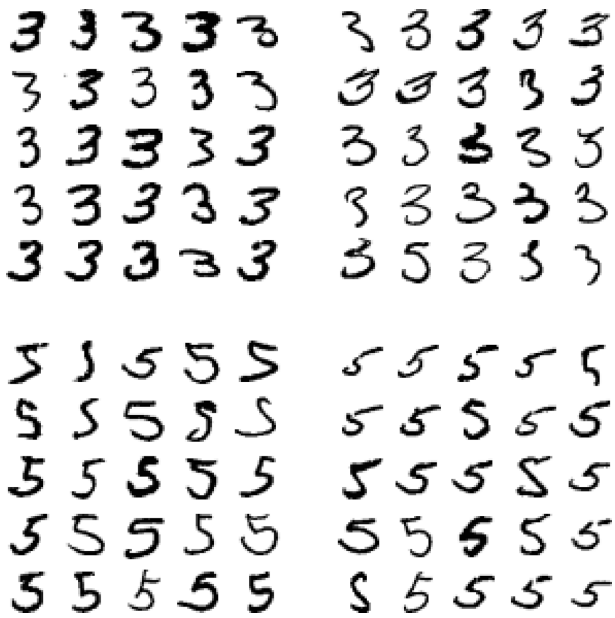
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

mẫu trong lớp này đều được phân loại chính xác. Có thể thấy, ma trận nhầm lẫn không nhất thiết phải đối xứng. Ngoài ra, lớp 3 và lớp 5 cũng thường bị nhầm lẫn với nhau (theo cả hai chiều).

Việc phân tích ma trận nhầm lẫn thường giúp ta nhìn ra hướng cải thiện bộ phân loại. Nhìn vào ma trận này, có vẻ như ta nên tập trung giảm thiểu việc phân loại nhầm thành 8. Ví dụ, ta có thể thu thập thêm dữ liệu huấn luyện của các số nhìn giống 8 (nhưng thực ra không phải) nhằm giúp bộ phân loại học cách phân biệt giữa chúng và những số 8 thật. Ta cũng có thể thiết kế những đặc trưng hỗ trợ bộ phân loại, ví dụ như viết một thuật toán đếm số đường tròn đóng (ví dụ, 8 có 2 đường tròn đóng, 6 có một và 5 không có). Một phương pháp khác là tiền xử lý ảnh (có thể sử dụng Scikit-Image, Pillow hoặc OpenCV) để làm nổi bật các khuôn mẫu, chẳng hạn như các đường tròn đóng.

Phân tích từng lỗi riêng biệt cũng có thể giúp tìm ra cách vận hành của bộ phân loại cũng như nguyên nhân thất bại, nhưng việc này sẽ tốn thời gian và khó khăn hơn rất nhiều. Ví dụ, hãy thử hiển thị một số mẫu của 3 và 5 (hàm `plot_digits()` chỉ dùng hàm `imshow()` của Matplotlib; tham khảo Jupyter Notebook của chương này để biết thêm chi tiết):

```
cl_a, cl_b = 3, 5
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]

plt.figure(figsize=(8,8))
plt.subplot(221); plot_digits(X_aa[:25], images_per_row=5)
plt.subplot(222); plot_digits(X_ab[:25], images_per_row=5)
plt.subplot(223); plot_digits(X_ba[:25], images_per_row=5)
plt.subplot(224); plot_digits(X_bb[:25], images_per_row=5)
plt.show()
```



Hai ma trận 5×5 ở bên trái là những ảnh được phân loại là 3, và hai ma trận bên phải là những ảnh được phân loại là 5. Một số hình bị phân loại sai (ở ma trận phải-trên và trái-dưới) được viết cẩu thả đến nỗi người thường cũng khó phân loại được (như số 5 ở dòng một, cột hai trong ma trận trái-dưới thực sự nhìn như một số 3 bị viết ẩu). Tuy nhiên, đa phần các ảnh bị phân

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

loại sai rõ ràng là lỗi của mô hình, và rất khó để hiểu tại sao bộ phân loại lại phạm những lỗi như vậy.[3] Nguyên nhân là vì ta đã sử dụng một mô hình tuyến tính `SGDClassifier` đơn giản. Mô hình này chỉ gán một trọng số của mỗi lớp cho các điểm ảnh, và khi thực hiện dự đoán trên ảnh mới, điểm cho mỗi lớp là tổng trọng số của các cường độ điểm ảnh. Vì 3 và 5 chỉ khác nhau ở một vài điểm ảnh, mô hình này dễ bị nhầm lẫn giữa hai chữ số này.

Điểm khác biệt chính giữa 3 và 5 là nét nối đường thẳng phía trên với phần vòng cung bên dưới. Nếu ta viết số 3 và dịch nét đó về phía bên trái một chút, bộ phân loại có thể sẽ phân loại thành 5, và ngược lại. Nói cách khác, bộ phân loại này khá nhạy cảm với phép tịnh tiến và xoay ảnh. Vậy nên một cách để giảm sự nhầm lẫn giữa 3 và 5 là tiền xử lý ảnh để đảm bảo chúng được căn giữa và không bị xoay quá nhiều. Việc này cũng có thể giúp giảm thiểu những loại lỗi khác.

Phân loại Đa nhãn

Cho đến hiện tại, mỗi mẫu chỉ được gán với đúng một nhãn. Trong một vài trường hợp, ta sẽ muốn bộ phân loại dự đoán nhiều nhãn cho một mẫu. Xét một bộ phân loại cho bài toán nhận diện khuôn mặt: bộ phân loại nên làm gì khi phát hiện bức ảnh có nhiều người? Bộ phân loại nên gán một nhãn cho mỗi người mà nó phát hiện. Giả sử bộ phân loại được huấn luyện để nhận diện ba khuôn mặt: Alice, Bob, và Charlie. Khi bộ phân loại nhận diện một tấm ảnh của Alice và Charlie, nó nên trả về [1, 0, 1] (nghĩa là "có Alice, không có Bob, có Charlie"). Hệ thống phân loại trả về nhiều hơn một nhãn được gọi là hệ thống *phân loại đa nhãn* (*multilabel classification*).

Hãy xem xét ví dụ minh họa đơn giản sau trước khi đi sâu vào bài toán nhận diện khuôn mặt:

```
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= 7)
y_train_odd = (y_train % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]

knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

Đoạn mã trên tạo một mảng `y_multilabel` chứa hai nhãn mục tiêu cho mỗi ảnh chữ số: nhãn đầu tiên cho biết liệu đây có phải một số mang giá trị lớn (7, 8, hoặc 9) và nhãn thứ hai cho biết liệu đó có phải số lẻ không. Hai dòng tiếp theo tạo một bộ phân loại `KNeighborsClassifier` (bộ phân loại này hỗ trợ phân loại đa nhãn) và sử dụng mảng đa mục tiêu để huấn luyện. Giờ ta có thể thực hiện dự đoán, và hãy lưu ý rằng mô hình trả về hai nhãn đầu ra:

```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

Mô hình đã dự đoán chính xác! Số 5 không phải là một số lớn (`False`) và là số lẻ (`True`).

Có nhiều cách để đánh giá một bộ phân loại đa nhãn và việc chọn phép đo phù hợp sẽ tùy thuộc vào từng dự án. Một hướng tiếp cận là tính điểm F_1 (hoặc bất kỳ phép đo chất lượng nào cho bài toán phân loại nhị phân đã đề cập trước đó) cho từng nhãn riêng biệt rồi lấy trung bình. Đoạn mã sau lấy trung bình F_1 trên các nhãn:

[3] Nhưng hãy nhớ rằng não bộ là một hệ thống nhận diện khuôn mẫu tuyệt vời và hệ thống thị giác của ta đã thực hiện nhiều bước tiền xử lý phức tạp trước khi nhận thức bất kỳ thông tin nào. Vậy nên việc ta có thể làm được như vậy một cách dễ dàng không có nghĩa là tác vụ này đơn giản.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

Cách tính này giả định rằng mọi nhãn đều quan trọng như nhau, dù điều này không phải lúc nào cũng đúng. Ví dụ như khi ta có nhiều ảnh của Alice hơn Bob hoặc Charlie, ta có thể sẽ muốn gán trọng số lớn hơn cho các dự đoán của bộ phân loại với ảnh của Alice, tức gán trọng số dựa trên số lượng mẫu của nhãn tương ứng. Để làm vậy, ta chỉ cần đặt `average="weighted"` trong đoạn mã trước đó.[4]

Phân loại Đa Đầu ra

Tác vụ phân loại cuối cùng mà ta sẽ thảo luận là *phân loại đa lớp - đa đầu ra* (hay đơn giản là *phân loại đa đầu ra*). Đây là trường hợp tổng quát của bài toán phân loại đa nhãn, trong đó mỗi nhãn có thể có nhiều lớp (hay mỗi nhãn có thể có nhiều hơn hai giá trị).

Để minh họa, ta sẽ xây dựng một hệ thống loại bỏ nhiễu ra khỏi ảnh. Hệ thống nhận đầu vào là một ảnh chữ số bị nhiễu và (hy vọng) trả về ảnh đã được khử nhiễu, biểu diễn dưới dạng một mảng chứa các giá trị cường độ điểm ảnh, giống như ảnh trong tập MNIST. Lưu ý rằng kết quả mà bộ phân loại trả về gồm nhiều nhãn (mỗi điểm ảnh tương ứng với một nhãn) và mỗi nhãn có thể có nhiều giá trị (cường độ điểm ảnh có giá trị trong khoảng từ 0 đến 255). Vậy nên, đây là một hệ thống phân loại đa đầu ra.

Ghi chú

Ranh giới giữa bài toán phân loại và bài toán hồi quy đôi khi khá mong manh. Chẳng hạn như trong ví dụ này, việc dự đoán cường độ điểm ảnh giống bài toán hồi quy hơn là phân loại. Hơn nữa, hệ thống đa đầu ra không chỉ bị giới hạn trong các bài toán phân loại. Ta còn có thể xây dựng một hệ thống trả về nhiều loại nhãn cho mỗi mẫu, bao gồm cả nhãn lớp và nhãn giá trị.

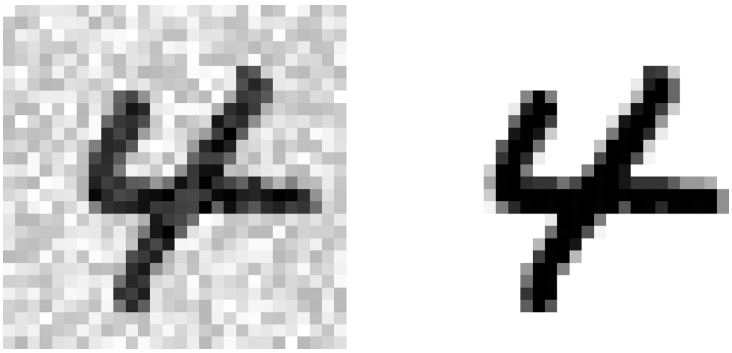
Ta sẽ tạo một tập huấn luyện và tập kiểm tra từ các ảnh MNIST và thêm nhiễu vào mỗi điểm ảnh bằng hàm `randint()` trong NumPy. Ảnh mục tiêu sẽ là ảnh gốc không có nhiễu:

```
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
y_train_mod = X_train
y_test_mod = X_test
```

Hãy xem qua một ảnh trong tập kiểm tra (bạn hẳn đang cau mày vì ta đang xem trộm dữ liệu kiểm tra):

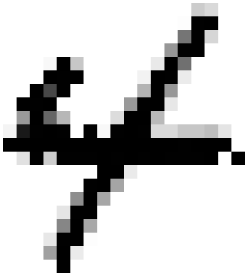
[4] Scikit-Learn cung cấp một vài cách tính trung bình cũng như các phép đo cho tác vụ phân loại đa nhãn. Hãy tham khảo tài liệu để biết thêm chi tiết.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Bên trái là ảnh đầu vào bị nhiễu, còn bên phải là ảnh mục tiêu không có nhiễu. Tiếp theo, hãy huấn luyện một bộ phân loại để khử nhiễu cho tấm ảnh này:

```
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[some_index]])
plot_digit(clean_digit)
```



Kết quả trông khá giống với ảnh mục tiêu! Đây cũng là bước cuối cùng trong hành trình khám phá bài toán phân loại. Giờ ta đã biết cách chọn một phép đo tốt cho tác vụ phân loại cùng mức đánh đổi precision/recall hợp lý, từ đó so sánh các bộ phân loại và tổng quát hơn là xây dựng một hệ thống phân loại cho các tác vụ khác nhau.

Bài tập

1. Xây dựng một bộ phân loại cho tập dữ liệu MNIST có độ chính xác trên 97% trên tập kiểm tra. Gợi ý: `KNeighborsClassifier` hoạt động tốt cho tác vụ này. Bạn chỉ cần tìm những giá trị siêu tham số phù hợp (hãy thử tìm kiếm dạng lưới với hai siêu tham số `weights` và `n_neighbors`).

2. Hãy viết một hàm để dịch chuyển một tấm ảnh MNIST theo hướng bất kỳ (trái, phải, trên, dưới) một đơn vị điểm ảnh.[5] Sau đó, với mỗi ảnh trong tập huấn luyện, tạo bốn bản sao bị dịch chuyển (mỗi bản sao bị dịch theo một hướng) và thêm chúng vào tập huấn luyện. Cuối cùng, huấn luyện mô hình tốt nhất bạn tìm được trên tập dữ liệu đã mở rộng này và đo độ chính xác của nó trên tập kiểm tra. Bạn sẽ thấy mô hình bây giờ hoạt động thậm chí còn tốt hơn! Kỹ thuật tăng số lượng mẫu trong tập huấn luyện một cách nhân tạo này được gọi là *tăng cường dữ liệu* (*data augmentation*) hay *mở rộng tập huấn luyện* (*training set expansion*).

3. Hãy làm việc với tập dữ liệu Titanic trên Kaggle.

[5] Bạn có thể dùng hàm `shift()` trong mô-đun `scipy.ndimage.interpolation`. Ví dụ, `shift(image, [2, 1], cval=0)` dịch chuyển ảnh xuống dưới hai đơn vị điểm ảnh và sang phải một đơn vị điểm ảnh.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

4. Hãy xây dựng một bộ phân loại thư rác (bài tập này khó hơn):

   - Tải tập dữ liệu gồm thư bình thường và thư rác từ các tập dữ liệu công khai của Apache SpamAssassin.
   - Giải nén tập dữ liệu và làm quen với định dạng dữ liệu.
   - Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.
   - Hãy viết một pipeline chuẩn bị dữ liệu nhằm chuyển đổi mỗi email thành một vector đặc trưng. Pipeline chuẩn bị dữ liệu này sẽ biến đổi một email thành một ma trận (thưa) biểu diễn sự hiện diện hoặc vắng mặt của mỗi từ khả dĩ. Ví dụ, nếu tất cả các email đều chỉ có bốn từ "Hello," "how," "are," "you," thì email có nội dung "Hello you Hello Hello you" sẽ được biến đổi thành [1, 0, 0, 1] (nghĩa là ["Hello" có trong thư, "how" không, "are" không và "you" có trong thư]), hoặc [3, 0, 0, 2] nếu bạn thích đếm số lần xuất hiện của mỗi từ.
   - Bạn có thể sẽ muốn thêm vào pipeline này các siêu tham số để điều khiển các lựa chọn, đó là liệu có nên bỏ đi phần tiêu đề của email, chuyển nội dung của email về chữ viết thường, bỏ đi các dấu câu, thay thế tất cả URL thành chuỗi ký tự "URL", thay đổi tất cả các giá trị số thành chuỗi "NUMBER" hoặc thậm chí là thực hiện *stemming* (tức loại bỏ đuôi từ, có nhiều thư viện Python có sẵn hỗ trợ việc này).
   - Cuối cùng, hãy thử nghiệm một vài bộ phân loại khác nhau và xem liệu bạn có thể xây dựng một bộ phân loại đạt được cả precision và recall cao hay không.

Lời giải của bài tập có thể được tìm thấy trong Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 4

Huấn luyện Mô hình

Cho đến nay, ta vẫn coi các mô hình Học Máy và các thuật toán huấn luyện chúng như những hộp đen. Nếu đã làm bài tập trong những chương trước, bạn có thể sẽ ngạc nhiên vì rất nhiều tác vụ có thể được hoàn thành mà không cần hiểu biết cụ thể về tác vụ đó, ví dụ: ta đã tối ưu hóa một hệ thống hồi quy, cải thiện một bộ phân loại hình ảnh chữ số hoặc thậm chí là xây dựng từ đầu một bộ phân loại thư rác mà không cần biết chúng thực sự hoạt động như thế nào. Thật vậy, việc biết chi tiết quá trình thực thi là không thực sự cần thiết trong nhiều tình huống.

Tuy nhiên, việc hiểu rõ cách vận hành có thể giúp ta nhanh chóng tìm được mô hình, thuật toán huấn luyện phù hợp, hay một bộ siêu tham số tốt. Hiểu rõ cách thức hoạt động cũng giúp ta phát hiện và phân tích lỗi hiệu quả hơn. Cuối cùng, hầu hết những chủ đề được thảo luận trong chương này sẽ rất cần thiết trong việc tìm hiểu, xây dựng và huấn luyện mạng nơ-ron (sẽ được thảo luận trong Tập 2 của cuốn sách này).

Trong chương này, ta sẽ bắt đầu với một trong những mô hình đơn giản nhất – mô hình Hồi quy Tuyến tính. Cụ thể, ta sẽ thảo luận về hai phương thức khác nhau để huấn luyện mô hình này:

- Sử dụng phương trình "dạng đóng" (*closed-form*) để tính toán trực tiếp các tham số mô hình sao cho mô hình đó khớp nhất với tập huấn luyện (ví dụ như để tối thiểu hóa hàm mất mát trên tập huấn luyện.)

- Sử dụng một phương pháp tối ưu lặp, được gọi là Hạ Gradient (*Gradient Descent – GD*), để dần dần điều chỉnh các tham số mô hình nhằm giảm thiểu hàm mất mát trên tập huấn luyện, cuối cùng hội tụ về cùng một bộ tham số như phương pháp đầu tiên. Ta cũng sẽ tìm hiểu một số biến thể của phương pháp Hạ Gradient sẽ được sử dụng thường xuyên cho các mạng nơ-ron trong Tập 2: Hạ Gradient theo Batch, Hạ gradient theo Mini-batch, và Hạ Gradient Ngẫu nhiên.

Tiếp đó, ta sẽ xét đến Hồi quy Đa thức (*Polynominal Regression*), một mô hình phức tạp hơn có thể được dùng để khớp dữ liệu phi tuyến. Vì có nhiều tham số hơn mô hình Hồi quy Tuyến tính, mô hình này cũng dễ quá khớp dữ liệu hơn. Do đó, ta sẽ tìm hiểu cách phát hiện quá khớp bằng đồ thị quá trình học và các kỹ thuật điều chuẩn (*regularization*) nhằm giảm nguy cơ quá khớp.

Cuối cùng, ta sẽ xét thêm hai mô hình thường được sử dụng cho các tác vụ phân loại: Hồi quy Logistic và Hồi quy Softmax.

Lưu ý

Chương này sẽ có khá nhiều phương trình toán học liên quan đến các khái niệm cơ bản trong đại số tuyến tính và giải tích. Để hiểu các phương trình này, độc giả cần có kiến thức cơ bản về vector, ma trận, phép chuyển vị, nhân, nghịch đảo ma trận, cũng như đạo hàm riêng; Nếu chưa nắm rõ những khái niệm này, xin vui lòng tham khảo các hướng dẫn nhập môn đại số tuyến tính và giải tích dưới dạng Jupyter Notebook trong tài liệu bổ sung. Nếu không mặn mà lắm với phần toán học, bạn vẫn nên điểm qua chương này dù có thể bỏ qua các phương trình toán. Chúng tôi hi vọng rằng, chỉ riêng các nội dung được trình bày dưới đây cũng đủ giúp bạn đọc hiểu được phần nào các khái niệm nêu trên.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hồi quy Tuyến tính

Trong Chương 1, ta đã xét một mô hình hồi quy đơn giản về mức độ hài lòng trong cuộc sống: `life_satisfaction` $= \theta_0 + \theta_1 \times$ `GDP_per_capita`.

Mô hình này chỉ là một hàm tuyến tính của đặc trưng đầu vào `GDP_per_capita` với các tham số của mô hình θ_0 và θ_1.

Một cách tổng quát hơn, mô hình tuyến tính đưa ra dự đoán bằng cách tính tổng trọng số các đặc trưng đầu vào, sau đó cộng tổng này với một hằng số gọi là *hệ số điều chỉnh* (*bias term*, hoặc còn được gọi là *hệ số chặn – intercept term*), như có thể thấy trong Phương trình 4.1.

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n I_n$$

Phương trình 4.1. Dự đoán của mô hình Hồi quy Tuyến tính

Trong đó:

- $\hat{y}$ là giá trị dự đoán.
- n là số lượng đặc trưng.
- x_i là giá trị đặc trưng thứ i.
- θ_j là tham số thứ j của mô hình (bao gồm cả hệ số điều chỉnh θ_0 và các trọng số đặc trưng $\theta_1, \theta_2, ..., \theta_n$).

Biểu thức này có thể viết gọn dưới dạng vector như trong Phương trình 4.2.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

Phương trình 4.2. Dự đoán của mô hình Hồi quy Tuyến tính (dạng vector)

Trong đó:

- $\boldsymbol{\theta}$ là *vector tham số* của mô hình, chứa hệ số điều chỉnh θ_0 và các trọng số đặc trưng từ θ_1 đến θ_n.
- $\mathbf{x}$ là *vector đặc trưng* của mẫu, chứa x_0 tới x_n, với x_0 luôn bằng 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ là tích vô hướng của vector $\boldsymbol{\theta}$ và $\mathbf{x}$, được tính bằng $\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$.

- $h_{\boldsymbol{\theta}}$ là hàm giả thuyết, sử dụng các tham số mô hình $\boldsymbol{\theta}$.

Ghi chú

Trong Học Máy, vector thường được biểu diễn dưới dạng *vector cột*, là mảng 2D với một cột duy nhất. Nếu $\boldsymbol{\theta}$ và $\mathbf{x}$ là các vector cột, thì giá trị dự đoán được tính là $\hat{y} = \boldsymbol{\theta}^\top \mathbf{x}$, trong đó $\boldsymbol{\theta}^\top$ là chuyển vị của $\boldsymbol{\theta}$ (một ma trận hàng, thay vì ma trận cột) và $\boldsymbol{\theta}^\top \mathbf{x}$ là phép nhân ma trận của $\boldsymbol{\theta}^\top$ và $\mathbf{x}$. Tất nhiên nó vẫn trả về cùng kết quả, chỉ khác là được biểu diễn dưới dạng ma trận một phần tử (*single cell matrix*) thay vì số vô hướng. Trong cuốn sách này, tác giả sử dụng ký hiệu trên để tránh việc chuyển đổi giữa tích vô hướng và phép nhân ma trận.

Vậy là ta đã biết khái niệm mô hình Hồi quy Tuyến tính, vậy còn phương thức để huấn luyện? Hãy nhớ rằng huấn luyện một mô hình là tìm kiếm bộ giá trị của các tham số mô hình để khớp tập huấn luyện một cách tốt nhất. Để làm được điều này, trước tiên ta cần một phép đo độ khớp (tốt hay kém) của mô hình trên tập huấn luyện. Trong Chương 2, ta đã biết rằng phép đo phổ biến nhất của mô hình hồi quy là Căn bậc hai Trung bình Bình phương Sai số (*Root Mean Square Error – RMSE*) (Phương trình 2.1). Vì vậy, để huấn luyện mô hình Hồi quy Tuyến tính, ta cần tìm giá trị $\boldsymbol{\theta}$ để cực tiểu hóa RMSE. Trong thực tế, việc cực tiểu hóa trung bình bình phương sai số (*mean squared error – MSE*) đơn giản hơn RMSE, trong khi kết quả thu được là tương tự (vì cực tiểu hóa một giá trị cũng sẽ cực tiểu hóa căn bậc hai của nó).[1]

Cách tính MSE của một giả thuyết Hồi quy Tuyến Tính $h_{\boldsymbol{\theta}}$ trên tập huấn luyện $\mathbf{X}$ được biểu diễn trong Phương trình 4.3.

$$\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Phương trình 4.3. Hàm chi phí MSE cho mô hình Hồi quy Tuyến tính

Hầu hết các ký hiệu trên đã được giới thiệu tại Chương 2 (tham khảo Ký hiệu). Điểm khác biệt duy nhất là ta viết $h_{\boldsymbol{\theta}}$ thay vì chỉ h để làm rõ việc mô hình được tham số hóa bởi vector $\boldsymbol{\theta}$. Để đơn giản, ta sẽ chỉ viết $\mathrm{MSE}(\boldsymbol{\theta})$ thay vì $\mathrm{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}})$.

Phương trình Pháp tuyến

Để tìm giá trị $\boldsymbol{\theta}$ nhằm cực tiểu hóa hàm mất mát, ta có một *biểu thức dạng đóng*, tức một phương trình toán học cho kết quả trực tiếp. Phương trình 4.4 được gọi là *Phương trình Pháp tuyến – Normal Equation.*

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Phương trình 4.4. Phương trình Pháp tuyến

Trong đó:

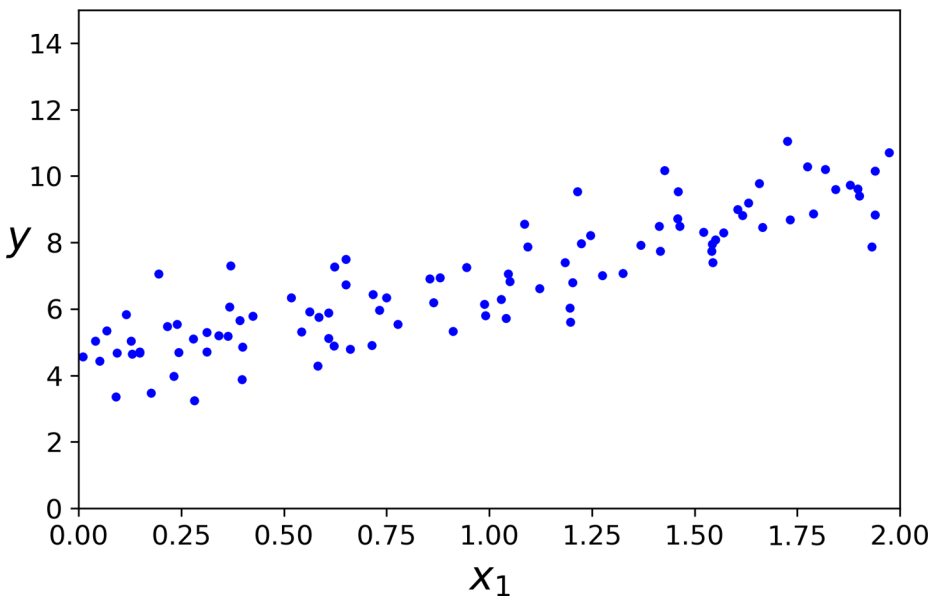
[1] Trong nhiều trường hợp, thuật toán học sẽ cố gắng tối ưu hóa một hàm khác thay vì phép đo chất lượng được sử dụng để đánh giá mô hình. Phần lớn là do hàm này dễ tính toán hơn nhờ các đặc tính vi phân hữu ích mà phép đo chất lượng không có, hoặc do ta muốn ràng buộc mô hình trong quá trình huấn luyện. Vấn đề này sẽ được đề cập khi thảo luận về điều chuẩn.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- $\hat{\boldsymbol{\theta}}$ là giá trị của $\boldsymbol{\theta}$ làm cho hàm mất mát đạt cực tiểu.
- $\mathbf{y}$ là vector chứa các giá trị mục tiêu từ $y^{(1)}$ tới $y^{(m)}$.

Hãy tạo một chút dữ liệu tuyến tính để thử phương trình trên (Hình 4.1):

```
import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```



Hình 4.1. Tập dữ liệu tuyến tính được tạo ngẫu nhiên

Bây giờ, hãy tính $\hat{\boldsymbol{\theta}}$ bằng Phương trình Pháp tuyến. Ta sẽ sử dụng hàm `inv()` trong mô-đun đại số tuyến tính của Numpy (`np.linalg`) để tính nghịch đảo ma trận và phương thức `dot()` để nhân ma trận:

```
X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

Hàm số mà chúng ta đã sử dụng để tạo dữ liệu là $y = 4 + 3x_1 +$ `nhiễu Gauss`. Hãy cùng xem kết quả của phương trình trên:

```
>>> theta_best
array([[4.21509616],
       [2.77011339]])
```

Kết quả mong muốn là $\theta_0 = 4$ và $\theta_1 = 3$ thay vì $\theta_0 = 4.215$ và $\theta_1 = 2.770$. Dù gần đúng, nhiễu khiến cho việc khôi phục chính xác các tham số của hàm số gốc trở nên bất khả thi.

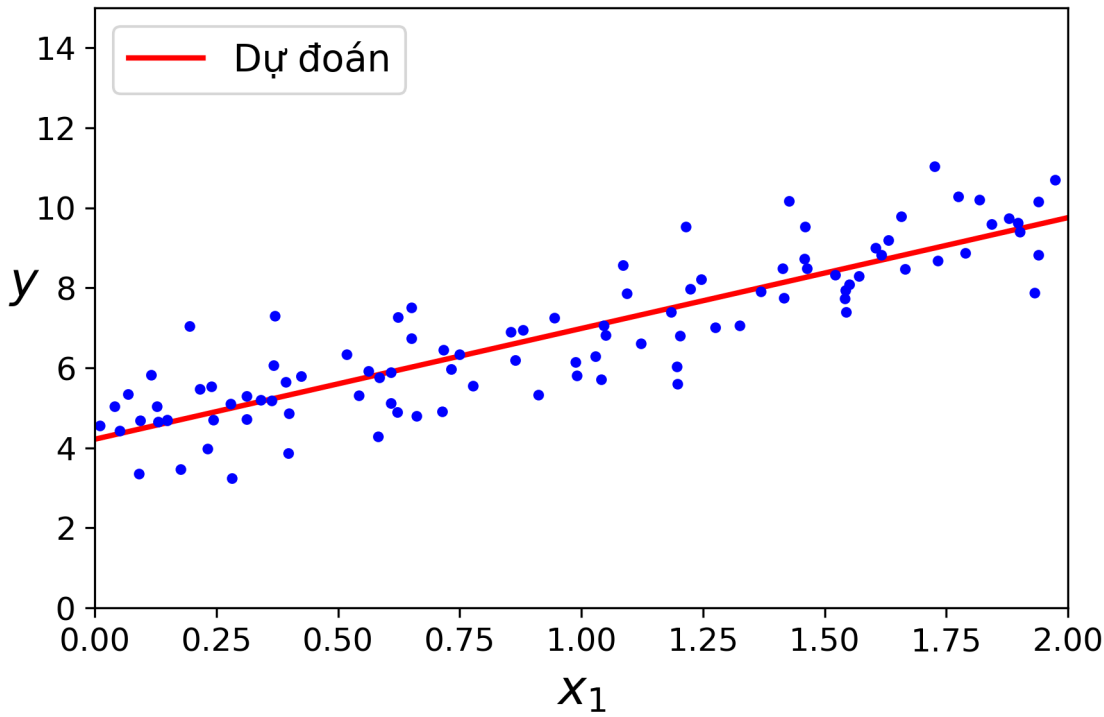
Bây giờ, ta có thể thực hiện dự đoán với $\hat{\boldsymbol{\theta}}$:

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> X_new = np.array([[0], [2]])
>>> X_new_b = np.c_[np.ones((2, 1)), X_new] # add x0 = 1 to each instance
>>> y_predict = X_new_b.dot(theta_best)
>>> y_predict
array([[4.21509616],
       [9.75532293]])
```

Hãy vẽ đồ thị biểu diễn các dự đoán của mô hình này (Hình 4.2):

```
plt.plot(X_new, y_predict, "r-")
plt.plot(X, y, "b.")
plt.axis([0, 2, 0, 15])
plt.show()
```



Hình 4.2. Các dự đoán của mô hình Hồi quy Tuyến tính

Với Scikit-Learn, việc áp dụng Hồi quy Tuyến tính tương đối đơn giản:[2]

```
>>> from sklearn.linear_model import LinearRegression
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([4.21509616]), array([[2.77011339]]))
>>> lin_reg.predict(X_new)
array([[4.21509616],
       [9.75532293]])
```

[2] Chú ý rằng Scikit-Learn tách rời hệ số điều chỉnh (`intercept_`) và trọng số đặc trưng (`coef_`).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lớp `LinearRegression` được dựa trên hàm `scipy.linalg.lstsq()` (viết tắt cho "least squares" – "bình phương nhỏ nhất") mà ta có thể gọi trực tiếp:

```
>>> theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y, rcond=1e-6)
>>> theta_best_svd
array([[4.21509616],
       [2.77011339]])
```

Hàm này tính $\hat{\boldsymbol{\theta}} = \mathbf{X}^{+}\mathbf{y}$, trong đó $\mathbf{X}^{+}$ là *ma trận giả nghịch đảo* (*pseudoinverse*) của $\mathbf{X}$ (cụ thể là nghịch đảo Moore-Penrose). Ta có thể sử dụng `np.linalg.pinv()` để tính trực tiếp ma trận giả nghịch đảo:

```
>>> np.linalg.pinv(X_b).dot(y)
array([[4.21509616],
       [2.77011339]])
```

Bản thân ma trận nghịch đảo tổng quát này có thể được tính bằng một kỹ thuật phân rã ma trận tiêu chuẩn là *Phân tích Giá trị Suy biến* (*Singular Value Decomposition* – SVD). Kỹ thuật này có thể phân rã ma trận chứa tập huấn luyện $\mathbf{X}$ thành tích của ba ma trận $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\top}$ (tham khảo `numpy.linalg.svd()`). Ma trận giả nghịch đảo được tính bằng công thức $\mathbf{X}^{+} = \mathbf{V}\boldsymbol{\Sigma}^{+}\mathbf{U}^{\top}$. Để tính ma trận $\boldsymbol{\Sigma}^{+}$, thuật toán lấy $\boldsymbol{\Sigma}$ và thay giá trị 0 vào tất cả các giá trị nhỏ hơn một ngưỡng (rất nhỏ) nhất định, sau đó thay thế tất cả các giá trị khác 0 bằng nghịch đảo của chúng và cuối cùng chuyển vị ma trận thu được. Hướng tiếp cận này hiệu quả hơn so với việc giải Phương trình Pháp tuyến, đồng thời xử lý tốt hơn các trường hợp đặc biệt. Thật vậy, ta không giải được Phương trình Pháp tuyến nếu $\mathbf{X}^{\top}\mathbf{X}$ không có nghịch đảo (tức là ma trận suy biến – singular matrix), chẳng hạn như khi $m < n$ hoặc khi một số đặc trưng là dư thừa, trong khi ma trận giả nghịch đảo thì luôn xác định được.

Độ phức tạp Tính toán

Phương trình Pháp tuyến được dùng để tính nghịch đảo của $\mathbf{X}^{\top}\mathbf{X}$, là một ma trận $(n+1)\times(n+1)$ (trong đó n là số lượng đặc trưng). Độ phức tạp tính toán của việc nghịch đảo ma trận thường nằm trong khoảng từ $O(n^{2.4})$ tới $O(n^3)$, phụ thuộc vào cách lập trình. Nói cách khác, nếu ta tăng gấp đôi số lượng đặc trưng, thời gian tính toán sẽ tăng từ $2^{2.4} = 5.3$ tới $2^3 = 8$ lần.

Hướng tiếp cận SVD được sử dụng trong lớp `LinearRegression` của Scikit-Learn có độ phức tạp khoảng $O(n^2)$. Như vậy, nếu ta tăng gấp đôi số đặc trưng, thời gian tính toán sẽ tăng khoảng 4 lần.

Lưu ý

Cả Phương trình Pháp tuyến và SVD đều trở nên rất chậm khi số lượng các đặc trưng trở nên rất lớn (ví dụ như 100,000). Mặt tích cực là độ phức tạp của cả hai đều chỉ tăng tuyến tính theo số lượng mẫu trong tập huấn luyện ($O(m)$), vì vậy chúng có thể xử lý các tập huấn luyện lớn một cách hiệu quả, miễn là có đủ bộ nhớ.

Ngoài ra, một khi ta đã huấn luyện xong mô hình Hồi quy Tuyến tính (sử dụng Phương trình Pháp tuyến hoặc bất kỳ thuật toán nào khác), việc dự đoán diễn ra rất nhanh: độ phức tạp tính toán chỉ tăng tuyến tính với số lượng mẫu muốn dự đoán cũng như số lượng đặc trưng. Nói

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

cách khác, việc dự đoán trên gấp đôi số mẫu (hoặc gấp đôi số đặc trưng) sẽ chỉ tốn xấp xỉ gấp đôi thời gian.

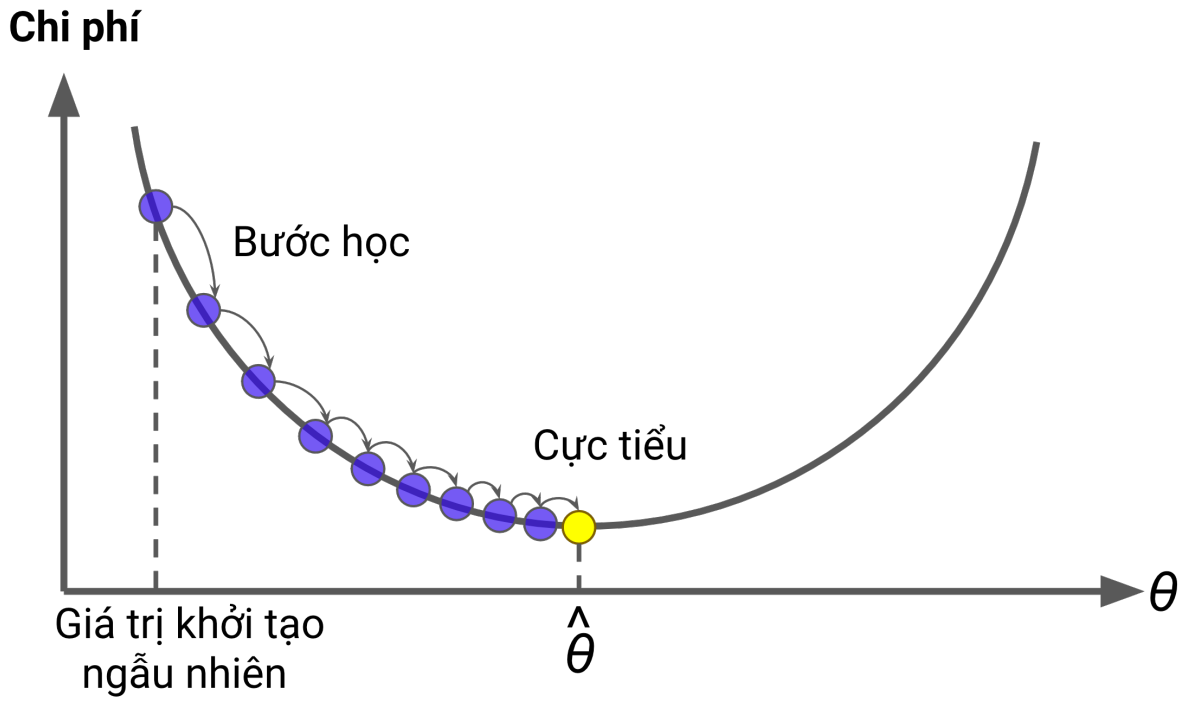
Bây giờ, chúng ta sẽ xem xét một phương pháp khác hẳn để huấn luyện mô hình Hồi quy Tuyến tính, phù hợp hơn trong trường hợp số lượng đặc trưng rất lớn hoặc có quá nhiều mẫu để có thể lưu vừa bộ nhớ.

Hạ Gradient

Hạ Gradient là một thuật toán tối ưu tổng quát, có khả năng tìm nghiệm tối ưu cho rất nhiều dạng bài toán. Ý tưởng chung của Hạ Gradient là liên tục điều chỉnh các tham số để cực tiểu hóa một hàm chi phí.

Giả sử, ta đang bị lạc trên núi trong sương mù dày đặc và chỉ có thể cảm nhận được độ dốc của mặt đất dưới chân. Một chiến lược tối ưu để xuống chân núi một cách nhanh chóng là đi xuống theo hướng dốc nhất. Đây chính xác là những gì mà Hạ Gradient thực hiện: nó tính gradient cục bộ của hàm chi phí theo vector tham số $\boldsymbol{\theta}$, rồi đi theo hướng ngược với gradient đó. Khi gradient bằng 0, ta đã tới một điểm cực tiểu!

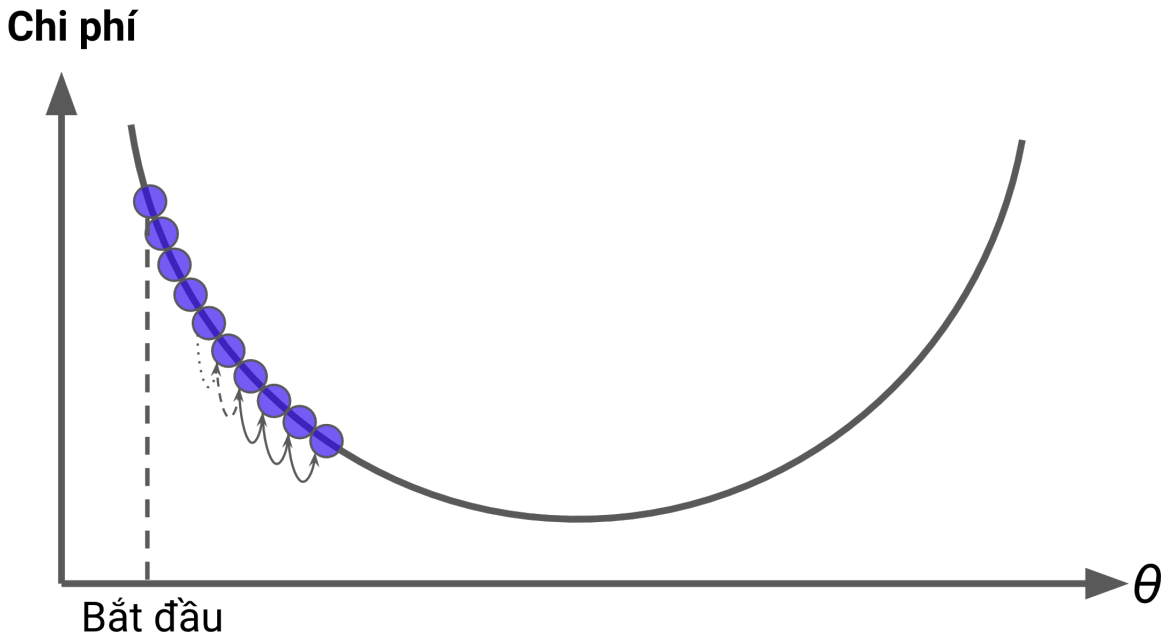
Cụ thể hơn, ta bắt đầu bằng việc gán các giá trị ngẫu nhiên cho $\boldsymbol{\theta}$ (đây được gọi là *khởi tạo ngẫu nhiên – random initialization*). Sau đó các giá trị này dần được cải thiện bằng cách đi từng bước nhỏ, mỗi bước cố gắng làm giảm hàm chi phí (như MSE), cho đến khi thuật toán *hội tụ* tại điểm cực tiểu (tham khảo Hình 4.3).



Hình 4.3. Minh hoạ thuật toán Hạ Gradient: các tham số mô hình được khởi tạo ngẫu nhiên, và được cập nhật liên tục để cực tiểu hóa hàm chi phí. Kích thước mỗi bước cập nhật tỉ lệ thuận với độ dốc của hàm chi phí, nên thuật toán đi càng chậm khi tham số càng ở gần điểm cực tiểu

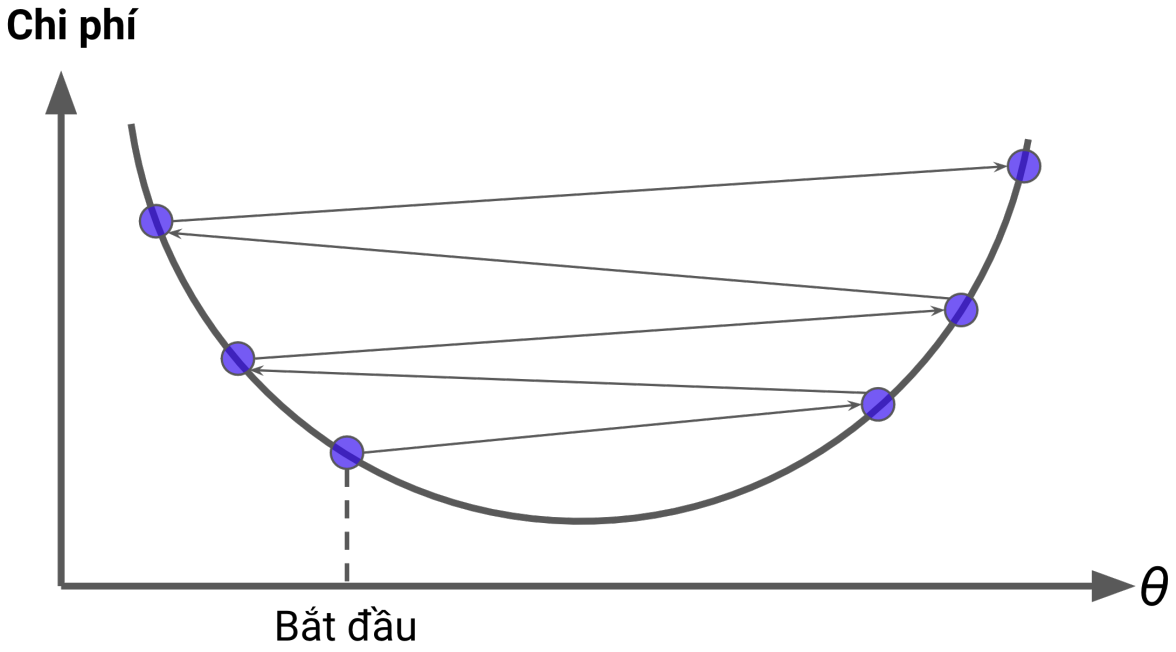
Một tham số quan trọng của Hạ Gradient là kích thước của bước cập nhật, được quyết định bởi siêu tham số *tốc độ học* (*learning rate*). Nếu tốc độ học quá nhỏ, thuật toán sẽ cần nhiều vòng lặp để hội tụ nên mất nhiều thời gian (tham khảo Hình 4.4).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.4. Tốc độ học quá nhỏ

Ngược lại, nếu tốc độ học quá lớn, ta có thể sẽ nhảy vọt qua điểm tối ưu đến sườn dốc bên kia, thậm chí có thể cao hơn vị trí trước đó. Điều này có thể khiến thuật toán phân kỳ với các giá trị ngày càng lớn và không thể tìm được lời giải tốt (tham khảo Hình 4.5).

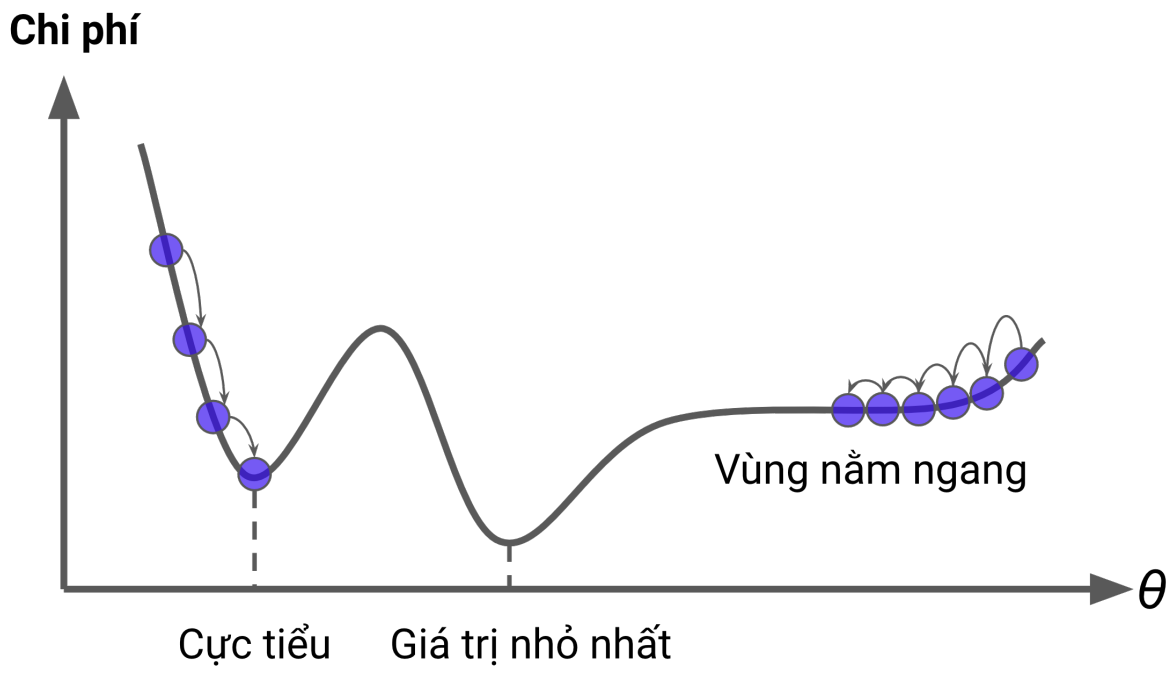


Hình 4.5. Tốc độ học quá lớn

Cuối cùng, không phải tất cả hàm chi phí đều có hình dạng giống như một cái bát đều đặn. Chúng có thể có hình hố, chóp núi, cao nguyên, và vô số loại địa hình kì dị, khiến việc hội tụ về điểm cực tiểu trở nên khó khăn. Hình 4.6 minh hoạ hai thách thức lớn với Hạ Gradient. Nếu

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

khởi tạo ngẫu nhiên về bên trái, thuật toán sẽ hội tụ về một *điểm cực tiểu* (*local minimum*), không tốt bằng *giá trị nhỏ nhất* (*global minimum*). Nếu khởi tạo về bên phải, thuật toán sẽ mất rất nhiều thời gian để vượt qua vùng nằm ngang. Và nếu dừng quá sớm, ta không thể đạt được giá trị nhỏ nhất.



Hình 4.6. Các thách thức của Hạ Gradient

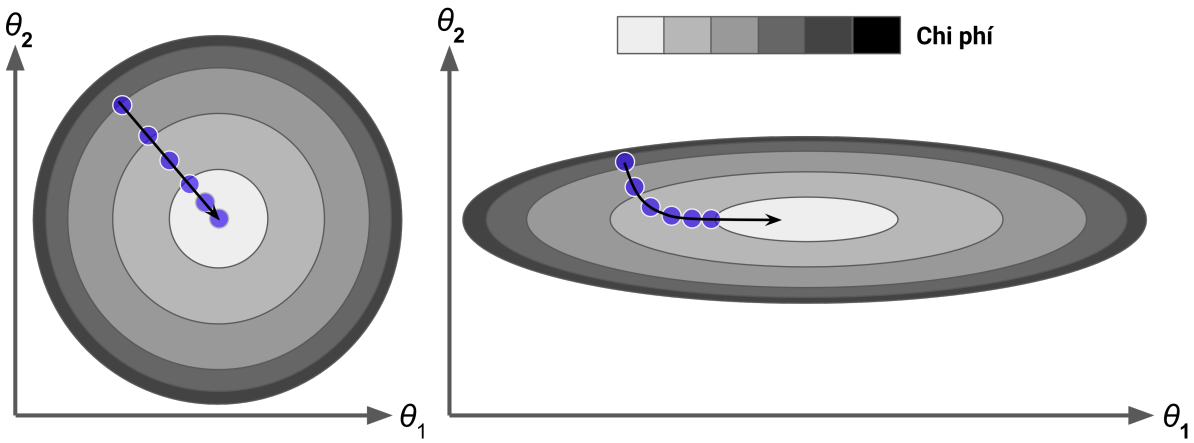
May thay, hàm chi phí MSE của mô hình Hồi quy Tuyến tính là một *hàm lồi* (*convex function*), nghĩa là đoạn thẳng nối hai điểm bất kỳ trên đường cong không bao giờ cắt đường cong đó. Điều này đồng nghĩa với việc đồ thị không có điểm cực tiểu, mà chỉ có một giá trị nhỏ nhất. Hàm lồi cũng là một hàm liên tục với độ dốc không bao giờ thay đổi đột ngột.[3] Hai điều này dẫn đến một hệ quả mong muốn: Hạ Gradient cho Hồi quy Tuyến tính chắc chắn sẽ tiệm cận rất gần tới giá trị nhỏ nhất (nếu chờ đủ lâu và tốc độ học không quá cao).

Thực tế, hàm chi phí này có dạng hình cái bát, nhưng có thể trở nên thuôn dài nếu các đặc trưng có khoảng giá trị rất khác nhau. Hình 4.7 minh hoạ Hạ Gradient trên một tập huấn luyện mà đặc trưng 1 và 2 có cùng khoảng giá trị (hình bên trái), và trên một tập khác mà đặc trưng 1 có giá trị nhỏ hơn nhiều so với đặc trưng 2 (hình bên phải).[4]

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

[3] Theo ngôn ngữ kỹ thuật, thì đạo hàm của hàm này *liên tục Lipschitz*.

[4] Vì khoảng giá trị đặc trưng 1 nhỏ hơn, cần thay đổi θ_1 nhiều hơn để gây ảnh hưởng tới hàm chi phí, do đó đồ thị thuôn dài theo trục θ_1.



Hình 4.7. Hạ Gradient với có (trái) và không có (phải) co giãn đặc trưng

Có thể thấy, trong hình bên trái, Hạ Gradient nhanh chóng hội tụ về điểm tối ưu. Trong khi ở hình bên phải, ban đầu thuật toán đi theo hướng gần như vuông góc với hướng từ gốc toạ độ đến điểm tối ưu, rồi đi một đoạn dài chỉ theo trục θ_1. Nó cuối cùng cũng sẽ đạt tới cực tiểu, nhưng sẽ mất nhiều thời gian.

Lưu ý

Khi sử dụng Hạ Gradient, bạn nên đảm bảo tất cả đặc trưng có khoảng giá trị gần giống nhau (bằng cách dùng lớp `StandardScaler` của Scikit-Learn), nếu không thuật toán sẽ mất rất nhiều thời gian để hội tụ.

Biểu đồ này cũng cho thấy rằng, việc huấn luyện mô hình thực chất là việc tìm kiếm tổ hợp các tham số tốt nhất để cực tiểu hóa hàm chi phí (trên tập huấn luyện). Đây là việc tìm kiếm trong *không gian tham số* (*parameter space*) của mô hình: mô hình càng có nhiều tham số, không gian này càng có nhiều chiều, và việc tìm kiếm càng khó khăn: tìm điểm tối ưu trong không gian 300 chiều khó hơn nhiều so với không gian 3 chiều. May thay, vì hàm chi phí là lồi trong trường hợp của Hồi quy Tuyến tính, ta không cần lo về việc nghiệm tìm được là cực tiểu địa phương.

Hạ Gradient theo Batch

Để lập trình Hạ Gradient, ta cần tính gradient của hàm chi phí theo từng tham số mô hình θ_j. Nói cách khác, ta cần biết hàm chi phí sẽ thay đổi bao nhiêu khi θ_j thay đổi một lượng nhỏ. Khái niệm này được gọi là *đạo hàm riêng* (*partial derivative*). Việc tính gradient theo từng tham số mô hình giống như tự hỏi "Sườn núi dưới chân dốc như thế nào khi hướng về phía đông?" rồi lặp lại câu hỏi khi hướng về phía bắc (cũng như cho tất cả các chiều không gian khác, nếu vũ trụ có nhiều hơn 3 chiều). Phương trình 4.5 tính đạo hàm riêng của hàm chi phí theo tham số θ_j, kí hiệu là $\frac{\partial}{\partial \theta_j}\text{MSE}(\boldsymbol{\theta})$.

$$\frac{\partial}{\partial \theta_j}\text{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)})\, x_j^{(i)}$$

Phương trình 4.5. Đạo hàm riêng của hàm chi phí

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thay vì tính từng đạo hàm riêng theo mỗi tham số, ta có thể sử dụng Phương trình 4.6 để tính toán các đại lượng này cùng lúc. Vector gradient, kí hiệu $\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$, chứa tất cả đạo hàm riêng của hàm chi phí (theo từng tham số mô hình).

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Phương trình 4.6. Vector gradient của hàm chi phí

Lưu ý

Chú ý rằng trong công thức này, việc tính toán được thực hiện trên toàn bộ tập huấn luyện $\mathbf{X}$ tại mỗi bước Hạ Gradient! Vậy nên, thuật toán mới được gọi là *Hạ Gradient theo Batch* (*Batch Gradient Descent*): sử dụng toàn bộ batch dữ liệu huấn luyện tại tất cả các bước (thành thật mà nói, *Hạ Gradient Đầy đủ – Full Gradient Descent* có thể là một cái tên tốt hơn). Do đó, thuật toán hoạt động cực kỳ chậm trên các tập huấn luyện rất lớn (nhưng sớm thôi, ta sẽ biết một thuật toán Hạ Gradient khác nhanh hơn rất nhiều). Tuy nhiên, Hạ Gradient mở rộng rất tốt khi số lượng đặc trưng tăng lên. Ví dụ, việc huấn luyện mô hình Hồi quy Tuyến tính bằng Hạ Gradient khi có hàng trăm nghìn đặc trưng nhanh hơn nhiều so với khi sử dụng Phương trình Pháp tuyến hoặc phân rã SVD.

Khi đã có vector gradient (chỉ hướng đi lên), ta chỉ cần đi về hướng ngược lại để đi xuống, nghĩa là trừ đi một lượng $\nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$ từ $\boldsymbol{\theta}$. Để xác định kích thước bước cập nhật, ta nhân tốc độ học η[5] với vector gradient (Phương trình 4.7).

$$\boldsymbol{\theta}^{(\text{bước tiếp theo})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

Phương trình 4.7. Bước cập nhật của Hạ Gradient

Hãy cùng nhìn qua đoạn mã minh họa cho thuật toán này:

```
eta = 0.1  # learning rate
n_iterations = 1000
m = 100

theta = np.random.randn(2,1)  # random initialization

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```

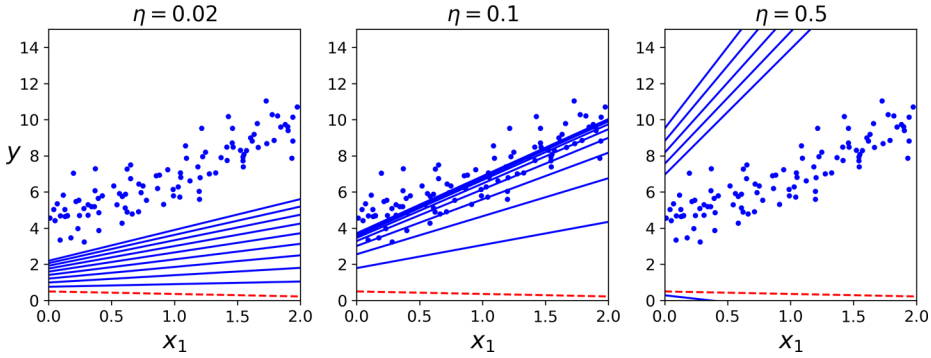
Đoạn mã cũng không quá phức tạp. Giờ ta hãy xem thử đầu ra `theta`:

[5] Eta (η) là ký tự thứ 7 trong bảng chữ cái Hy Lạp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> theta
array([[4.21509616],
       [2.77011339]])
```

Đây chính xác là những gì Phương trình Pháp tuyến tìm được! Hạ Gradient hoạt động như mong đợi. Nhưng sẽ ra sao nếu ta sử dụng tốc độ học `eta` khác? Hình 4.8 minh hoạ 10 bước Hạ Gradient đầu tiên của 3 tốc độ học khác nhau (đường nét đứt biểu diễn điểm bắt đầu).



Hình 4.8. Hạ Gradient với nhiều tốc độ học

Trong hình bên trái, tốc độ học quá nhỏ: thuật toán cuối cùng cũng sẽ hội tụ, nhưng mất rất nhiều thời gian. Trong hình giữa, tốc độ học khá phù hợp: chỉ cần một vài vòng lặp là đã hội tụ. Trong hình bên phải, tốc độ học quá cao: thuật toán phân kỳ, nhảy lung tung và đi xa dần khỏi nghiệm tối ưu.

Để tìm tốc độ học phù hợp, ta có thể dùng tìm kiếm dạng lưới (tham khảo Chương 2). Tuy nhiên, ta có thể giới hạn số vòng lặp để tìm kiếm dạng lưới loại bỏ các mô hình tốn quá nhiều thời gian để hội tụ.

Có thể bạn sẽ băn khoăn về việc chọn số lần lặp. Nếu quá ít, kết quả sẽ còn xa nghiệm tối ưu khi thuật toán kết thúc. Nhưng nếu quá nhiều, ta sẽ lãng phí thời gian khi các tham số mô hình không thể thay đổi thêm nữa. Một giải pháp đơn giản là đặt số lượng vòng lặp lớn nhưng sẽ dừng thuật toán khi vector gradient đủ nhỏ, tức là khi chuẩn của nó bé hơn một đại lượng rất nhỏ ϵ (gọi là *dung sai – tolerance*), vì lúc này này Hạ Gradient đã (gần như) đạt được nghiệm tối ưu.

Tốc độ Hội tụ

Khi hàm chi phí là lồi và độ dốc không thay đổi đột ngột (như trường hợp của MSE), Hạ Gradient theo Batch với tốc độ học cố định cuối cùng cũng sẽ hội tụ về nghiệm tối ưu, nhưng có thể sẽ mất thời gian: có thể tốn $O(1/\epsilon)$ vòng lặp để đạt nghiệm tối ưu nằm trong khoảng ϵ, tuỳ thuộc vào hình dạng của hàm chi phí. Nếu chia dung sai cho 10 để đạt nghiệm chính xác hơn, thuật toán có thể sẽ chạy lâu hơn gấp 10 lần.

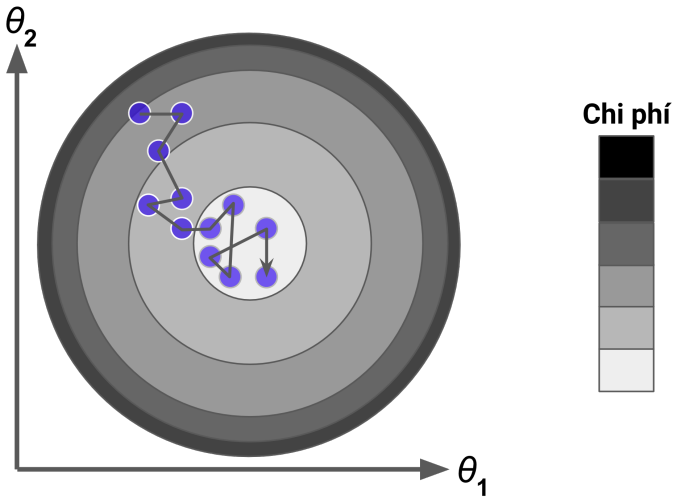
Hạ Gradient Ngẫu nhiên

Vấn đề chính của Hạ Gradient theo Batch nằm ở việc sử dụng toàn bộ tập huấn luyện để tính gradient tại tất cả các bước, và điều này làm giảm tốc độ thuật toán khi tập huấn luyện lớn. Mặt khác, *Hạ Gradient Ngẫu nhiên* (*Stochastic Gradient Descent*) lấy một mẫu ngẫu nhiên trong tập huấn luyện tại mỗi bước và tính gradient chỉ dựa trên mẫu đó. Rõ ràng, làm việc với một mẫu dữ liệu giúp thuật toán chạy nhanh hơn vì chỉ cần xử lý rất ít dữ liệu tại mỗi vòng lặp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Điều này cũng giúp việc huấn luyện trên các tập dữ liệu lớn trở nên khả thi, vì chỉ cần một mẫu dữ liệu trong bộ nhớ tại mỗi vòng lặp (SGD có thể được lập trình dưới dạng một thuật toán học ngoài bộ nhớ chính; tham khảo Chương 1).

Tuy nhiên, do tính chất ngẫu nhiên, thuật toán này không phổ biến bằng Hạ Gradient theo Batch: hàm chi phí thay vì giảm dần cho tới khi đạt cực tiểu, nó sẽ dao động lên xuống, dù nhìn chung có xu hướng giảm. Qua thời gian, hàm này sẽ dần tiến gần về cực tiểu, nhưng nó sẽ tiếp tục dao động xung quanh chứ không hoàn toàn hội tụ (tham khảo Hình 4.9). Kết quả là khi dừng thuật toán, giá trị tham số cuối cùng cho dù tốt, lại không phải là giá trị tối ưu.



Hình 4.9. Trong Hạ Gradient Ngẫu nhiên, mỗi bước huấn luyện sẽ nhanh hơn nhưng độ ngẫu nhiên cũng lớn hơn nhiều so với Hạ Gradient theo Batch

Khi hàm chi phí có hình dạng kì dị (như trong Hình 4.6), SGD nhờ đặc tính ngẫu nhiên có thể thoát ra khỏi cực tiểu địa phương, nên có khả năng đạt đến giá trị nhỏ nhất cao hơn Hạ Gradient theo Batch.

Như vậy, sự ngẫu nhiên dù có thể giúp thoát khỏi điểm cực tiểu địa phương, nhưng cũng đồng nghĩa với việc không thể đạt được giá trị nhỏ nhất. Một giải pháp cho vấn đề này là từ từ giảm tốc độ học. Ta sẽ bắt đầu với các bước cập nhật lớn (giúp tăng tốc nhanh và thoát khỏi các cực tiểu), rồi giảm dần để thuật toán đạt đến giá trị nhỏ nhất một cách ổn định. Quá trình này giống với *mô phỏng luyện kim* (*simulated annealing*), một thuật toán lấy cảm hứng từ quá trình ủ trong ngành luyện kim, khi kim loại nấu chảy được làm nguội từ từ. Hàm quyết định tốc độ học tại mỗi vòng lặp được gọi là *định thời học* (*learning schedule*). Nếu tốc độ học giảm quá nhanh, thuật toán có thể bị kẹt tại điểm cực tiểu, hoặc thậm chí dừng tiếp cận nghiệm tối ưu giữa chừng. Nếu tốc độ học giảm quá chậm, kết quả có thể dao động quanh điểm tối ưu rất lâu và có thể không cho nghiệm tốt nếu dừng huấn luyện quá sớm.

Đoạn mã dưới đây lập trình Hạ Gradient Ngẫu nhiên với một định thời học đơn giản:

```
n_epochs = 50
t0, t1 = 5, 50  # learning schedule hyperparameters

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)  # random initialization
```

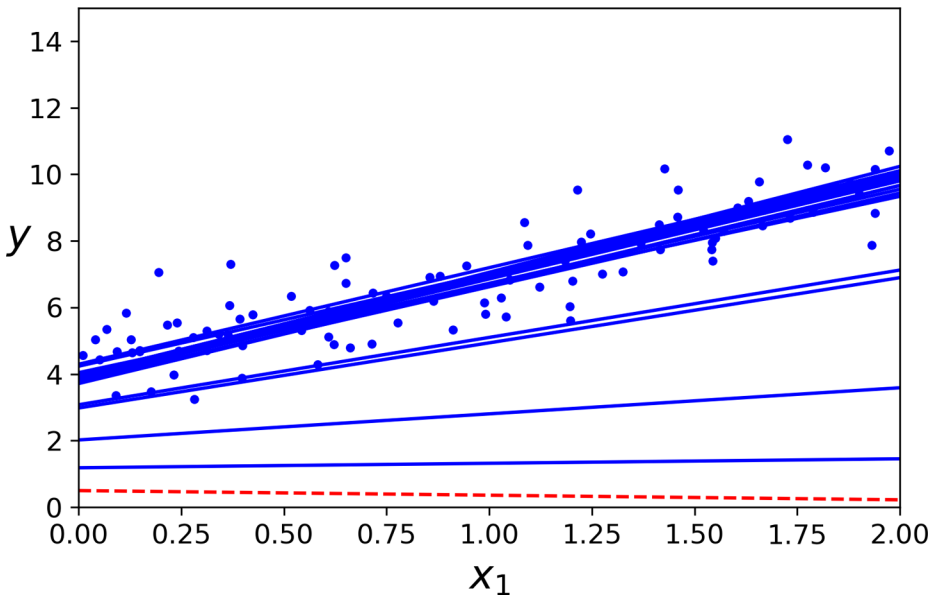

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
for epoch in range(n_epochs):
    for i in range(m):
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
```

Theo quy ước, ta gọi một *epoch* là một vòng lặp gồm m lần lặp. Trong khi ở ví dụ trước, Hạ Gradient theo Batch lặp 1,000 lần qua toàn bộ tập huấn luyện, trong đoạn mã này, Hạ Gradient Ngẫu nhiên chỉ đi qua tập huấn luyện 50 lần nhưng đã đạt được một kết quả khá tốt.

```
>>> theta
array([[4.21076011],
       [2.74856079]])
```

Hình 4.10 biểu thị 20 bước đầu tiên của quá trình huấn luyện (chú ý sự thay đổi bất thường của các bước).



Hình 4.10. 20 bước đầu tiên của Hạ Gradient Ngẫu nhiên

Vì các mẫu được chọn một cách ngẫu nhiên, một vài mẫu có thể được chọn nhiều lần trong một epoch, trong khi có thể có các mẫu không hề được lựa chọn. Nếu muốn chắc chắn rằng thuật toán sẽ đi qua tất cả các mẫu với mỗi epoch, ta có thể xáo trộn tập huấn luyện (đảm bảo rằng các đặc trưng đầu vào luôn đi với nhãn tương ứng), đi qua lần lượt từng mẫu của tập huấn luyện, rồi xáo trộn lại một lần nữa, và cứ thế. Tuy nhiên, nhìn chung cách này hội tụ chậm hơn.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lưu ý

Khi sử dụng Hạ Gradient Ngẫu nhiên, các mẫu huấn luyện cần phải là các mẫu độc lập và từ các phân phối giống nhau (*independent and identically distributed* – IID) để đảm bảo rằng các tham số, về trung bình, được kéo về giá trị nhỏ nhất. Một cách đơn giản để đảm bảo điều này, đó là xáo trộn các mẫu khi huấn luyện (ví dụ, bằng cách chọn một mẫu ngẫu nhiên, hoặc xáo trộn tập huấn luyện khi bắt đầu mỗi epoch). Nếu ta không xáo trộn các mẫu – giả sử các mẫu được sắp xếp theo nhãn – thì SGD sẽ bắt đầu bằng cách tối ưu cho một nhãn trước, rồi đến nhãn tiếp theo, và cứ thế, khiến nó không thể tới gần nghiệm tối ưu.

Để áp dụng Hồi quy Tuyến tính bằng SGD trong Scikit-Learn, ta có thể dùng lớp `SGDRegressor`, được mặc định sẽ tối ưu hóa hàm chi phí bình phương sai số. Đoạn mã bên dưới sẽ chạy tới khi đạt được 1,000 epoch hoặc khi mất mát hạ xuống thấp hơn 0.001 tại một epoch nào đó (`max_iter=1000`, `tol=1e-3`). Ta bắt đầu huấn luyện với tốc độ học 0.1 (`eta0=0.1`), sử dụng bộ định thời học mặc định (khác với lần trước) và không sử dụng điều chuẩn (`penalty=None`; chúng tôi sẽ đề cập chi tiết hơn về kỹ thuật này sau):

```
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, penalty=None, eta0=0.1)
sgd_reg.fit(X, y.ravel())
```

Một lần nữa, ta thu được kết quả khá gần với kết quả trả về bởi Phương trình Pháp tuyến:

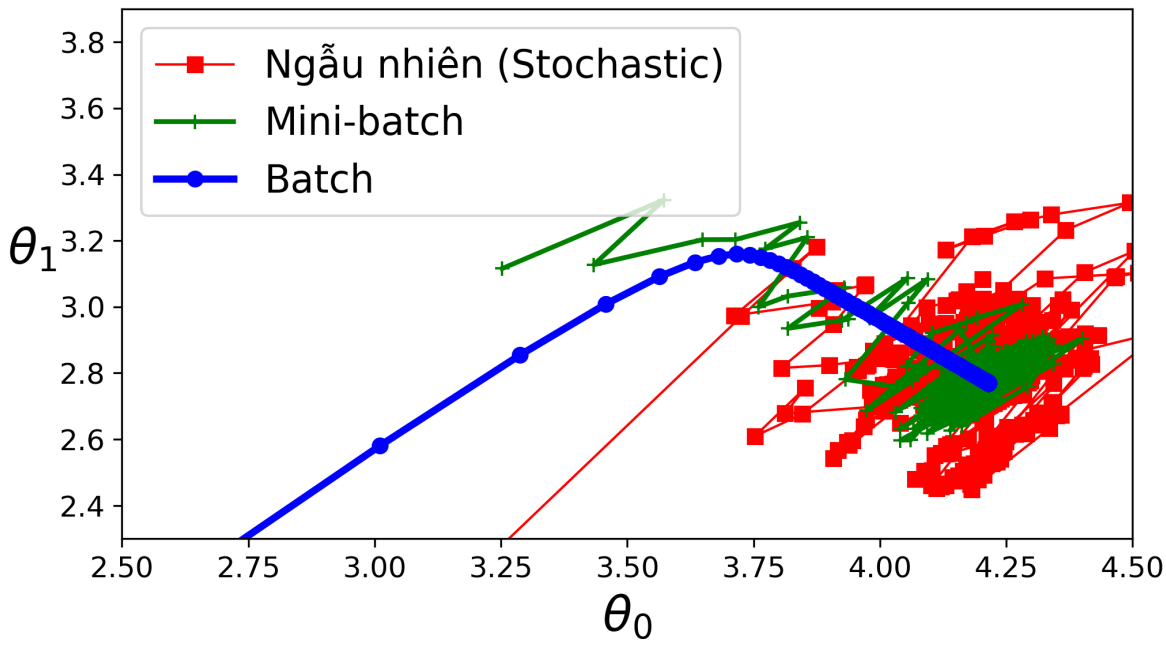
```
>>> sgd_reg.intercept_, sgd_reg.coef_
(array([4.24365286]), array([2.8250878]))
```

Hạ Gradient theo Mini-batch

Thuật toán Hạ Gradient cuối cùng mà ta sẽ tìm hiểu là *Hạ Gradient theo Mini-batch* (*Mini-batch Gradient Descent*). Thuật toán này khá dễ hiểu khi đã nắm vững Hạ Gradient theo Batch và Hạ Gradient Ngẫu nhiên: tại mỗi bước, thay vì tính gradient dựa trên toàn bộ tập huấn luyện (như trong Hạ Gradient theo Batch) hoặc dựa trên chỉ một mẫu (như trong Hạ Gradient Ngẫu nhiên), Hạ Gradient theo Mini-batch thực hiện tính gradient dựa trên một tập nhỏ các mẫu ngẫu nhiên được gọi là các *mini-batch*. So với Hạ Gradient Ngẫu nhiên, phương pháp này có thể tăng hiệu suất các phép tính ma trận thông qua tối ưu phần cứng, nhất là khi sử dụng GPU.

Thuật toán này hoạt động trong miền tham số ổn định hơn so với Hạ Gradient Ngẫu nhiên, nhất là khi các mini-batch có kích thước khá lớn. Do đó, Hạ Gradient theo Mini-batch tiến gần tới giá trị cực tiểu hơn so với Hạ Gradient Ngẫu nhiên – nhưng cũng khó thoát khỏi các cực tiểu địa phương hơn (trong trường hợp bài toán có nhiều hơn một cực tiểu, khác với bài toán Hồi quy Tuyến tính). Hình 4.11 phác họa đường đi của ba thuật toán Hạ Gradient trong miền tham số khi huấn luyện. Tất cả các thuật toán đều kết thúc gần với điểm có giá trị nhỏ nhất, tuy nhiên Hạ Gradient theo Batch thực sự dừng lại tại chính điểm đó trong khi Hạ Gradient Ngẫu nhiên và Hạ Gradient theo Mini-batch tiếp tục dao động xung quanh. Dù vậy, đừng quên rằng Hạ Gradient theo Batch cần rất nhiều thời gian để thực hiện một bước cập nhật, và Hạ Gradient Ngẫu nhiên cũng như Hạ Gradient theo Mini-batch vẫn có thể đạt tới điểm tối ưu đó nếu ta sử dụng một bộ định thời học thích hợp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.11. Đường đi của các thuật toán Hạ Gradient trong miền tham số

Hãy so sánh các thuật toán cho bài toán Hồi quy Tuyến tính mà ta đã tìm hiểu[6] (nhắc lại rằng m là số lượng mẫu huấn luyện và n là số lượng đặc trưng); tham khảo Bảng 4.1.

Bảng 4.1. So sánh các thuật toán cho bài toán Hồi quy Tuyến tính

Thuật toán	m lớn	Hỗ trợ học ngoài bộ nhớ chính	n lớn	Số siêu tham số	Cần co giãn đầu vào	Scikit-Learn
Phương trình Pháp tuyến	Nhanh	Không	Chậm	0	Không	N/A
SVD	Nhanh	Không	Chậm	0	Không	`LinearRegression`
Hạ Gradient theo Batch	Chậm	Không	Nhanh	2	Có	`SGDRegressor`
Hạ Gradient Ngẫu nhiên	Nhanh	Có	Nhanh	≥ 2	Có	`SGDRegressor`
Hạ Gradient theo Mini-batch	Nhanh	Có	Nhanh	≥ 2	Có	`SGDRegressor`

Ghi chú

Các mô hình dường như không có sự khác biệt sau huấn luyện: tất cả các thuật toán đều kết thúc với những mô hình khá giống nhau và đưa ra các dự đoán tương tự nhau.

[6] Trong khi Phương trình Pháp tuyến chỉ áp dụng cho Hồi quy Tuyến tính, Hạ Gradient có thể dùng để huấn luyện nhiều mô hình khác, như ta sẽ thấy sau này

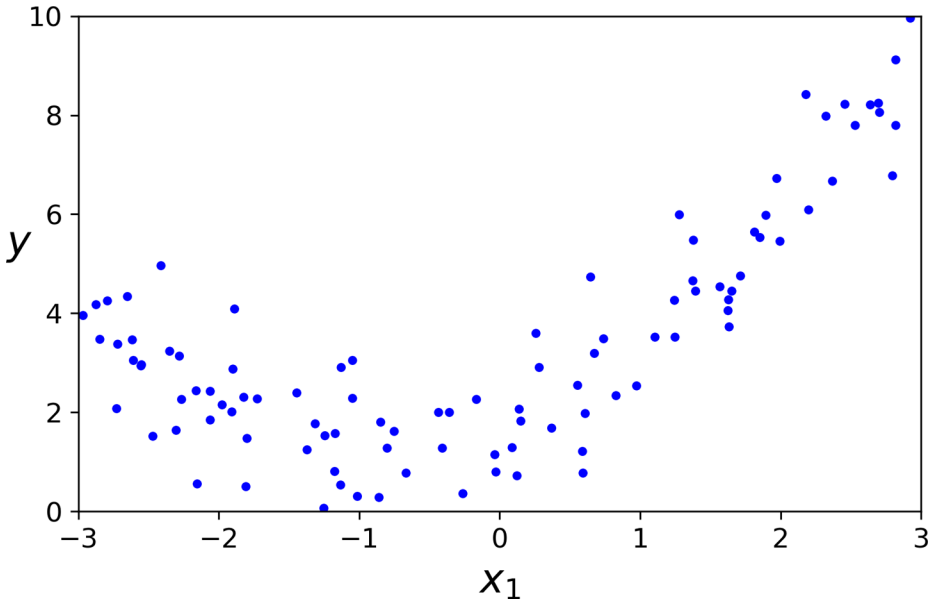
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hồi quy Đa thức

Nếu dữ liệu của ta phức tạp hơn dạng tuyến tính thì sao? Bất ngờ thay, ta có thể sử dụng một mô hình tuyến tính để khớp vào dữ liệu phi tuyến. Một cách đơn giản để làm điều này là thêm các bậc lũy thừa của mỗi đặc trưng, coi chúng như những đặc trưng mới, sau đó huấn luyện một mô hình tuyến tính trên bộ đặc trưng mở rộng này. Kỹ thuật này được gọi là *Hồi quy Đa thức* (*Polynomial Regression*).

Hãy cùng xem thử một ví dụ. Đầu tiên, ta tạo một vài mẫu dữ liệu phi tuyến dựa trên một phương trình bậc hai[7] đơn giản (cộng thêm nhiễu; tham khảo Hình 4.12):

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



Hình 4.12. Tập dữ liệu phi tuyến có nhiễu

Rõ ràng, một đường thẳng sẽ không thể khớp tốt với tập dữ liệu này. Vậy nên hãy sử dụng lớp `PolynomialFeatures` của Scikit-Learn để biến đổi dữ liệu huấn luyện bằng cách thêm giá trị bình phương (đa thức bậc hai) của mỗi đặc trưng vào tập huấn luyện như một đặc trưng mới:

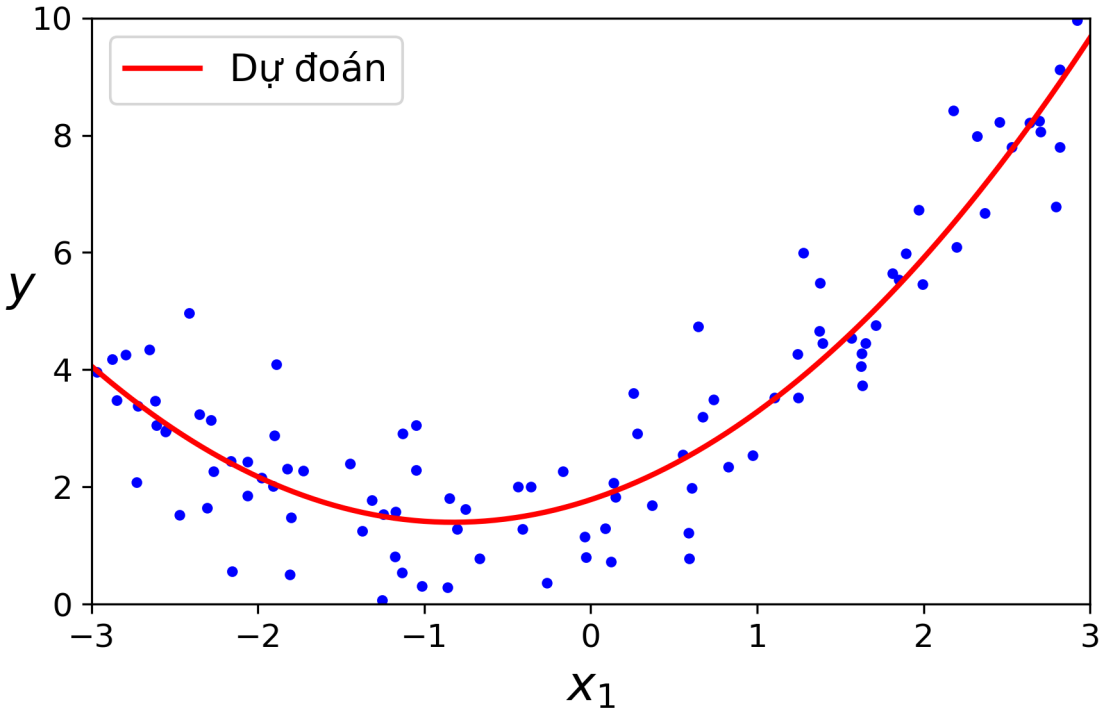
```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly_features = PolynomialFeatures(degree=2, include_bias=False)
>>> X_poly = poly_features.fit_transform(X)
>>> X[0]
array([-0.75275929])
>>> X_poly[0]
array([-0.75275929, 0.56664654])
```

`X_poly` hiện đang chứa cả đặc trưng ban đầu `X` lẫn bình phương của đặc trưng này. Giờ ta có thể khớp một mô hình `LinearRegression` vào tập huấn luyện mới này (Hình 4.13):

[7] Một phương trình bậc hai có dạng $y = ax^2 + bx + c$.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> lin_reg = LinearRegression()
>>> lin_reg.fit(X_poly, y)
>>> lin_reg.intercept_, lin_reg.coef_
(array([1.78134581]), array([[0.93366893, 0.56456263]]))
```



Hình 4.13. Dự đoán của mô hình Hồi quy Đa thức

Kết quả không quá tệ: mô hình ước đoán $\hat{y} = 0.56{x_1}^2 + 0.93x_1 + 1.78$ trong khi đó hàm gốc ban đầu là $y = 0.5{x_1}^2 + 1.0x_1 + 2.0 +$ `nhiễu Gauss`.

Lưu ý rằng, trong trường hợp nhiều đặc trưng, Hồi quy Đa thức có khả năng tìm ra mối quan hệ giữa các đặc trưng (điều mà mô hình Hồi quy Tuyến tính đơn giản không thể làm được). Khả năng này có được là nhờ `PolynomialFeatures` cũng thêm vào tất cả tổ hợp của các đặc trưng, miễn tổng số bậc của số hạng không lớn hơn bậc cho trước. Ví dụ, giả sử có hai đặc trưng a và b, `PolynomialFeatures` với `degree=3` sẽ không chỉ thêm các đặc trưng a^2, a^3, b^2, b^3, mà còn cả các tổ hợp $ab, a^2b,$ và ab^2.

Lưu ý

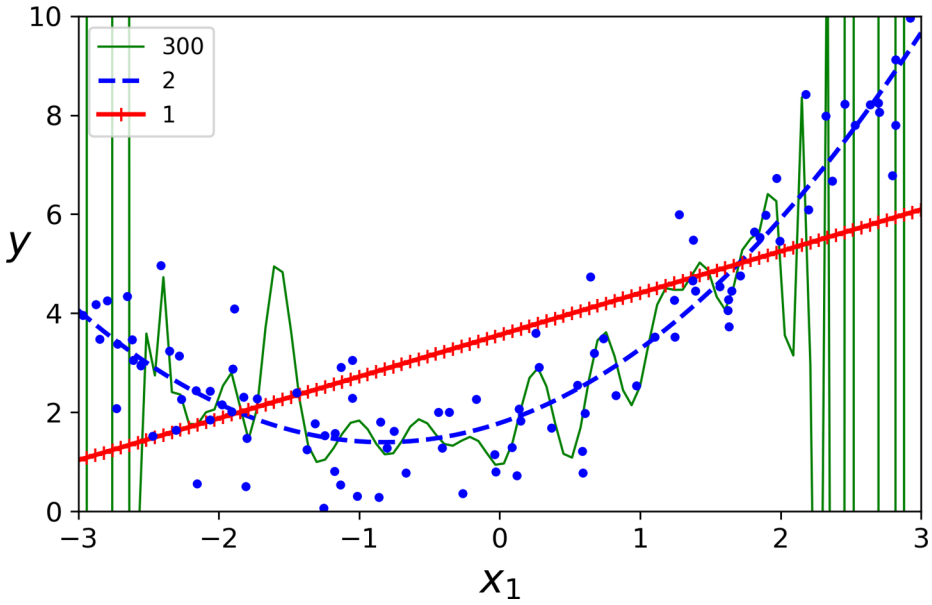
`PolynomialFeatures(degree=d)` biến đổi một mảng chứa n đặc trưng thành một mảng $\frac{(n+d)!}{d!n!}$ đặc trưng, với $n!$ là *giai thừa* của n, bằng $1 \times 2 \times 3 \times \cdots \times n$. Hãy cẩn thận với sự bùng nổ tổ hợp của số lượng các đặc trưng!

Đồ thị Quá trình học

Nếu ta sử dụng Hồi quy Đa thức bậc cao, khả năng cao mô hình sẽ khớp với dữ liệu huấn luyện hơn so với Hồi quy Tuyến tính đơn thuần. Ví dụ, Hình 4.14 áp dụng một mô hình đa thức bậc

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

300 trên tập huấn luyện được sinh trước đó, rồi so sánh với kết quả của một mô hình tuyến tính thuần và một mô hình bậc hai (đa thức bậc hai). Hãy chú ý cách mô hình đa thức bậc 300 đã biến đổi để gần với các mẫu huấn luyện nhất có thể.



Hình 4.14. Hồi quy Đa thức bậc cao

Mô hình Hồi quy Đa thức bậc cao này bị quá khớp cực kỳ nặng, trong khi đó mô hình tuyến tính lại dưới khớp. Trong trường hợp này, mô hình khái quát hóa tốt nhất là mô hình bậc hai, điều này là hợp lý vì dữ liệu của ta được sinh ra từ một hàm bậc hai. Tuy nhiên trong thực tế, ta không biết dữ liệu được sinh ra từ loại hàm số nào, vậy làm sao ta có thể quyết định độ phức tạp cho mô hình của mình? Làm sao ta có thể biết liệu mô hình có quá khớp hay dưới khớp dữ liệu?

Trong Chương 2, ta đã sử dụng kiểm định chéo để ước lượng khả năng khái quát hóa của mô hình. Nếu mô hình hoạt động tốt trên tập huấn luyện nhưng khái quát tệ theo kết quả kiểm định chéo, thì nó đang bị quá khớp. Nếu mô hình hoạt động tệ trên cả hai, thì nó đang bị dưới khớp. Đó là một cách để biết liệu mô hình có đang quá đơn giản hoặc quá phức tạp.

Một cách khác để biết được điều này đó là nhìn vào *đồ thị quá trình học* (*learning curves*): đây là đồ thị biểu thị chất lượng mô hình trên tập huấn luyện và tập kiểm định như một hàm của kích thước tập huấn luyện (hoặc của số lần lặp trong quá trình huấn luyện). Để tạo đồ thị này, ta huấn luyện mô hình nhiều lần trên các tập con có kích thước khác nhau của tập huấn luyện. Đoạn mã bên dưới định nghĩa một hàm nhận vào tập huấn luyện và một mô hình bất kỳ, rồi vẽ đồ thị quá trình học của mô hình đó:

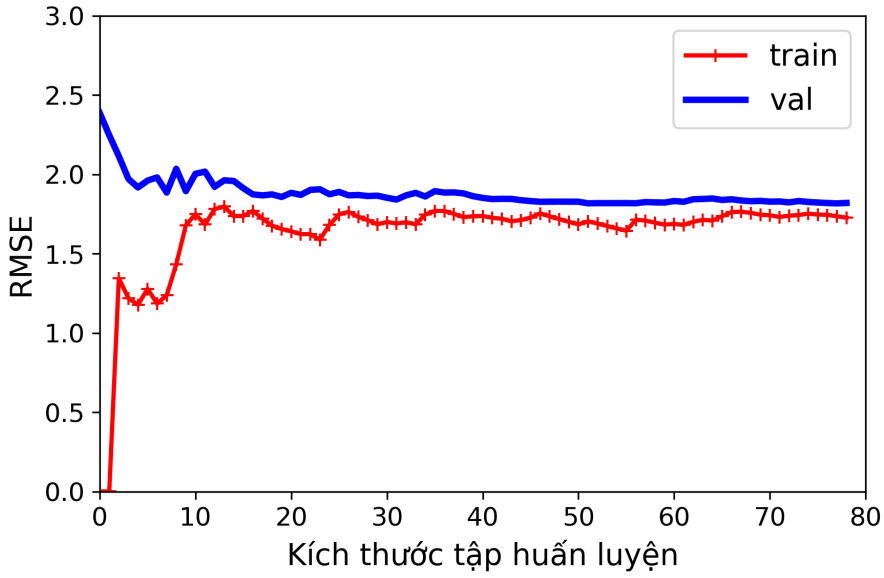
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train)):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
```


Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train[:m], y_train_predict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))
    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
```

Hãy cùng nhìn vào đồ thị quá trình học của một mô hình Hồi quy Tuyến tính đơn thuần (một đường thẳng; tham khảo Hình 4.15):

```
lin_reg = LinearRegression()
plot_learning_curves(lin_reg, X, y)
```



Hình 4.15. Đồ thị quá trình học

Hãy giải thích một chút về mô hình bị dưới khớp này. Đầu tiên, nhìn vào chất lượng trên tập huấn luyện: khi chỉ có một hoặc hai mẫu trong bộ huấn luyện, mô hình có thể khớp với chúng một cách hoàn hảo, đó là lý do đường cong bắt đầu tại 0. Nhưng khi các mẫu mới được thêm vào trong tập huấn luyện, mô hình không thể khớp được toàn bộ dữ liệu do nhiễu trong dữ liệu cũng như vì chúng không phải tuyến tính. Vì thế, sai số trên tập huấn luyện tăng dần cho đến khi không thể tăng được nữa, khi đó việc thêm các mẫu mới vào tập huấn luyện không thể làm sai số trung bình tốt hơn hay tệ hơn. Giờ hãy nhìn vào chất lượng trên tập kiểm định. Khi mô hình chỉ được huấn luyện trên lượng dữ liệu ít, nó không thể khái quát hóa tốt, khiến cho sai số kiểm định ban đầu khá lớn. Sau đó, khi mô hình được huấn luyện với nhiều mẫu huấn luyện hơn, nó bắt đầu học được các khuôn mẫu và vì thế sai số dần đi xuống. Tuy nhiên, vì một đường thẳng không thể làm tốt việc mô hình hóa loại dữ liệu này, sai số cuối cùng không thể giảm được nữa và kết thúc rất gần với đường cong phía bên dưới.

Đây là một đồ thị quá trình học điển hình của một mô hình bị dưới khớp. Cả hai đường cong đi ngang, gần nhau và hàm chi phí vẫn có giá trị khá lớn.

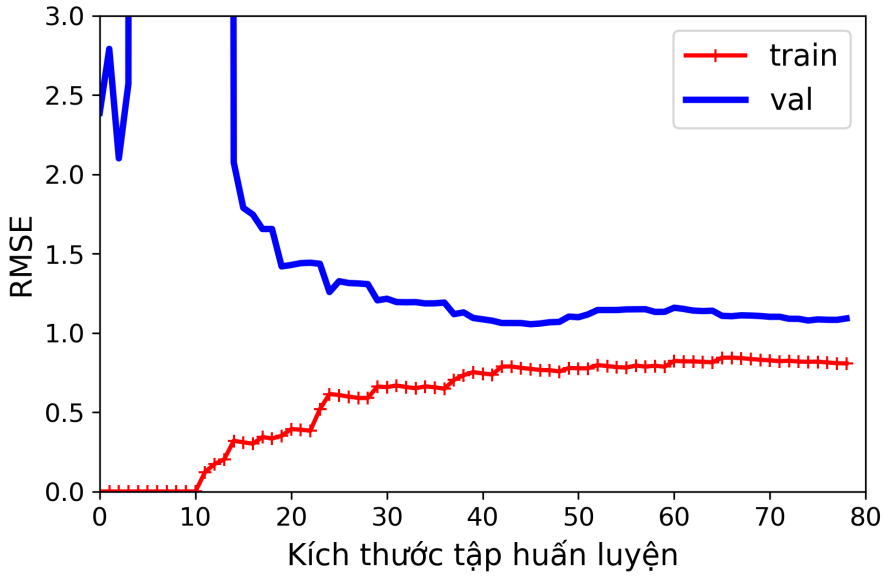
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Mẹo

Nếu mô hình của bạn đang bị dưới khớp trên tập dữ liệu huấn luyện, tăng kích thước của tập huấn luyện sẽ không có tác dụng. Bạn cần sử dụng một mô hình phức tạp hơn hoặc các đặc trưng tốt hơn.

Bây giờ hãy nhìn vào đồ thị quá trình học của một mô hình đa thức bậc 10 trên cùng tập dữ liệu (Hình 4.16):

```
from sklearn.pipeline import Pipeline

polynomial_regression = Pipeline([
        ("poly_features", PolynomialFeatures(degree=10, include_bias=False)),
        ("lin_reg", LinearRegression()),
    ])

plot_learning_curves(polynomial_regression, X, y)
```



Hình 4.16. Đồ thị quá trình học của mô hình đa thức bậc 10

Đồ thị này nhìn khá giống với đồ thị trước, dù vậy nó có hai điểm khác biệt rất quan trọng:

- Sai số trên tập huấn luyện nhỏ hơn nhiều so với mô hình Hồi quy Tuyến tính.
- Có một khoảng trống giữa hai đường cong. Điều này có nghĩa mô hình hoạt động trên tập huấn luyện tốt hơn đáng kể so với trên tập kiểm định, dấu hiệu cho thấy mô hình đang quá khớp. Tuy nhiên, nếu ta tiếp tục tăng kích thước tập huấn luyện, hai đường cong sẽ tiếp tục tiến lại gần nhau hơn nữa.

Mẹo

Một cách để cải thiện mô hình quá khớp là tiếp tục thêm dữ liệu huấn luyện cho đến khi sai số kiểm định bằng với sai số huấn luyện.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Đánh đổi Độ chệch/Phương sai

Một lý thuyết quan trọng trong Thống kê và Học Máy đó là sai số khái quát hóa của mỗi mô hình đều có thể được biểu diễn dưới dạng tổng của ba sai số riêng biệt:

Độ chệch (Bias)
Phần này của sai số khái quát hóa bắt nguồn từ việc đặt sai giả thuyết, chẳng hạn như cho rằng dữ liệu là tuyến tính trong khi nó thực chất là bậc hai. Một mô hình có độ chệch cao rất có thể sẽ dưới khớp dữ liệu huấn luyện.[8]

Phương sai (Variance)
Phần này đến từ việc mô hình cực kỳ nhạy với những thay đổi trong dữ liệu huấn luyện. Một mô hình với bậc tự do cao (một mô hình đa thức bậc cao chẳng hạn) sẽ có khả năng biến đổi linh hoạt và vì thế trở nên quá khớp với dữ liệu huấn luyện.

Sai số bất khả giảm (Irreducible error)
Phần này đến từ chính sự nhiễu loạn trong dữ liệu. Cách duy nhất để giảm sai số loại này đó là làm sạch dữ liệu (ví dụ, kiểm tra lại nguồn nhận dữ liệu: sửa các cảm biến hỏng, hoặc phát hiện và loại bỏ các điểm ngoại lai).

Tăng độ phức tạp của một mô hình thường sẽ tăng phương sai và giảm độ chệch của nó. Ngược lại, giảm độ phức tạp của một mô hình sẽ tăng độ chệch và giảm phương sai. Đó là lý do tại sao đây được gọi là lý thuyết "đánh đổi".

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Mô hình Tuyến tính Điều chuẩn

Như được trình bày trong Chương 1 và Chương 2, một cách hiệu quả để giảm quá khớp là điều chuẩn mô hình (tức ràng buộc mô hình): số bậc tự do càng ít, mô hình càng khó quá khớp dữ liệu. Một cách đơn giản để điều chuẩn mô hình đa thức là giảm số bậc của đa thức đó.

Với một mô hình tuyến tính, việc điều chuẩn thường đạt được bằng cách ràng buộc trọng số của mô hình. Sau đây, chúng tôi sẽ trình bày 3 mô hình: Hồi quy Ridge, Hồi quy Lasso và Elastic Net với ba cách ràng buộc trọng số khác nhau.

Hồi quy Ridge

Hồi Quy Ridge (*Ridge Regression*, còn gọi là *điều chuẩn Tikhonov*) là một phiên bản được điều chuẩn của Hồi quy Tuyến tính: *tham số điều chuẩn* (*regularization term*) $\alpha \sum_{i=1}^{n} \theta_i^2$ được thêm vào hàm chi phí. Việc này buộc thuật toán không những phải khớp dữ liệu mà còn phải giữ trọng số của mô hình nhỏ nhất có thể. Lưu ý rằng, tham số điều chuẩn chỉ nên được thêm vào hàm chi phí dành cho quá trình huấn luyện. Sau khi huấn luyện, chất lượng mô hình nên được đánh giá với hàm chi phí không dùng điều chuẩn.

[8] Khái niệm độ chệch (bias) này không nên bị nhầm lẫn với khái niệm hệ số điều chỉnh (cũng có tên tiếng Anh là bias) trong mô hình tuyến tính.

Ghi chú

Việc dùng các hàm chi phí khác nhau cho quá trình huấn luyện và kiểm tra mô hình khá phổ biến. Ngoài mục đích điều chuẩn, một hàm chi phí tốt cho huấn luyện cần phải có đạo hàm thuận tiện cho việc tối ưu, trong khi các phép đo sử dụng khi kiểm tra cần càng gần với mục tiêu cuối cùng càng tốt. Ví dụ, mô hình phân loại thường được huấn luyện bằng hàm chi phí entropy chéo (sẽ được đề cập sau) nhưng lại được đánh giá bằng precision/recall.

Siêu tham số α kiểm soát mức độ điều chuẩn mô hình. Nếu $\alpha = 0$, thì Hồi quy Ridge trở thành Hồi quy Tuyến Tính. Nếu α rất lớn, thì tất cả trọng số sau quá trình huấn luyện sẽ rất gần với 0, và kết quả là một đường thẳng đi qua giá trị trung bình của dữ liệu. Phương trình 4.8 định nghĩa hàm chi phí của Hồi quy Ridge.[9]

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \frac{1}{2} \sum_{i=1}^{n} {\theta_i}^2$$

Phương trình 4.8. Hàm chi phí của Hồi quy Ridge

Lưu ý rằng hệ số điều chỉnh θ_0 không được điều chuẩn (tổng bắt đầu từ $i = 1$ thay vì $i = 0$). Nếu $\mathbf{w}$ được định nghĩa là vector của các trọng số từ (θ_1 đến θ_n), thì tham số điều chuẩn bằng với $\frac{1}{2}(\|\mathbf{w}\|_2)^2$, trong đó $\|\mathbf{w}\|_2$ là chuẩn ℓ_2 của vector trọng số.[10] Đối với thuật toán Hạ Gradient, ta chỉ cần cộng $\alpha\mathbf{w}$ vào vector gradient MSE (Phương trình 4.6).

Lưu ý

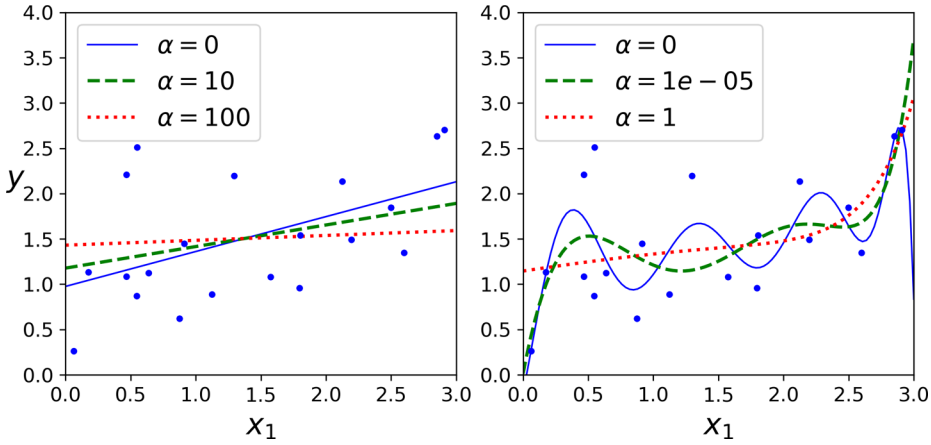
Việc co giãn dữ liệu (bằng cách sử dụng `StandardScaler`) trước khi thực hiện Hồi quy Ridge rất quan trọng, bởi vì mô hình này nhạy cảm với tỉ lệ của các đặc trưng đầu vào. Hầu hết các mô hình được điều chuẩn đều như vậy.

Hình 4.17 biểu thị các mô hình Ridge được huấn luyện với các giá trị α khác nhau trên một tập dữ liệu tuyến tính. Ở bên trái là các mô hình Ridge đơn thuần, cho ra dự đoán tuyến tính. Ở bên phải, đầu tiên dữ liệu được mở rộng bằng `PolynomialFeatures(degree=10)`, sau đó được co giãn bằng `StandardScaler`. Cuối cùng, mô hình Ridge được áp dụng cho các đặc trưng đã qua tiền xử lý: đây là Hồi quy Đa thức với điều chuẩn Ridge. Chú ý rằng khi α tăng, thì các dự đoán trở nên phẳng hơn (nghĩa là ít các cực trị và hợp lý hơn), do đó phương sai của mô hình giảm nhưng độ chệch lại tăng.

[9] Ký hiệu $J(\boldsymbol{\theta})$ thường được sử dụng cho hàm chi phí, và sẽ được sử dụng trong suốt phần còn lại của cuốn sách. Tùy ngữ cảnh, ta sẽ biết cụ thể hàm chi phí nào đang được thảo luận.

[10] Các loại chuẩn được thảo luận trong Chương 2.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.17. Mô hình tuyến tính (trái) và mô hình đa thức (phải) với những mức độ điều chuẩn Ridge khác nhau

Như với Hồi quy Tuyến tính, ta có thể thực hiện Hồi quy Ridge bằng cách tính phương trình dạng đóng hoặc bằng Hạ Gradient. Ưu và nhược điểm của các phương pháp là không đổi. Phương trình 4.9 trình bày biểu thức dạng đóng, trong đó, $\mathbf{A}$ là *ma trận đơn vị* (*identity matrix*)[11] có kích thước $(n+1) \times (n+1)$, nhưng có phần tử ở góc trên bên trái là 0 tương ứng với hệ số điều chỉnh.

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^\top \mathbf{X} + \alpha \mathbf{A}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Phương trình 4.9. Phương trình dạng đóng của Hồi quy Ridge

Đây là cách thực hiện Hồi quy Ridge với biểu thức dạng đóng trong Scikit-Learn (một biến thể của Phương trình 4.9, sử dụng một kỹ thuật phân rã ma trận của André-Louis Cholesky):

```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[1.55071465]])
```

Còn đây là cách sử dụng với Hạ Gradient Ngẫu nhiên:[12]

```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([1.47012588])
```

Siêu tham số `penalty` quyết định loại điều chuẩn nào được sử dụng. Việc gán giá trị `"l2"` có nghĩa là thuật toán SGD sẽ thêm hệ số điều chuẩn bằng với một nửa căn bậc hai của chuẩn ℓ_2 của vector trọng số vào hàm chi phí: nói cách khác, đó là Hồi quy Ridge.

[11] Một ma trận vuông chỉ chứa giá trị 0, ngoại trừ đường chéo chính (từ góc trên trái đến góc dưới phải) chứa giá trị 1.

[12] Ngoài ra, ta có thể sử dụng lớp `Ridge` với bộ giải (*solver*) `"sag"`. Hạ Gradient Ngẫu nhiên Trung bình (*Stochastic Average GD*) là một biến thể của Hạ Gradient Ngẫu nhiên. Để biết thêm chi tiết, hãy tham khảo bài thuyết trình "Minimizing Finite Sums with the Stochastic Average Gradient Algorithm" của Mark Schmidt và cộng sự từ Đại học British Columbia.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

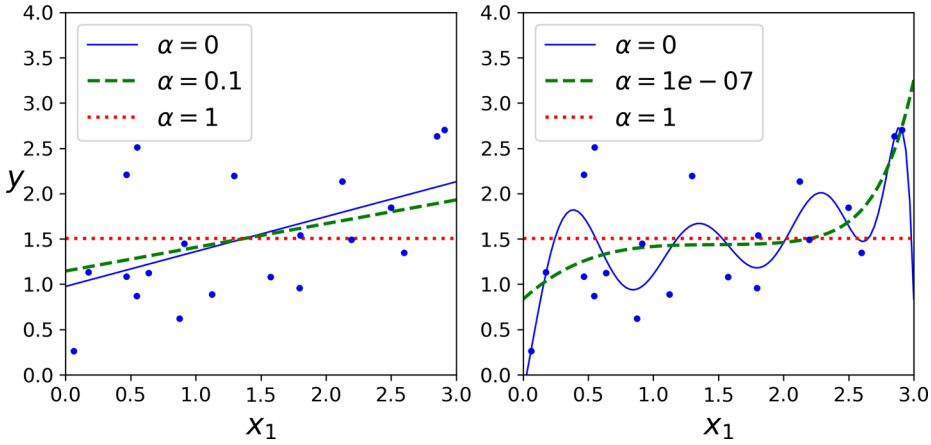
Hồi quy Lasso

Hồi quy Toán tử Lựa chọn và Co Tuyệt đối Nhỏ nhất (*Least Absolute Shrinkage and Selection Operator Regression*, thường được gọi là *Hồi quy Lasso*) là một phiên bản điều chuẩn khác của Hồi quy Tuyến tính. Tương tự như Hồi quy Ridge, phương pháp này thêm hệ số điều chuẩn vào hàm chi phí, nhưng sử dụng chuẩn ℓ_1 của vector trọng số thay vì một nửa bình phương của chuẩn ℓ_2 (tham khảo Phương trình 4.10)

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{\theta}) + \alpha \sum_{i=1}^{n} |\theta_i|$$

Phương trình 4.10. Hàm chi phí của Hồi quy Lasso

Hình 4.18 trình bày cùng nội dung với Hình 4.17 nhưng thay mô hình Ridge bằng mô hình Lasso và sử dụng giá trị α nhỏ hơn.



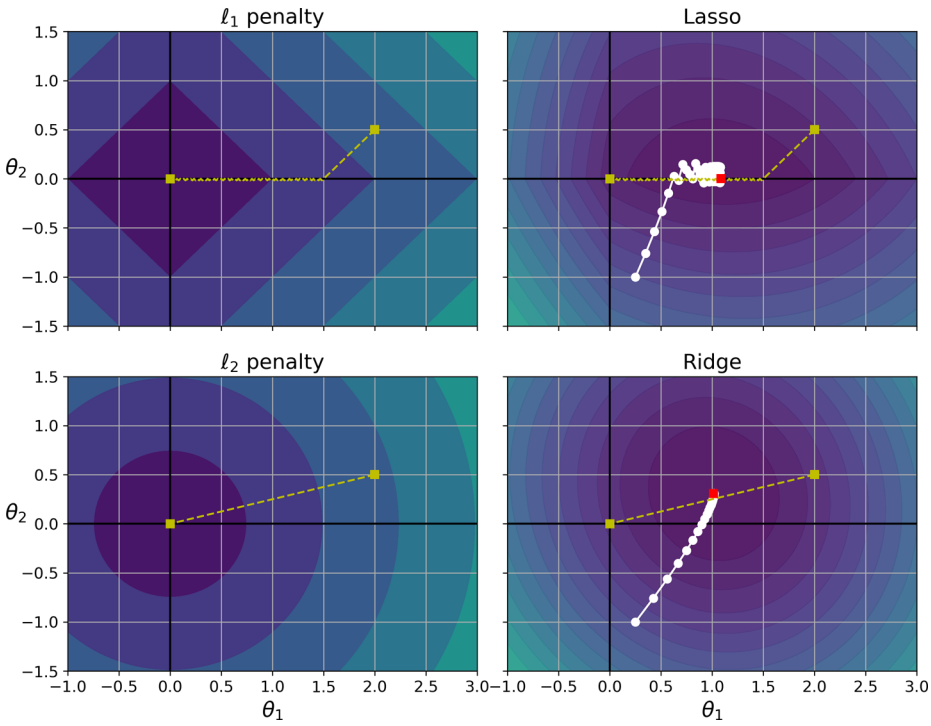
Hình 4.18. Mô hình Tuyến tính (trái) và Mô hình Đa thức (phải) với các mức độ điều chuẩn Lasso khác nhau

Một tính chất quan trọng của Hồi quy Lasso là nó có khuynh hướng loại bỏ trọng số của những đặc trưng ít quan trọng (nghĩa là đưa chúng về 0). Ví dụ, đường nét đứt trên biểu đồ bên phải trong Hình 4.18 (với $\alpha = 10^{-7}$) trông giống hàm bậc hai, gần như tuyến tính: tất cả trọng số cho những đặc trưng đa thức ở bậc cao đều bằng 0. Nói cách khác, Hồi quy Lasso tự động lựa chọn đặc trưng và cho ra một *mô hình thưa* (*sparse model* – mô hình chỉ có một vài trọng số đặc trưng khác không).

Ta có thể hiểu vì sao lại như vậy khi nhìn vào Hình 4.19: các trục biểu diễn hai tham số của mô hình và các đường đồng mức đại diện cho những hàm mất mát khác nhau. Ở biểu đồ góc trên-trái, các đường đồng mức biểu diễn giá trị mất mát ℓ_1 – có giá trị $(|\theta_1| + |\theta_2|)$ – giá trị này giảm tuyến tính khi tiến gần đến các trục. Ví dụ, nếu tham số mô hình được khởi tạo với giá trị $\theta_1 = 2$ và $\theta_2 = 0.5$, Hạ Gradient sẽ làm giảm cả hai tham số như nhau (đường đứt nét màu vàng) vì vậy θ_2 sẽ đạt giá trị 0 trước (vì được khởi tạo gần 0 hơn). Sau đó, Hạ Gradient sẽ giảm nhanh cho đến khi $\theta_1 = 0$ (với một chút dao động quanh 0 vì các gradient của ℓ_1 sẽ không bao giờ gần với 0: chúng sẽ bằng -1 hoặc 1 cho mỗi tham số). Ở biểu đồ góc trên-phải, các đường đồng mức biểu diễn hàm chi phí theo hồi quy Lasso (là hàm chi phí MSE cộng hàm mất mát ℓ_1). Các điểm tròn trắng biểu diễn quá trình tối ưu bằng Hạ Gradient, với các tham số của mô hình được khởi tạo là $\theta_1 = 0.25$ và $\theta_2 = -1$: lưu ý lại rằng θ_2 đạt giá trị 0 một cách nhanh chóng, sau đó quỹ đạo hầu như chỉ tiến theo phương θ_1 và cuối cùng dao động quanh điểm tối ưu toàn

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

cục (điểm hình vuông màu đỏ). Nếu tăng α, điểm tối ưu toàn cục sẽ dịch chuyển về bên trái dọc theo đường nét đứt màu vàng; trong khi nếu giảm α, điểm tối ưu toàn cục sẽ dịch chuyển về phía phải (trong ví dụ này, các tham số tối ưu cho MSE chưa điều chuẩn là $\theta_1 = 2$ và $\theta_2 = 0.5$).



Hình 4.19. Điều chuẩn Lasso so với điều chuẩn Ridge

Hai biểu đồ phía dưới cũng tương tự như trên nhưng sử dụng lượng phạt ℓ_2. Biểu đồ góc dưới-trái cho thấy giá trị mất mát ℓ_2 giảm theo khoảng cách đến gốc toạ độ, vì vậy Hạ Gradient chỉ đi thẳng tới đó. Các đường đồng mức trong đồ thị ở góc dưới-phải đại diện cho hàm chi phí theo Hồi quy Ridge (là hàm chi phí MSE cộng với hàm mất mát ℓ_2). Có hai điểm khác biệt chính với Lasso. Thứ nhất, các giá trị gradient càng nhỏ khi tham số tiến càng gần tới điểm tối ưu toàn cục, vì vậy Hạ Gradient chậm xuống một cách tự nhiên hỗ trợ cho việc hội tụ (vì không có dao động). Thứ hai, các tham số tối ưu (đại diện bởi hình vuông màu đỏ) càng gần với gốc toạ độ khi càng tăng α nhưng không bao giờ nằm chính xác tại gốc toạ độ.

Mẹo

Để tránh việc Hạ Gradient dao động xung quanh điểm tối ưu lúc gần kết thúc khi sử dụng Lasso, tốc độ học nên được giảm từ từ trong quá trình huấn luyện (không thể loại bỏ hoàn toàn các bước dao động, nhưng chúng sẽ nhỏ dần và thuật toán sẽ hội tụ).

Hàm chi phí Lasso không khả vi tại $\theta_i = 0$ (với $i = 1, 2, \cdots, n$), nhưng Hạ Gradient vẫn hoạt động tốt nếu sử dụng *vector subgradient* $\mathbf{g}$[13] khi $\theta_i = 0$. Phương trình 4.11 biểu diễn công thức vector subgradient được sử dụng cho Hạ Gradient với hàm chi phí Lasso.

Sau đây là một ví dụ nhỏ về việc sử dụng lớp `Lasso` trong Scikit-Learn:

```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
```

[13] Vector subgradient tại một điểm không khả vi có thể được hiểu là vector trung bình của các vector gradient xung quanh điểm đó.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$g(\boldsymbol{\theta}, J) = \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) + \alpha \begin{pmatrix} sign(\theta_1) \\ sign(\theta_2) \\ \vdots \\ sign(\theta_n) \end{pmatrix} \quad \text{với } sign(\theta_i) = \begin{cases} -1 & \text{nếu } \theta_i < 0 \\ 0 & \text{nếu } \theta_i = 0 \\ +1 & \text{nếu } \theta_i > 0 \end{cases}$$

Phương trình 4.11. Vector subgradient của Hồi quy Lasso

```
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([1.53788174])
```

Lưu ý rằng ta cũng có thể thay bằng `SGDRegressor(penalty="l1")`.

Elastic Net

Elastic Net dung hoà Hồi quy Ridge và Hồi quy Lasso, bằng cách kết hợp các hệ số điều chuẩn của Ridge và Lasso với tỉ lệ kết hợp r. Khi $r = 0$, Elastic Net tương đương với Hồi quy Ridge, và khi $r = 1$, nó tương đương với Hồi quy Lasso (tham khảo Phương trình 4.12).

$$J(\boldsymbol{\theta}) = \text{MSE}(\boldsymbol{\theta}) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} {\theta_i}^2$$

Phương trình 4.12. Hàm chi phí của Elastic

Vậy khi nào nên sử dụng Hồi quy Tuyến tính đơn thuần (không điều chuẩn), Ridge, Lasso, hay Elastic Net? Gần như trong mọi trường hợp, dù ít hay nhiều ta cũng nên sử dụng điều chuẩn và tránh chỉ sử dụng Hồi quy Tuyến tính đơn thuần. Ridge là một mặc định hiệu quả nhưng nếu vẫn nghi ngờ rằng chỉ có một vài đặc trưng hữu ích, ta nên sử dụng Lasso hay Elastic Net vì chúng có khuynh hướng giảm trọng số của những đặc trưng không có ích về 0 như đã thảo luận ở trên. Thông thường, Elastic Net được ưa chuộng hơn Lasso vì Lasso có thể hoạt động bất thường khi lượng đặc trưng lớn hơn lượng mẫu huấn luyện hoặc khi một vài đặc trưng tương quan chặt chẽ với nhau.

Sau đây là một ví dụ nhỏ về việc sử dụng `ElasticNet` trong Scikit-Learn (`l1_ratio` tương ứng với tỉ lệ kết hợp r):

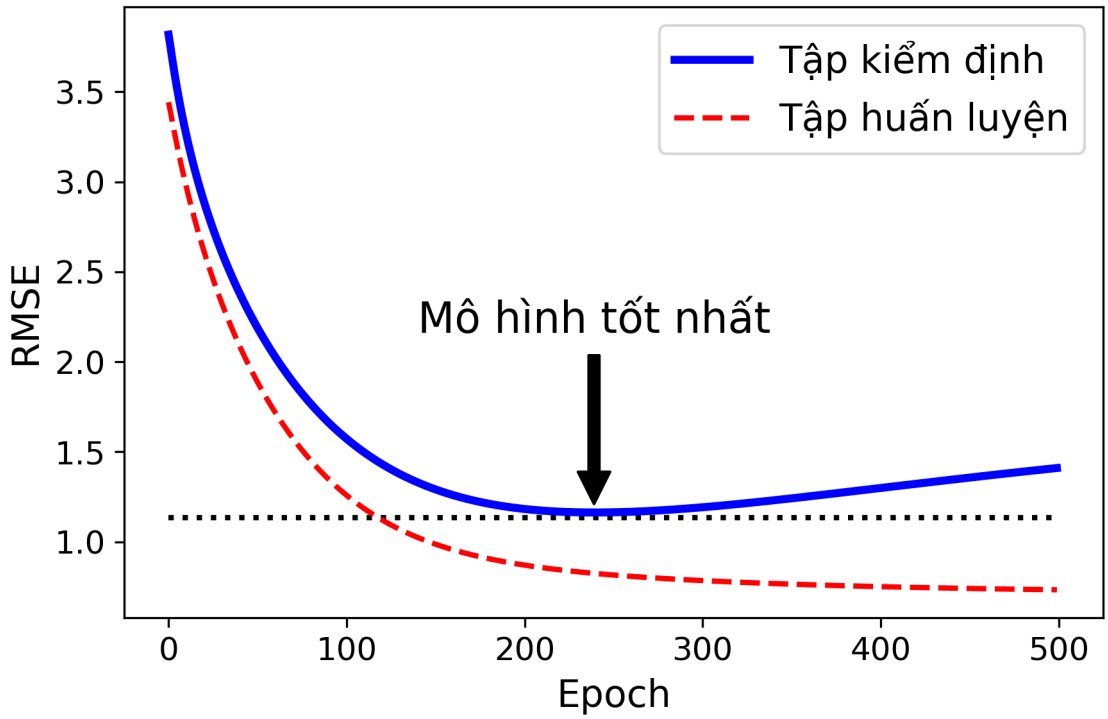
```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```

Phương pháp Dừng Sớm

Một cách khác để điều chuẩn thuật toán học lặp lại như Hạ Gradient là dừng huấn luyện ngay khi lỗi kiểm định đã đạt giá trị nhỏ nhất. Cách này được gọi là *dừng sớm* (*early stopping*). Hình 4.20 trình bày một mô hình phức tạp (trong trường hợp này là một mô hình Hồi quy Đa thức bậc cao) được huấn luyện với Hạ Gradient theo Batch. Sau mỗi epoch, lỗi dự đoán trên tập huấn luyện thường giảm xuống cùng với lỗi dự đoán trên tập kiểm định. Tuy nhiên sau một

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

vài epoch, lỗi kiểm định ngừng giảm xuống và bắt đầu tăng lên. Dấu hiệu này cho thấy mô hình bắt đầu quá khớp trên tập huấn luyện. Bằng cách dừng sớm, ta dừng quá trình huấn luyện ngay khi lỗi kiểm định đạt giá trị nhỏ nhất. Đây là một kỹ thuật điều chuẩn đơn giản và hiệu quả mà Geoffrey Hinton gọi là "một bữa trưa miễn phí ngon lành".



Hình 4.20. Điều chuẩn bằng cách dừng sớm

Mẹo

Với Hạ Gradient Ngẫu nhiên và Hạ Gradient theo Mini-batch, đồ thị sẽ không được mềm mại và có thể khó để biết được ta đã đạt được giá trị nhỏ nhất hay chưa. Một giải pháp là chỉ dừng huấn luyện một vài epoch sau khi lỗi kiểm định vượt quá giá trị nhỏ nhất (để tự tin rằng lỗi kiểm định không tiếp tục giảm xuống), sau đó quay lại với tham số của mô hình tại điểm mà lỗi kiểm định nhỏ nhất.

Sau đây là một ví dụ cơ bản về dừng sớm:

```
from sklearn.base import clone

# prepare the data
poly_scaler = Pipeline([
        ("poly_features", PolynomialFeatures(degree=90, include_bias=False)),
        ("std_scaler", StandardScaler())
    ])
X_train_poly_scaled = poly_scaler.fit_transform(X_train)
X_val_poly_scaled = poly_scaler.transform(X_val)

sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
                        penalty=None, learning_rate="constant", eta0=0.0005)

minimum_val_error = float("inf")
best_epoch = None
best_model = None
for epoch in range(1000):
    sgd_reg.fit(X_train_poly_scaled, y_train)  # continues where it left off
    y_val_predict = sgd_reg.predict(X_val_poly_scaled)
    val_error = mean_squared_error(y_val, y_val_predict)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = clone(sgd_reg)
```

Lưu ý rằng với `warm_start=True`, khi gọi phương thức `fit()`, việc huấn luyện sẽ tiếp diễn ngay tại thời điểm dừng huấn luyện trước đó thay vì bắt đầu lại từ đầu.

Hồi quy Logistic

Như đã thảo luận trong Chương 1, một vài thuật toán hồi quy có thể được sử dụng cho bài toán phân loại (và ngược lại). *Hồi quy Logistic* (*Logistic Regression*, còn được gọi là *Hồi quy Logit* – *Logit Regression*) được sử dụng phổ biến để ước lượng xác suất một mẫu dữ liệu thuộc về một lớp cụ thể nào đó (ví dụ, xác suất một email là thư rác). Nếu xác suất ước lượng cho một lớp lớn hơn 50%, thì mô hình dự đoán mẫu này thuộc về lớp đó (được gọi là *lớp dương*, được gán nhãn là "1"); nếu không, mô hình dự đoán mẫu không thuộc lớp đó (tức thuộc *lớp âm*, được gán nhãn là "0"). Vì vậy, đây là một bộ phân loại nhị phân.

Ước lượng Xác suất

Vậy Hồi quy Logistic hoạt động như thế nào? Cũng giống như Hồi quy Tuyến tính, mô hình Hồi quy Logistic tính tổng trọng số các đặc trưng đầu vào (cộng với hệ số điều chỉnh), nhưng thay vì cho ra kết quả trực tiếp như mô hình Hồi quy Tuyến tính, nó cho ra *logistic* của tổng này (tham khảo Phương trình 4.13).

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^\top \boldsymbol{\theta}\right)$$

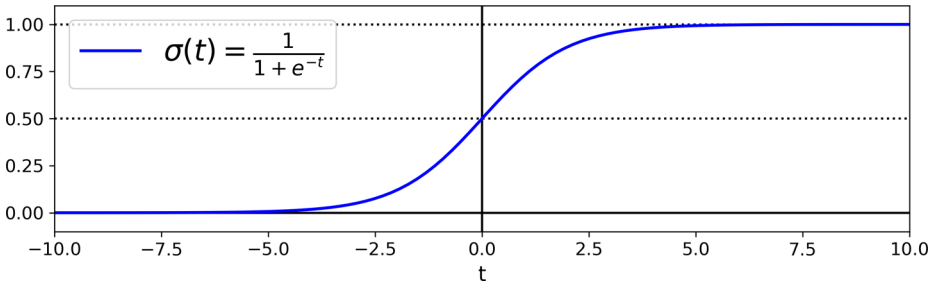
Phương trình 4.13. Xác suất ước lượng của mô hình Hồi quy Logistic (dạng vector)

Logistic – được ký hiệu là $\sigma(\cdot)$ – là một *hàm sigmoid* (*sigmoid function*) (có đồ thị dạng chữ S) cho đầu ra từ 0 đến 1. Hàm này được mô tả trong Phương trình 4.14 và Hình 4.21.

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Phương trình 4.14. Hàm logistic

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.21. Hàm Logistic

Khi mô hình Hồi quy Logistic đã ước lượng xác suất mẫu $\mathbf{x}$ thuộc lớp dương $\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x})$, nó có thể đưa ra dự đoán một cách dễ dàng (tham khảo Phương trình 4.15).

$$\hat{y} = \begin{cases} 0 & \text{nếu } \hat{p} < 0.5 \\ 1 & \text{nếu } \hat{p} \geq 0.5 \end{cases}$$

Phương trình 4.15. Dự đoán của mô hình Hồi quy Logistic

Lưu ý rằng $\sigma(t) < 0.5$ khi $t < 0$ và $\sigma(t) \geq 0.5$ khi $t \geq 0$, vì vậy mô hình Hồi quy Logistic dự đoán là 1 nếu giá trị $\boldsymbol{\theta}^\top \mathbf{x}$ là dương và 0 nếu là âm.

Ghi chú

t thường được gọi là *logit*. Cái tên này xuất phát từ việc hàm logit – được định nghĩa $\text{logit}(p) = \log(p/(1-p))$ – là nghịch đảo của hàm logistic. Thật vậy, nếu tính logit của xác suất ước lượng p, ta sẽ thu được kết quả là t. Logit còn được gọi là *log-odd* vì nó là logarit của tỉ lệ giữa xác suất ước lượng cho lớp dương và lớp âm.

Huấn luyện và Hàm Chi phí

Bây giờ ta đã biết cách một mô hình Hồi quy Logistic ước lượng xác suất và đưa ra dự đoán, nhưng làm cách nào để huấn luyện mô hình? Mục tiêu của việc huấn luyện là tìm một vector tham số $\boldsymbol{\theta}$ sao cho mô hình ước lượng xác suất cao cho mẫu dương ($y = 1$) và xác suất thấp cho mẫu âm ($y = 0$). Ý tưởng này được thể hiện trong hàm chi phí ở Phương trình 4.16 cho mỗi mẫu dữ liệu huấn luyện $\mathbf{x}$.

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{nếu } y = 1 \\ -\log(1-\hat{p}) & \text{nếu } y = 0 \end{cases}$$

Phương trình 4.16. Hàm chi phí của một mẫu huấn luyện

Hàm chi phí này hợp lý vì $-\log(t)$ sẽ rất lớn khi t tiệm cận 0 nên chi phí sẽ rất lớn nếu mô hình ước lượng xác suất mẫu dương gần với 0. Tương tự, chi phí cũng sẽ rất lớn nếu mô hình ước lượng xác suất mẫu âm gần với 1. Mặt khác, $-\log(t)$ gần với 0 khi t gần với 1 nên chi phí sẽ gần bằng 0 nếu xác suất ước lượng gần với 0 cho mẫu thuộc lớp âm hoặc gần với 1 cho mẫu thuộc lớp dương. Đây chính là điều ta mong muốn.

Hàm chi phí trên toàn tập dữ luyện huấn luyện là trung bình chi phí trên toàn bộ các mẫu huấn luyện. Nó có thể được biểu diễn bằng một biểu thức đơn giản gọi là *logarit mất mát* (*log loss*), được trình bày ở Phương trình 4.17.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + (1-y^{(i)})\log\left(1-\hat{p}^{(i)}\right)\right]$$

Phương trình 4.17. Hàm chi phí Hồi quy Logistic

Tin xấu là không tồn tại phương trình dạng đóng để tính trực tiếp giá trị $\boldsymbol{\theta}$ mà tại đó hàm chi phí đạt giá trị nhỏ nhất (không có phiên bản tương tự như Phương trình Pháp tuyến). Còn tin tốt là hàm chi phí này là hàm lồi, do đó thuật toán Hạ Gradient (hay bất cứ thuật toán tối ưu nào khác) chắc chắn sẽ tìm được giá trị nhỏ nhất (nếu tốc độ học không quá cao và ta kiên nhẫn đợi). Đạo hàm riêng $\boldsymbol{\theta}_j$ của hàm chi phí theo tham số thứ j của mô hình được tính theo Phương trình 4.18.

$$\frac{\partial}{\partial\theta_j}\mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\boldsymbol{\theta}^\top\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

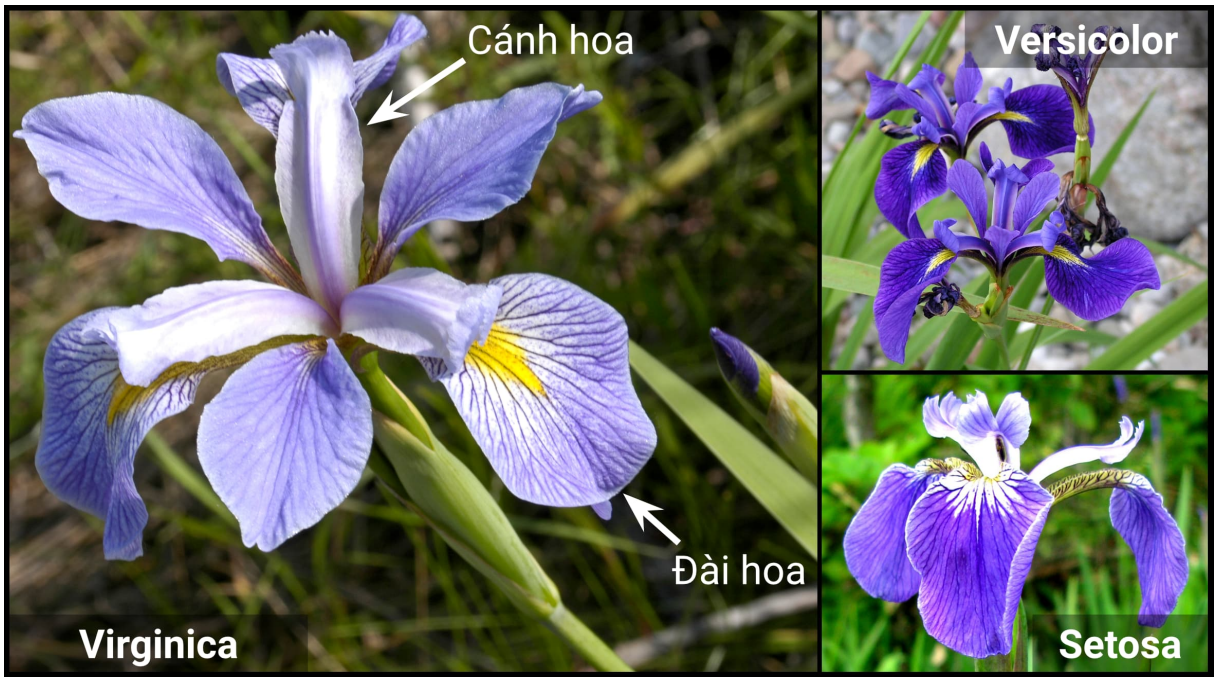
Phương trình 4.18. Đạo hàm riêng của hàm chi phí logistic

Phương trình này trông rất giống Phương trình 4.5: với mỗi mẫu, ta tính lỗi dự đoán và nhân nó với giá trị đặc trưng thứ j, rồi lấy trung bình trên toàn bộ tập huấn luyện. Một khi có được vector gradient chứa tất cả các đạo hàm riêng, ta có thể dùng vector này trong thuật toán Hạ Gradient theo Batch. Đó là cách huấn luyện mô hình Hồi quy Logistic. Nếu sử dụng Hạ Gradient Ngẫu nhiên, thì ta chỉ tính gradient trên một mẫu thay vì toàn bộ tập huấn luyện. Nếu sử dụng Hạ Gradient theo Mini-batch, thì chỉ tính gradient trên một mini-batch.

Ranh giới Quyết định

Ta sẽ sử dụng tập dữ liệu Iris để minh họa Hồi quy Logistic. Đây là một tập dữ liệu nổi tiếng chứa kích thước dài rộng của đài và cánh hoa của 150 bông hoa Iris thuộc ba loài khác nhau: *Iris setosa*, *Iris versicolor*, và *Iris virginica* (tham khảo Hình 4.22).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.22. Ảnh của ba loài hoa Iris[14]

Ta sẽ xây dựng một bộ phân loại để nhận diện hoa *Iris virginica* dựa trên một đặc trưng duy nhất, đó là độ rộng cánh hoa. Trước tiên, ta nạp dữ liệu như sau:

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> list(iris.keys())
['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename']
>>> X = iris["data"][:, 3:]  # petal width
>>> y = (iris["target"] == 2).astype(np.int)  # 1 if Iris virginica, else 0
```

Tiếp theo, ta sẽ huấn luyện một mô hình Hồi quy Logistic:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X, y)
```

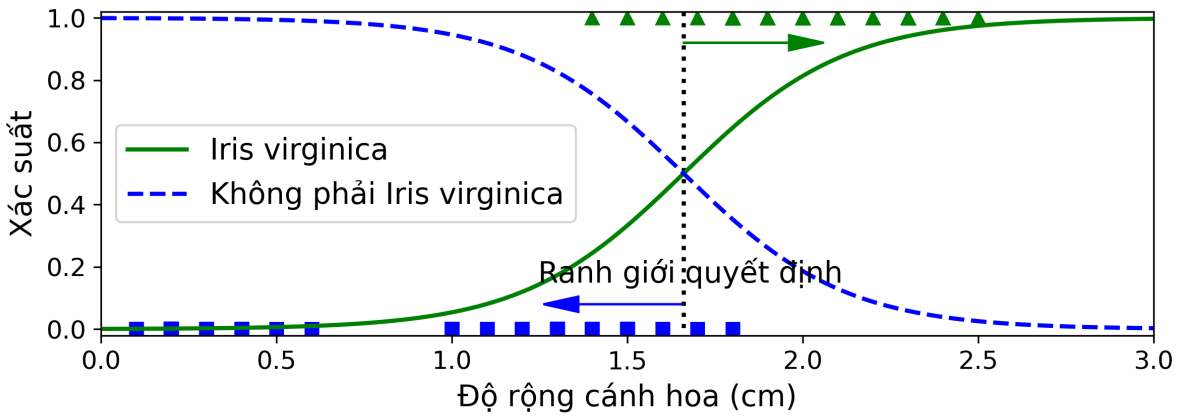
Hãy quan sát xác suất ước lượng cho những bông hoa có độ rộng cánh hoa từ 0 cm đến 3 cm (Hình 4.23):[15]

```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
plt.plot(X_new, y_proba[:, 1], "g-", label="Iris virginica")
plt.plot(X_new, y_proba[:, 0], "b--", label="Not Iris virginica")
# + more Matplotlib code to make the image look pretty
```

[14] Ảnh được sao chép từ các trang Wikipedia tương ứng. Ảnh *Iris virginica* của Frank Mayfield (Creative Commons BY-SA 2.0), ảnh *Iris versicolor* của D. Gordon E. Robertson (Creative Commons BY-SA 3.0), ảnh *Iris setosa* thuộc phạm vi công khai.

[15] Hàm `reshape()` của NumPy cho phép một chiều có giá trị bằng –1, tức "ngầm định": giá trị này được suy ra từ chiều dài của mảng và các chiều còn lại.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.23. Xác suất ước lượng và ranh giới quyết định

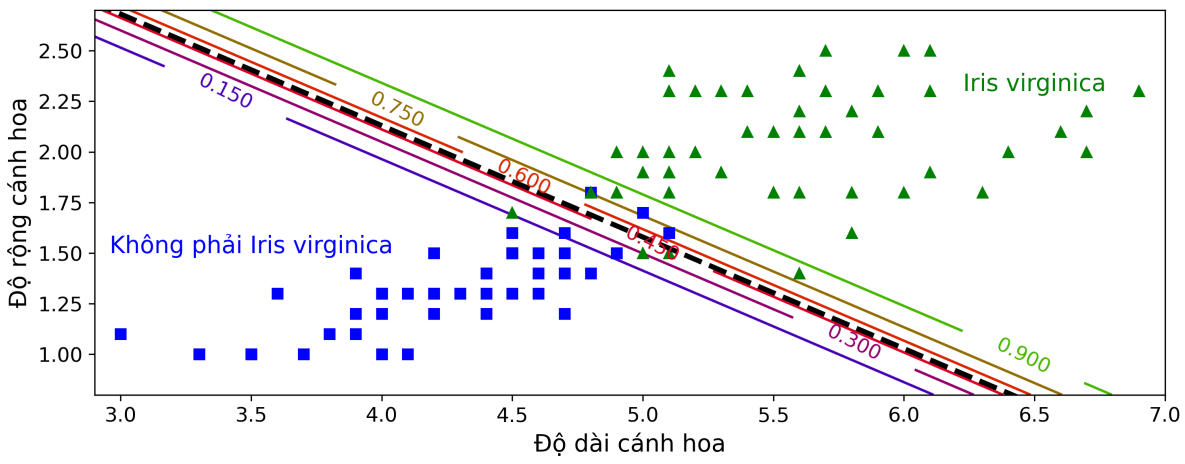
Chiều rộng cánh hoa *Iris virginica* (được biểu diễn bởi hình tam giác) nằm trong khoảng từ 1.4 cm đến 2.5 cm, trong khi các loài hoa Iris còn lại (được biểu diễn bởi hình vuông) thường có độ rộng cánh nhỏ hơn, từ 0.1 cm đến 1.8 cm. Chú ý rằng hai khoảng giá trị này có hơi giao nhau. Nếu độ rộng cánh hoa lớn hơn 2 cm, bộ phân loại sẽ dự đoán đây là hoa *Iris virginica* với độ tin cậy cao (đưa ra xác suất cao cho lớp "Iris virginica"). Nếu độ rộng cánh hoa nhỏ hơn 1 cm, bộ phân loại sẽ dự đoán đây không phải hoa *Iris virginica* (xác suất cao cho lớp "Không phải Iris virginica"). Nếu độ rộng cánh hoa nằm trong khoảng 1 cm đến 2 cm, bộ phân loại sẽ đưa ra dự đoán không chắc chắn. Tuy nhiên, nếu muốn biết dự đoán cụ thể (bằng cách sử dụng phương thức `predict()` thay vì `predict_proba()`), bộ phân loại sẽ đưa ra lớp có xác suất dự đoán cao nhất. Vì vậy, sẽ có một *ranh giới quyết định* (*decision boundary*) ở khoảng 1.6 cm mà tại đó xác suất của hai lớp đều bằng 50%: nếu độ rộng cánh hoa lớn hơn 1.6 cm, bộ phân loại sẽ dự đoán đây là hoa *Iris virginica* và ngược lại (kể cả khi không chắc chắn cho lắm):

```
>>> log_reg.predict([[1.7], [1.5]])
array([1, 0])
```

Hình 4.24 biểu diễn tập dữ liệu này với hai đặc trưng: độ rộng và độ dài cánh hoa. Sau khi được huấn luyện, bộ phân loại Hồi quy Logistic có thể ước lượng xác suất một bông hoa mới là hoa *Iris virginica* dựa trên hai đặc trưng trên. Đường nét đứt biểu diễn những điểm có xác suất ước lượng là 50%: đây là ranh giới quyết định của mô hình. Chú ý rằng đây là một ranh giới tuyến tính.[16]

[16] Ranh giới tuyến tính là tập hợp các điểm $\mathbf{x}$ sao cho $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, và tập hợp các điểm này là một đường thẳng.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 4.24. Ranh giới quyết định tuyến tính

Tương tự như các mô hình tuyến tính khác, mô hình Hồi quy Logistic có thể được điều chuẩn bằng lượng phạt ℓ_1 hoặc ℓ_2. Theo mặc định, Scikit-Learn sẽ sử dụng lượng phạt ℓ_2.

Ghi chú

Siêu tham số điều khiển mức điều chuẩn của mô hình `LogisticRegression` trong Scikit-learn không phải là `alpha` (như trong các mô hình tuyến tính khác), mà là giá trị nghịch đảo của nó: `C`. `C` có giá trị càng cao thì mô hình càng *ít* bị điều chuẩn.

Hồi quy Softmax

Mô hình Hồi quy Logistic có thể được tổng quát hóa cho bài toán phân loại đa lớp mà không cần phải huấn luyện và kết hợp nhiều bộ phân loại nhị phân (như đã thảo luận trong Chương 3). Mô hình tổng quát này được gọi là *Hồi quy Softmax* (*Softmax Regression*), hay *Hồi quy Logistic Đa thức* (*Multinomial Logistic Regression*).

Ý tưởng khá đơn giản: với một mẫu $\mathbf{x}$, mô hình Hồi quy Softmax sẽ tính điểm $s_k(\mathbf{x})$ cho mỗi lớp k, rồi ước lượng xác suất cho mỗi lớp bằng cách áp dụng *hàm softmax* (còn được gọi là *lũy thừa chuẩn – normalized exponential*) cho các điểm số đó. Phương trình để tính $s_k(\mathbf{x})$ khá quen thuộc, bởi đây chính là phương trình dự đoán trong Hồi quy Tuyến tính (tham khảo Phương trình 4.19).

$$s_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta}^{(k)}$$

Phương trình 4.19. Điểm số cho lớp k

Chú ý rằng mỗi lớp có một vector tham số $\boldsymbol{\theta}^{(k)}$ riêng. Các vector này thường được lưu dưới dạng các hàng trong *ma trận tham số* (*parameter matrix*) $\boldsymbol{\theta}$.

Sau khi tính xong điểm số của mỗi lớp cho mẫu $\mathbf{x}$, ta có thể ước lượng xác suất $\hat{p}_k$ mà mẫu này thuộc về lớp k bằng cách đưa các điểm số này qua hàm softmax (Phương trình 4.20). Hàm này tính lũy thừa của mỗi điểm số rồi chuẩn hóa bằng cách chia cho tổng của tất cả các lũy thừa. Những điểm số này thường được gọi là logit hoặc log-odd (mặc dù thực chất đây là các giá trị log-odd chưa chuẩn hóa).

Trong đó:

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$$

Phương trình 4.20. Hàm softmax

- K là số lớp.
- $\mathbf{s}(\mathbf{x})$ là vector chứa điểm số của mỗi lớp cho mẫu $\mathbf{x}$.
- $\sigma(\mathbf{s}(\mathbf{x}))_k$ là xác suất ước lượng mà mẫu $\mathbf{x}$ thuộc lớp k khi biết điểm số của mỗi lớp cho mẫu này.

Tương tự như bộ phân loại Hồi quy Logistic, dự đoán của bộ phân loại Hồi quy Softmax là lớp có xác suất ước lượng cao nhất (tức lớp có điểm cao nhất), như trong Phương trình 4.21.

$$\hat{y} = \underset{k}{\operatorname{argmax}}\, \sigma(\mathbf{s}(\mathbf{x}))_k = \underset{k}{\operatorname{argmax}}\, s_k(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \left(\left(\boldsymbol{\theta}^{(k)} \right)^{\top} \mathbf{x} \right)$$

Phương trình 4.21. Dự đoán của bộ phân loại Hồi quy Softmax

Toán tử *argmax* trả về giá trị của biến mà tại đó hàm đạt giá trị lớn nhất. Trong phương trình này, nó trả về giá trị của lớp k có xác suất ước lượng $\sigma(\mathbf{s}(\mathbf{x}))_k$ lớn nhất.

Mẹo

Bộ phân loại Hồi quy Softmax chỉ dự đoán một lớp một lần (đây là bộ phân loại đa lớp, không phải đa đầu ra), do đó ta chỉ nên dùng bộ phân loại này với các lớp loại trừ lẫn nhau (*mutually exclusive*), ví dụ như các loài hoa khác nhau. Ta không thể sử dụng bộ phân loại này để nhận diện nhiều người cùng lúc trong một bức ảnh.

Sau khi biết cách mô hình ước lượng xác suất và đưa ra dự đoán, ta hãy xem xét quá trình huấn luyện. Mục tiêu là thu được một mô hình có xác suất ước lượng cao cho lớp mục tiêu (và xác suất ước lượng thấp cho các lớp còn lại). Tối thiểu hóa hàm chi phí trong Phương trình 4.22, được gọi là *entropy chéo* (*cross entropy*), giúp ta đạt được mục tiêu trên vì nó sẽ phạt mô hình nếu ước lượng xác suất thấp cho lớp mục tiêu. Entropy chéo thường được dùng để đo lường sự tương tự giữa phân phối xác suất ước lượng và phân phối mục tiêu.

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

Phương trình 4.22. Hàm chi phí entropy chéo

Trong phương trình này:

- $y_k^{(i)}$ là xác suất mục tiêu mà mẫu thứ i thuộc về lớp k, thường có giá trị 1 hoặc 0, tương ứng với việc mẫu có hoặc không thuộc về lớp đó.

Lưu ý rằng khi chỉ có hai lớp ($K = 2$), hàm chi phí này tương đương với hàm chi phí của Hồi quy Logistic (logarit mất mát; tham khảo Phương trình 4.17).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Entropy Chéo

Entropy chéo có nguồn gốc từ lý thuyết thông tin. Giả sử ta muốn truyền thông tin về thời tiết hằng ngày một cách hiệu quả. Nếu có tám lựa chọn (nắng, mưa, v.v.), ta có thể sử dụng ba bit để mã hóa mỗi lựa chọn ($2^3 = 8$). Tuy nhiên, nếu như hầu hết các ngày đều nắng, thì sẽ hiệu quả hơn rất nhiều nếu ta mã hóa "nắng" chỉ với một bit (0) và mã hóa bảy lựa chọn còn lại bằng bốn bit (bắt đầu bằng 1). Entropy chéo đo số bit trung bình gửi đi của mỗi lựa chọn. Nếu giả định về thời tiết trên là đúng, entropy chéo sẽ chính bằng entropy của thời tiết (là nội tại không thể dự đoán được của nó). Nhưng nếu giả định trên là sai (ví dụ: trời thường xuyên mưa), entropy chéo sẽ lớn hơn một lượng được gọi là *phân kỳ Kullback–Leibler (Kullback–Leibler divergence – KL)*.

Entropy chéo giữa hai phân phối xác suất p và q được định nghĩa là $H(p, q) = -\Sigma_x p(x) \log q(x)$ (ít nhất trong trường hợp các phân phối rời rạc). Để biết thêm chi tiết, hãy tham khảo video của tác giả về chủ đề này.

Vector gradient theo $\boldsymbol{\theta}^{(k)}$ của hàm chi phí này được tính theo Phương trình 4.23.

$$\nabla_{\boldsymbol{\theta}^{(k)}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{p}_k^{(i)} - y_k^{(i)} \right) \mathbf{x}^{(i)}$$

Phương trình 4.23. Vector gradient của entropy chéo cho lớp k

Giờ ta có thể tính được vector gradient cho từng lớp, sau đó sử dụng thuật toán Hạ Gradient (hay bất cứ thuật toán tối ưu nào) để tìm ma trận tham số $\boldsymbol{\theta}$ cực tiểu hóa hàm chi phí trên.

Ta hãy sử dụng Hồi quy Softmax để phân loại hoa Iris thành ba lớp. `LogisticRegression` trong Scikit-Learn mặc định sử dụng chiến lược một-còn lại (*one-versus-the-rest*) khi huấn luyện mô hình với nhiều hơn hai lớp. Tuy nhiên, ta có thể thiết lập giá trị của siêu tham số `multi_class` thành `"multinomial"` để chuyển sang sử dụng mô hình Hồi quy Softmax. Tiếp theo, ta thiết lập bộ giải cho Hồi quy Softmax, ví dụ bộ giải `"lbfgs"` (tham khảo tài liệu Scikit-Learn để biết thêm chi tiết). Mô hình này cũng sử dụng phương pháp điều chuẩn mặc định là ℓ_2, mà ta có thể kiểm soát bằng cách điều chỉnh siêu tham số `C`:

```
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]

softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10)
softmax_reg.fit(X, y)
```

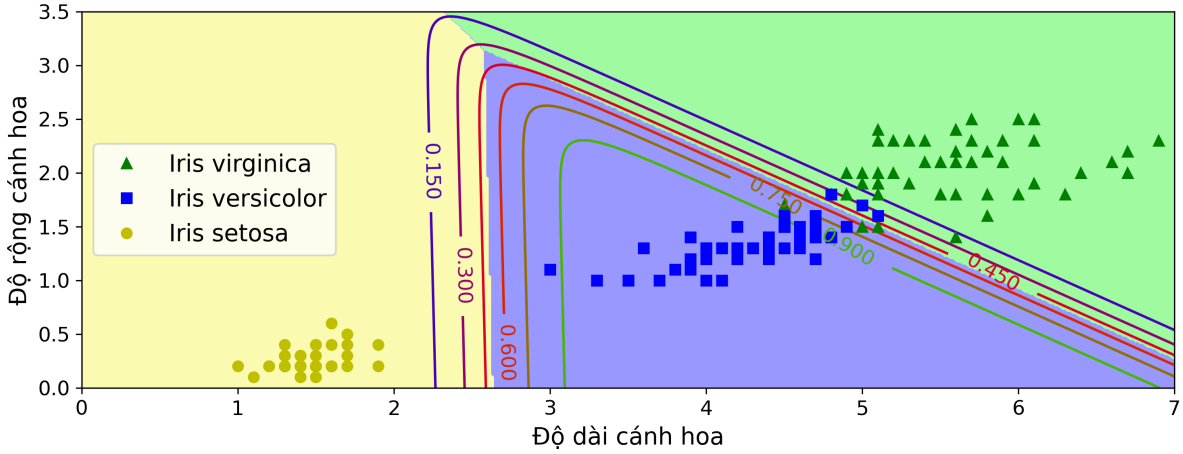
Giả sử ta bắt gặp một bông hoa Iris có cánh hoa dài 5 cm và rộng 2 cm, ta có thể yêu cầu mô hình dự đoán đó là loại hoa Iris nào. Mô hình sẽ trả về *Iris virginica* (lớp thứ 2) với xác suất 94.2% (hoặc *Iris versicolor* với xác suất 5.8%):

```
>>> softmax_reg.predict([[5, 2]])
array([2])
>>> softmax_reg.predict_proba([[5, 2]])
array([[6.38014896e-07, 5.74929995e-02, 9.42506362e-01]])
```

Hình 4.25 minh họa các ranh giới quyết định đã học được, chúng được biểu diễn bằng các màu nền khác nhau. Lưu ý rằng ranh giới quyết định giữa hai lớp bất kỳ là một đường thẳng tuyến

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

tính. Hình trên cũng minh họa xác suất cho lớp *Iris versicolor*, được biểu diễn bằng các đường cong (ví dụ, đường cong được gán nhãn 0.450 biểu diễn ranh giới xác suất là 45%). Cũng lưu ý rằng mô hình có thể dự đoán mẫu thuộc một lớp dù có xác suất ước lượng nhỏ hơn 50%. Ví dụ, tại điểm các ranh giới quyết định giao nhau, tất cả các lớp có cùng xác suất ước lượng là 33%.



Hình 4.25. Các ranh giới quyết định của Hồi quy Softmax

Bài tập

1. Nên sử dụng thuật toán nào để huấn luyện mô hình Hồi quy Tuyến tính, nếu tập huấn luyện có hàng triệu đặc trưng?

2. Giả sử các đặc trưng trong tập huấn luyện có các khoảng giá trị rất khác nhau. Giả định này có thể ảnh hưởng tới những thuật toán nào với mức độ ra sao? Ta có thể giải quyết vấn đề này bằng cách nào?

3. Thuật toán Hạ Gradient có thể bị kẹt trong điểm cực tiểu địa phương khi huấn luyện mô hình Hồi quy Logistic hay không?

4. Liệu tất cả các thuật toán Hạ Gradient sẽ dẫn đến cùng một mô hình, miễn là ta chạy thuật toán đủ lâu hay không?

5. Giả sử ta sử dụng thuật toán Hạ Gradient theo Batch và theo dõi lỗi kiểm định trong từng epoch, và phát hiện ra lỗi kiểm định tăng lên liên tục. Vấn đề này có thể là gì? Ta sẽ giải quyết vấn đề này bằng cách nào?

6. Liệu ta có nên dừng thuật toán Hạ Gradient theo Mini-batch ngay khi lỗi kiểm định tăng lên?

7. Thuật toán Hạ Gradient nào (trong số những thuật toán ta đã thảo luận) sẽ tới gần nghiệm tối ưu nhanh nhất? Thuật toán nào thực sự hội tụ? Ta có thể giúp các thuật toán khác cũng hội tụ bằng cách nào?

8. Giả sử ta đang sử dụng Hồi quy Đa thức. Khi theo dõi đồ thị quá trình học, ta nhận ra có một khoảng cách lớn giữa lỗi huấn luyện và lỗi kiểm định. Vấn đề gì đang xảy ra? Hãy nêu ra ba giải pháp cho vấn đề này.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

9. Giả sử ta đang sử dụng Hồi quy Ridge. Ta nhận ra rằng lỗi huấn luyện và lỗi kiểm định xấp xỉ bằng nhau và đều khá cao. Liệu mô hình có đang gặp vấn đề với độ chệch lớn hay phương sai lớn không? Ta nên tăng hay giảm giá trị siêu tham số điều chuẩn α?

10. Tại sao ta nên sử dụng:

    a. Hồi quy Ridge thay vì Hồi quy Tuyến tính (không sử dụng điều chuẩn)?

    b. Hồi quy Lasso thay vì Ridge?

    c. Elastic Net thay vì Lasso?

11. Giả sử ta muốn phân loại các bức ảnh theo nhãn ngoài trời/trong nhà và ban ngày/ban đêm. Liệu ta cần lập trình hai bộ phân loại Hồi quy Logistic hay chỉ cần một bộ phân loại Hồi quy Softmax?

12. Hãy lập trình thuật toán Hạ Gradient theo Batch với kỹ thuật dừng sớm cho Hồi quy Softmax mà không sử dụng thư viện Scikit-Learn.

Lời giải của các bài tập trên được trình bày trong Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 5

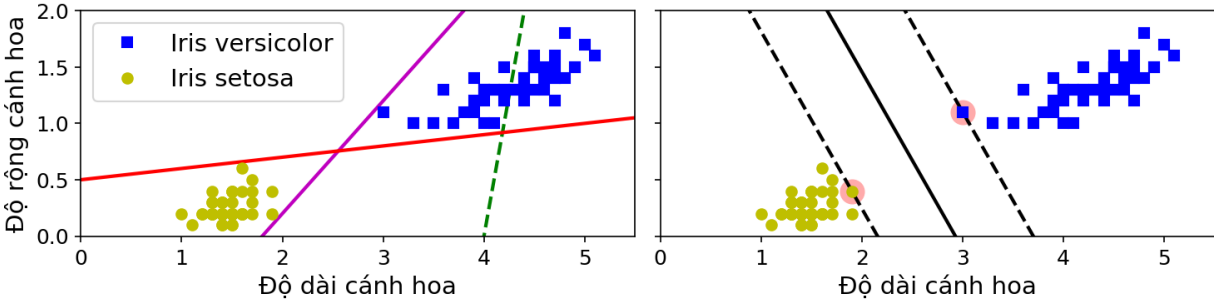
Máy Vector Hỗ trợ

Máy Vector Hỗ trợ (*Support Vector Machine* – SVM) là một mô hình Học Máy mạnh mẽ và linh hoạt, có khả năng thực hiện phân loại tuyến tính, phi tuyến, hồi quy, hay thậm chí là phát hiện ngoại lai. SVM là một trong những mô hình phổ biến nhất trong Học Máy và bất kỳ ai quan tâm đến Học Máy đều nên sở hữu nó trong bộ công cụ của mình. Các mô hình SVM đặc biệt phù hợp để phân loại các tập dữ liệu phức tạp có kích thước vừa và nhỏ.

Chương này sẽ giải thích các khái niệm cốt lõi, cách sử dụng cũng như cách hoạt động của SVM.

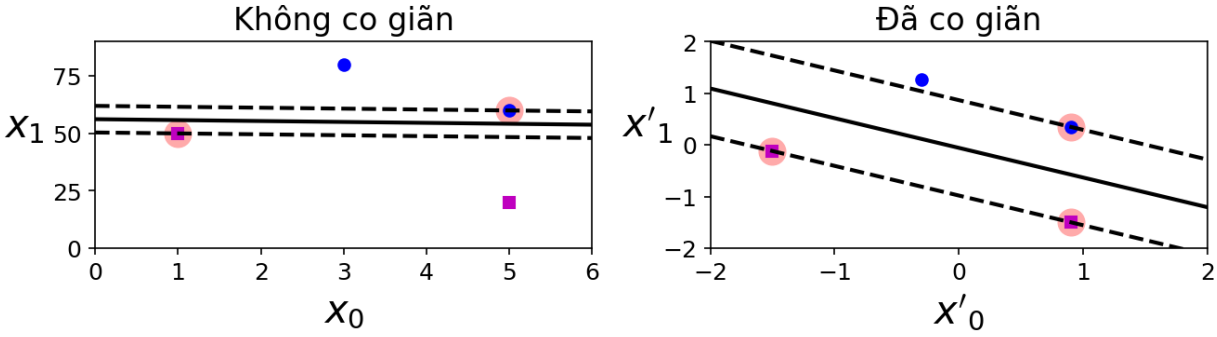
Phân loại với SVM Tuyến tính

Ý tưởng cơ bản của SVM có thể được giải thích bằng một vài hình minh họa dưới đây. Hình 5.1 biểu diễn một phần của tập dữ liệu Iris được giới thiệu tại cuối Chương 2. Ta thấy hai lớp có thể được phân tách rõ ràng bằng một đường thẳng (dữ liệu có tính *tách biệt tuyến tính – linearly separable*). Đồ thị bên trái biểu diễn các ranh giới quyết định của ba bộ phân loại tuyến tính khả thi. Mô hình với ranh giới quyết định được biểu diễn bằng nét đứt có chất lượng thấp đến mức không thể phân tách các lớp một cách chính xác. Hai mô hình còn lại thì hoạt động hoàn hảo trên tập huấn luyện, nhưng lại có đường ranh giới quyết định quá gần các mẫu, dẫn đến khả năng mô hình sẽ không hoạt động tốt với các mẫu mới. Mặt khác, đường nét liền trong biểu đồ bên phải biểu diễn ranh giới quyết định của bộ phân loại SVM. Đường thẳng này không chỉ tách biệt hai lớp mà còn giữ khoảng cách xa nhất có thể tới các mẫu huấn luyện gần nhất. Ta có thể hình dung rằng bộ phân loại SVM đang tìm một con đường rộng nhất có thể (biểu diễn bởi các đường nét đứt song song) giữa các lớp, nên việc phân loại còn được gọi là *phân loại biên lớn* (*large margin classification*).



Hình 5.1. Phân loại biên lớn

Lưu ý rằng việc thêm các mẫu huấn luyện khác "nằm ngoài con đường" sẽ không làm ảnh hưởng đến ranh giới quyết định. Lý do là vì ranh giới hoàn toàn được xác định (hoặc được "hỗ trợ – support") bởi các mẫu nằm ở rìa của "con đường". Những mẫu này được gọi là *vector hỗ trợ* (*support vector* – chúng được khoanh tròn trong Hình 5.1).



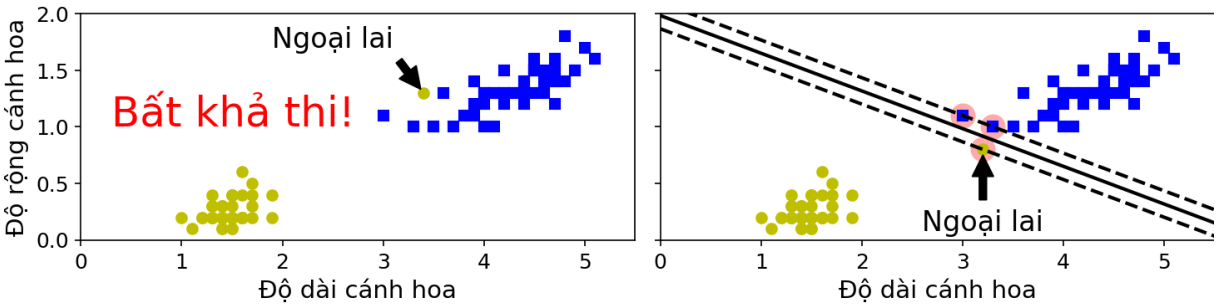
Hình 5.2. Độ nhạy đối với khoảng giá trị của đặc trưng

Lưu ý

Các mô hình SVM khá nhạy cảm với khoảng giá trị của đặc trưng, như có thể thấy trong Hình 5.2: trong đồ thị bên trái, khoảng giá trị theo chiều dọc lớn hơn nhiều so với khoảng giá trị theo chiều ngang, do đó con đường rộng nhất gần như nằm ngang. Sau khi co giãn đặc trưng (ví dụ như sử dụng `StandardScaler` của Scikit-Learn), ranh giới quyết định trong biểu đồ bên phải trông tốt hơn rất nhiều.

Phân loại Biên Mềm

Nếu ta bắt buộc các mẫu phải nằm ngoài con đường và ở đúng phía, thì thuật toán này được gọi là *phân loại biên cứng* (*hard margin classification*). Có hai vấn đề chính với phân loại biên cứng. Thứ nhất, nó chỉ hoạt động nếu dữ liệu có tính tách biệt tuyến tính. Thứ hai, nó nhạy cảm với các điểm ngoại lai. Hình 5.3 biểu diễn tập dữ liệu Iris với chỉ một điểm ngoại lai được thêm vào: ở đồ thị bên trái, việc tìm biên cứng là bất khả thi; ở đồ thị bên phải, ranh giới quyết định rất khác so với đồ thị mà chúng ta đã thấy ở Hình 5.1 khi không có điểm ngoại lai, và khả năng cao là nó sẽ khái quát kém hơn.



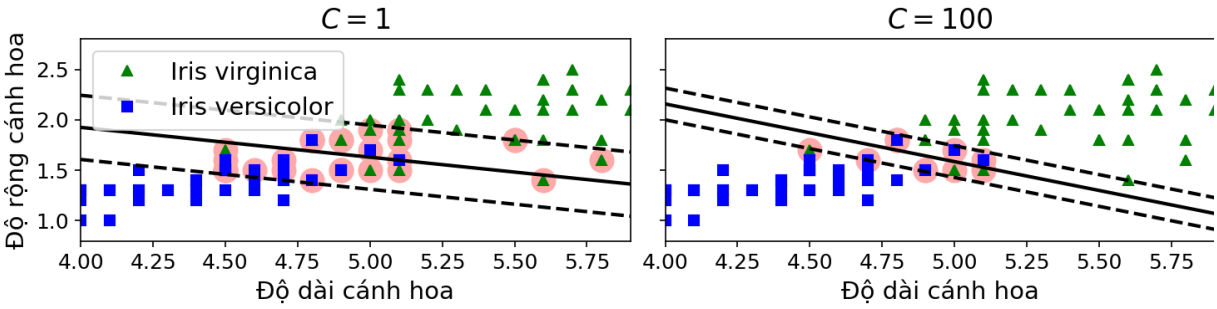
Hình 5.3. Sự nhạy cảm của biên cứng với điểm ngoại lai

Để tránh những vấn đề này, ta sẽ sử dụng một mô hình linh hoạt hơn. Mục tiêu là tìm ra sự cân bằng giữa việc giữ độ rộng của con đường càng lớn càng tốt, nhưng đồng thời hạn chế các

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

vi phạm biên (*margin violation* – tức các mẫu nằm trong con đường, hoặc thậm chí nằm ở sai phía). Thuật toán này được gọi là *phân loại biên mềm* (*soft margin classification*).

Khi khởi tạo mô hình SVM sử dụng Scikit-Learn, ta có thể chỉ định một vài siêu tham số, và `C` là một trong số đó. Nếu sử dụng giá trị `C` nhỏ, ta sẽ thu được mô hình phía bên trái trong Hình 5.4. Còn với một giá trị lớn, ta sẽ thu được mô hình phía bên phải. Thường thì vi phạm biên là điều không tốt và ta không muốn chúng xảy ra nhiều. Tuy nhiên trong trường hợp này, khả năng cao mô hình bên trái sẽ khái quát hóa tốt hơn dù có nhiều vi phạm biên hơn.



Hình 5.4. Biên lớn (trái) và ít vi phạm biên (phải)

Mẹo

Nếu mô hình SVM của bạn quá khớp, bạn có thể thử điều chuẩn nó bằng cách giảm `C`.

Đoạn mã Scikit-Learn dưới đây nạp tập dữ liệu Iris, tiến hành co giãn các đặc trưng, và sau đó huấn luyện một mô hình SVM (sử dụng lớp `LinearSVC` với `C=1` và hàm *mất mát hinge* – được thảo luận trong chương này) để nhận diện hoa *Iris virginica*:

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris virginica

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```

Kết quả là mô hình được biểu diễn ở đồ thị bên trái tại Hình 5.4.

Sau đó, ta có thể sử dụng mô hình này để đưa ra dự đoán như thường lệ:

```
>>> svm_clf.predict([[5.5, 1.7]])
array([1.])
```


Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ghi chú

Khác với bộ phân loại Hồi quy Logistic, bộ phân loại SVM không tính xác suất của mỗi lớp.

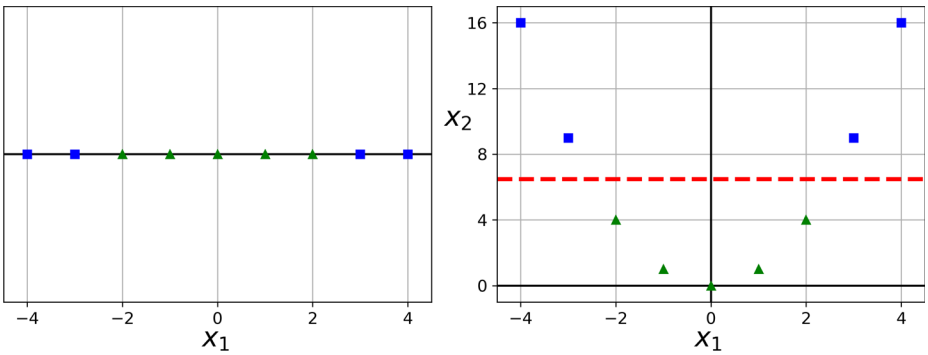
Thay vì sử dụng lớp `LinearSVC`, ta có thể sử dụng lớp `SVC` với hạt nhân tuyến tính được khởi tạo như sau: `SVC(kernel="linear", C=1)`. Mặt khác, ta cũng có thể sử dụng lớp `SGDClassifier`, khởi tạo như sau: `SGDClassifier(loss="hinge", alpha=1/(m*C))`. Lớp này áp dụng phương pháp Hạ Gradient Ngẫu nhiên (tham khảo Chương 4) thông thường để huấn luyện bộ phân loại SVM tuyến tính. Tuy nó không hội tụ nhanh như lớp `LinearSVC`, phương pháp này có thể hữu dụng khi xử lý các tác vụ phân loại trực tuyến hoặc các tập dữ liệu quá lớn so với bộ nhớ (huấn luyện ngoài bộ nhớ chính).

Mẹo

Lớp `LinearSVC` có thực hiện điều chuẩn cho hệ số điều chỉnh, do đó trước tiên bạn nên căn giữa tập huấn luyện bằng cách trừ đi giá trị trung bình. Việc này sẽ được thực hiện tự động nếu bạn co giãn tập dữ liệu bằng `StandardScaler`. Ngoài ra, hãy đảm bảo rằng siêu tham số `loss` được chỉ định là `"hinge"`, bởi nó không phải là giá trị mặc định. Cuối cùng, trừ khi số lượng đặc trưng nhiều hơn số mẫu huấn luyện, bạn nên đặt siêu tham số `dual` là `False` để có được chất lượng tốt hơn (ta sẽ bàn về tính đối ngẫu – *duality* ở phần sau của chương này).

Phân loại SVM Phi Tuyến

Mặc dù bộ phân loại SVM tuyến tính có hiệu năng cao và hoạt động tốt một cách bất ngờ trong nhiều trường hợp, rất nhiều tập dữ liệu lại không có tính tách biệt tuyến tính. Một hướng tiếp cận để xử lý các tập dữ liệu phi tuyến là sử dụng thêm đặc trưng, chẳng hạn như các đặc trưng đa thức (*polynomial features* – như ta đã làm tại Chương 4). Trong một số trường hợp, việc này có thể giúp ta có được một tập dữ liệu tách biệt tuyến tính. Hãy xem xét đồ thị bên trái tại Hình 5.5: nó biểu diễn một tập dữ liệu đơn giản với một đặc trưng duy nhất x_1. Như có thể thấy, tập dữ liệu này không có tính tách biệt tuyến tính. Tuy nhiên nếu ta thêm một đặc trưng thứ hai $x_2 = x_1^2$, tập dữ liệu hai chiều thu được hoàn toàn tách biệt tuyến tính.



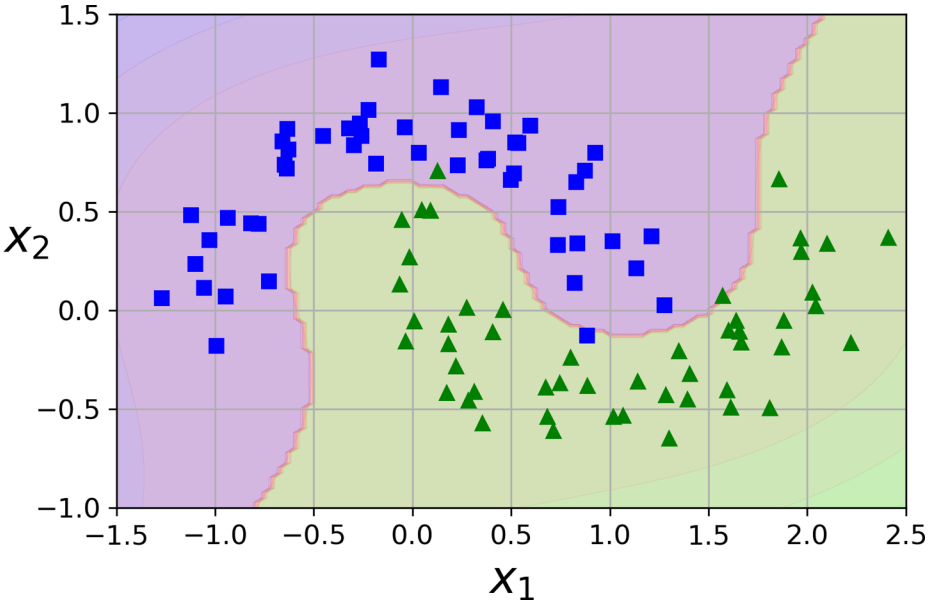
Hình 5.5. Sử dụng thêm đặc trưng để khiến một tập dữ liệu trở nên tách biệt tuyến tính

Để lập trình việc này bằng Scikit-Learn, ta sẽ tạo một `Pipeline` chứa bộ biến đổi `PolynomialFeatures` (đã bàn đến tại Mục 4), theo sau là `StandardScaler` và `LinearSVC`. Hãy thử `Pipeline` này với tập

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

dữ liệu moons: đây là một tập dữ liệu đồ chơi dành cho tác vụ phân loại nhị phân, trong đó các điểm dữ liệu tạo thành hai nửa vòng tròn đan xen nhau (tham khảo Hình 5.6). Ta có thể tạo tập dữ liệu này bằng cách sử dụng hàm `make_moons()`:

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

X, y = make_moons(n_samples=100, noise=0.15)
polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge"))
    ])

polynomial_svm_clf.fit(X, y)
```



Hình 5.6. Bộ phân loại SVM tuyến tính sử dụng các đặc trưng đa thức

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

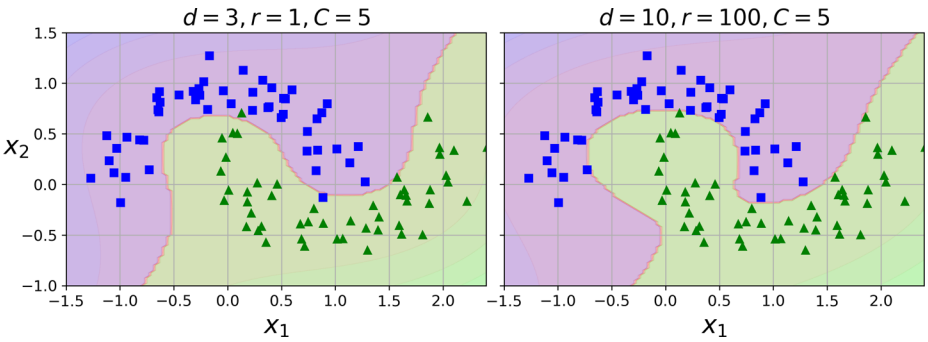
Hạt nhân Đa thức

Việc thêm các đặc trưng đa thức khá dễ để lập trình và có thể hoạt động hiệu quả với tất cả các thuật toán Học Máy (không chỉ với SVM). Tuy nhiên, ở bậc đa thức thấp, phương pháp này không thể xử lý các tập dữ liệu phức tạp. Ngược lại, ở bậc đa thức cao, một lượng rất lớn các đặc trưng được tạo ra sẽ khiến mô hình hoạt động rất chậm.

May mắn thay, khi sử dụng mô hình SVM, ta có thể áp dụng một kỹ thuật toán học khá kỳ diệu, được gọi là *thủ thuật hạt nhân* (*kernel trick*). Thủ thuật hạt nhân giúp ta đạt được kết quả tương tự như khi thêm một số lượng lớn các đặc trưng đa thức, kể cả với các đa thức bậc rất cao, nhưng lại không thực sự thêm bất kỳ đặc trưng nào. Do đó sẽ không có sự bùng nổ tổ hợp ở số lượng các đặc trưng. Thủ thuật này đã được lập trình trong lớp `SVC`. Hãy thử áp dụng nó trên tập dữ liệu moons:

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
```

Đoạn mã trên huấn luyện một bộ phân loại SVM sử dụng hạt nhân đa thức bậc ba. Mô hình này được biểu diễn ở đồ thị bên trái tại Hình 5.7. Đồ thị bên phải là một bộ phân loại SVM khác sử dụng hạt nhân đa thức bậc 10. Tất nhiên nếu mô hình bị quá khớp, ta có thể muốn giảm bậc của đa thức. Ngược lại, nếu mô hình bị dưới khớp, ta có thể thử tăng bậc đa thức lên. Siêu tham số `coef0` kiểm soát mức độ mà mô hình bị ảnh hưởng bởi đa thức bậc cao và đa thức bậc thấp.



Hình 5.7. Bộ phân loại SVM với hạt nhân đa thức

Mẹo

Một hướng tiếp cận phổ biến để tìm siêu tham số phù hợp là sử dụng tìm kiếm dạng lưới (tham khảo Chương 2). Thông thường sẽ nhanh hơn nếu ban đầu ta tìm kiếm dạng lưới với không gian lớn và thưa, sau đó thu hẹp xung quanh các giá trị tốt nhất tìm được. Việc hiểu rõ cơ chế hoạt động của từng siêu tham số cũng có thể giúp bạn định vị khu vực tìm kiếm phù hợp trong không gian siêu tham số.

Đặc trưng Tương tự

Một kỹ thuật khác để giải quyết các bài toán phi tuyến là thêm vào các đặc trưng được tính bằng một *hàm tương tự* (*similarity function*). Hàm này đo lường mức độ giống nhau của mỗi mẫu với một *mốc* (*landmark*) cụ thể. Ví dụ, hãy lấy tập dữ liệu một chiều mà ta đã bàn ở phần trước và thêm vào hai mốc tại $x_1 = -2$ và $x_1 = 1$ (tham khảo đồ thị bên trái của Hình 5.8). Tiếp theo, ta định nghĩa *hàm tương tự Gauss* (*Radial Basis Function* – RBF) với $\gamma = 0.3$ (tham khảo Phương trình 5.1).

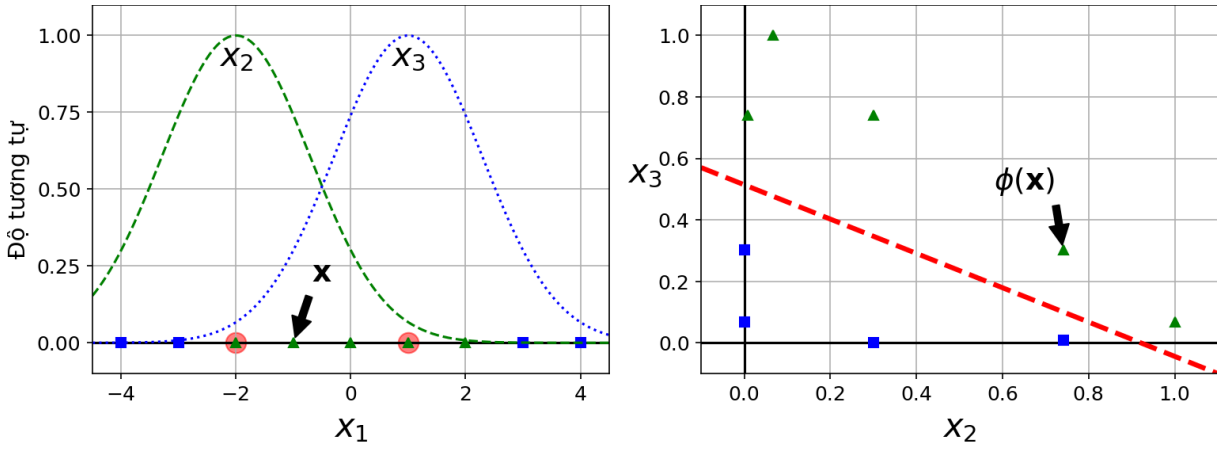
$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma\|\mathbf{x} - \ell\|^2\right)$$

Phương trình 5.1. Hàm tương tự Gauss

Đây là một hàm hình chuông có khoảng giá trị từ 0 (rất xa mốc) tới 1 (tại mốc). Bây giờ ta đã sẵn sàng để tính các đặc trưng mới. Ví dụ, mẫu tại $x_1 = -1$ có khoảng cách tới mốc thứ nhất

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

là 1 và tới mốc thứ hai là 2. Vì vậy đặc trưng mới của nó là $x_2 = \exp\left(-0.3 \times 1^2\right) \approx 0.74$ và $x_3 = \exp\left(-0.3 \times 2^2\right) \approx 0.30$. Đồ thị bên phải của Hình 5.8 biểu diễn tập dữ liệu sau quá trình biến đổi (đã loại bỏ đặc trưng gốc). Như ta có thể thấy, dữ liệu đã trở nên tách biệt tuyến tính.



Hình 5.8. Đặc trưng tương tự sử dụng Gaussian RBF

Bạn có thể sẽ thắc mắc về cách chọn mốc phù hợp. Hướng tiếp cận đơn giản nhất đó là tạo một mốc tại mỗi vị trí của tất cả các mẫu trong tập dữ liệu. Phương pháp này tạo ra nhiều chiều dữ liệu mới, từ đó tăng khả năng tập huấn luyện được biến đổi sẽ tách biệt tuyến tính. Nhược điểm của phương pháp này là một tập huấn luyện với m mẫu và n đặc trưng sẽ được biến đổi thành một tập huấn luyện với m mẫu và m đặc trưng (giả sử ta lược bỏ các đặc trưng gốc). Nếu tập huấn luyện rất lớn, số lượng đặc trưng cũng sẽ rất lớn.

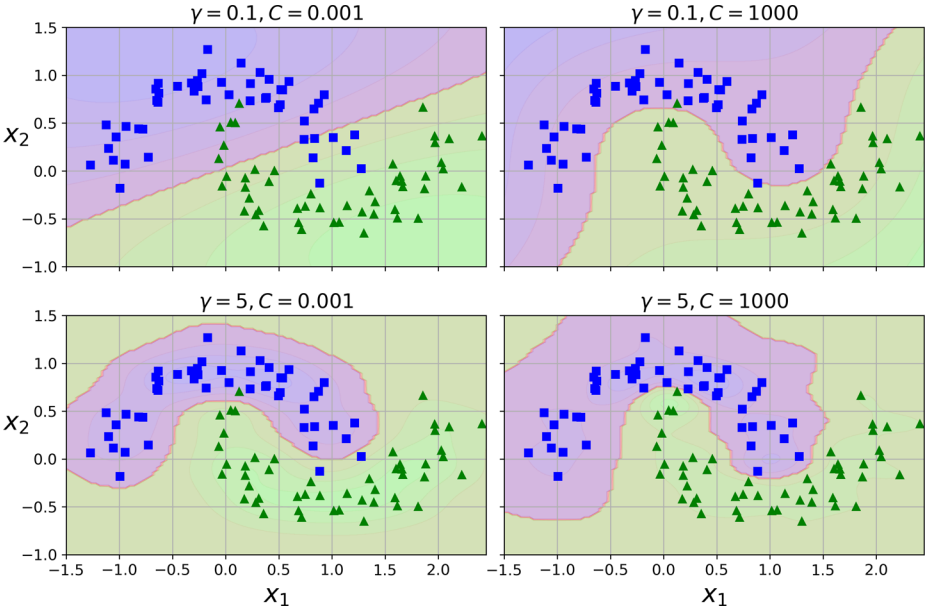
Hạt nhân Gaussian RBF

Cũng giống như phương pháp đặc trưng đa thức, đặc trưng tương tự có thể được sử dụng với bất kỳ thuật toán Học Máy nào, tuy nhiên nó sẽ đòi hỏi khối lượng tính toán khổng lồ để xử lý các đặc trưng thêm vào, đặc biệt là trên các tập dữ liệu lớn. Một lần nữa thủ thuật hạt nhân lại thể hiện sự kỳ diệu, giúp ta đạt được kết quả tương đương mà không cần thực sự thêm vào các đặc trưng tương tự. Hãy thử lớp `SVC` với hạt nhân Gaussian RBF:

```
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
```

Mô hình này được biểu diễn ở đồ thị góc dưới bên trái của Hình 5.9. Các đồ thị còn lại biểu diễn các mô hình được huấn luyện với các siêu tham số `gamma` (γ) và `C` khác nhau. Việc tăng `gamma` khiến đường cong hình chuông hẹp hơn (tham khảo đồ thị bên trái của Hình 5.8). Kết quả là vùng ảnh hưởng của mỗi mẫu trở nên nhỏ hơn: đường ranh giới quyết định trở nên bất thường hơn và dao động xung quanh các mẫu riêng lẻ. Ngược lại, giá trị `gamma` nhỏ làm cho đường cong hình chuông rộng hơn: các mẫu giờ đây có vùng ảnh hưởng lớn hơn, giúp đường ranh giới quyết định trở nên mượt hơn. Do đó γ hoạt động như một tham số điều chuẩn: ta nên giảm γ khi mô hình quá khớp và tăng γ khi mô hình dưới khớp (tương tự như siêu tham số `C`).

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 5.9. Các bộ phân loại SVM sử dụng hạt nhân RBF

Vẫn còn có các hạt nhân khác nhưng chúng hiếm khi được sử dụng. Một vài hạt nhân chỉ có thể được dùng cho một dạng cấu trúc dữ liệu cụ thể. *Hạt nhân Chuỗi* (*String kernel*) đôi khi được dùng để phân loại tài liệu văn bản hoặc chuỗi DNA (ví dụ như *hạt nhân chuỗi con – string subsequence kernel* hoặc các hạt nhân dựa trên *khoảng cách Levenshtein*)

Mẹo

Làm thế nào để lựa chọn giữa nhiều loại hạt nhân như vậy? Thông thường bạn nên thử hạt nhân tuyến tính trước tiên (nhớ rằng `LinearSVC` nhanh hơn nhiều so với `SVC(kernel="linear")`), đặc biệt là khi tập huấn luyện rất lớn và có nhiều đặc trưng. Nếu tập huấn luyện không quá lớn, bạn có thể thử thêm hạt nhân Gaussian RBF vì nó hoạt động khá tốt trong phần lớn các trường hợp. Sau đó, nếu bạn có nhiều thời gian và tài nguyên tính toán, bạn có thể thử nghiệm với các hạt nhân khác bằng phương pháp kiểm định chéo và tìm kiếm dạng lưới, đặc biệt là khi tồn tại các hạt nhân chuyên dụng cho dạng cấu trúc dữ liệu có trong tập huấn luyện của bạn.

Độ phức tạp Tính toán

Lớp `LinearSVC` được lập trình dựa trên thư viện `liblinear`. Thư viện này cài đặt một phiên bản tối ưu của mô hình SVM tuyến tính.[1] Lớp này không hỗ trợ thủ thuật hạt nhân nhưng độ phức tạp tăng gần như tuyến tính với số mẫu huấn luyện và số đặc trưng. Độ phức tạp thời gian của quá trình huấn luyện rơi vào khoảng $O(m \times n)$.

`LinearSVC` sẽ tốn thời gian hơn nếu ta đòi hỏi độ chính xác rất cao. Tuy nhiên, ta có thể kiểm soát điều đó bằng siêu tham số dung sai ε (được gọi là `tol` trong Scikit-Learn). Trong hầu hết các tác vụ phân loại, dung sai mặc định sẽ đem lại kết quả tương đối tốt.

[1] Chih-Jen Lin et al., "A Dual Coordinate Descent Method for Large-Scale Linear SVM," *Proceedings of the 25th International Conference on Machine Learning* (2008): 408–415.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

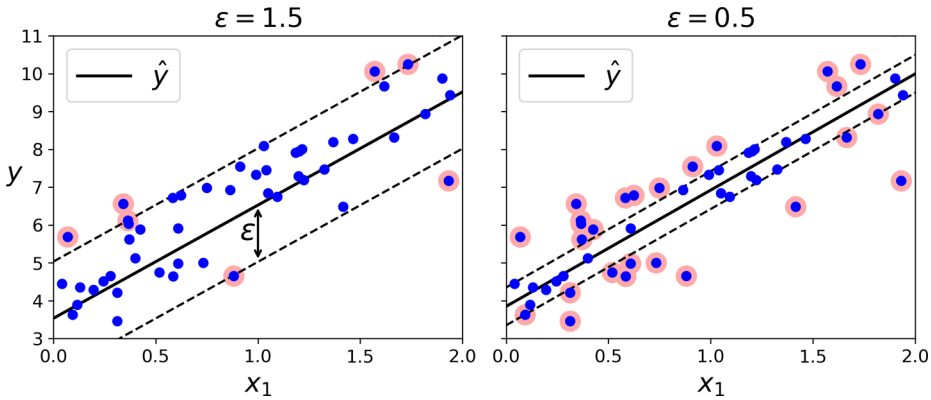
Lớp `SVC` được lập trình dựa trên thư viện `libsvm`. Thư viện này cài đặt thuật toán có hỗ trợ thủ thuật hạt nhân.[2] Độ phức tạp thời gian của quá trình huấn luyện thường nằm trong khoảng $O(m^2 \times n)$ đến $O(m^3 \times n)$. Thật không may, điều này có nghĩa là thuật toán sẽ cực kỳ chậm khi số lượng mẫu huấn luyện lớn (cỡ hàng trăm nghìn mẫu huấn luyện). Thuật toán này phù hợp cho các tập huấn luyện phức tạp có kích thước vừa và nhỏ. Đồng thời, thuật toán cũng mở rộng tốt với số lượng đặc trưng, đặc biệt là với *đặc trưng thưa* (*sparse feature* – tức khi mỗi mẫu chỉ có một vài đặc trưng khác không). Trong trường hợp này, độ phức tạp của thuật toán tỉ lệ xấp xỉ với số lượng đặc trưng khác không trung bình trên các mẫu. Bảng 5.1 so sánh chi tiết các bộ phân loại SVM của Scikit-Learn.

Bảng 5.1. So sánh các bộ phân loại SVM của Scikit-Learn

Lớp	Độ phức tạp thời gian	Hỗ trợ học ngoài bộ nhớ chính	Yêu cầu co giãn	Thủ thuật hạt nhân
`LinearSVC`	$O(m \times n)$	Không	Có	Không
`SGDClassifier`	$O(m \times n)$	Có	Có	Không
`SVC`	$O(m^2 \times n)$ đến $O(m^3 \times n)$	Không	Có	Có

Hồi quy SVM

Như đã đề cập ở phần trước, thuật toán SVM rất linh hoạt: không chỉ dùng được cho tác vụ phân loại mà còn dùng được cho tác vụ hồi quy, cả tuyến tính lẫn phi tuyến. Để sử dụng SVM cho tác vụ hồi quy, mánh khóe nằm ở việc đảo ngược mục tiêu: thay vì cố gắng khớp con đường lớn nhất có thể giữa hai lớp như trong tác vụ phân loại, Hồi quy SVM cố gắng khớp càng nhiều mẫu *nằm trong* con đường càng tốt đồng thời hạn chế việc vi phạm biên (tức hạn chế mẫu *nằm ngoài* con đường). Độ rộng của con đường được kiểm soát bằng siêu tham số ε. Hình 5.10 biểu diễn hai mô hình Hồi quy SVM tuyến tính được huấn luyện trên một vài điểm dữ liệu tuyến tính ngẫu nhiên, một mô hình với biên lớn ($\varepsilon = 1.5$) và mô hình còn lại với biên nhỏ ($\varepsilon = 0.5$).



Hình 5.10. Hồi quy SVM

[2] John Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines" (Microsoft Research technical report, April 21, 1998), *https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-98-14.pdf*.

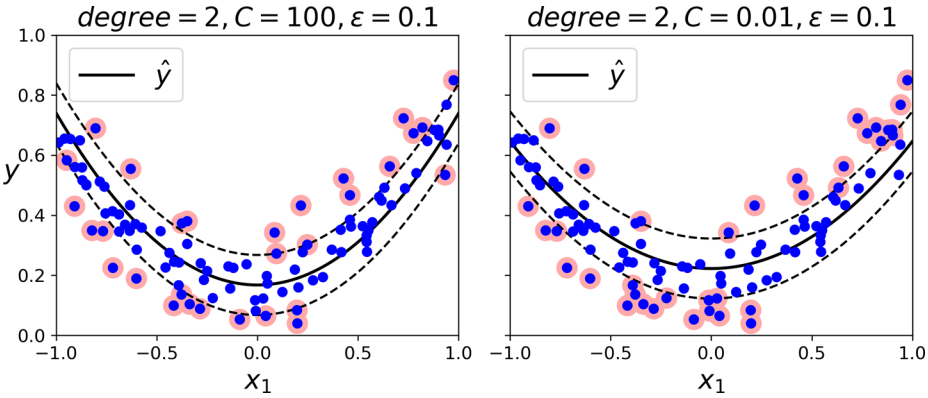
Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Việc thêm mẫu huấn luyện giữa hai biên không ảnh hưởng tới kết quả dự đoán của mô hình nên mô hình được xem là *không nhạy với* ε (*ε-insensitive*).

Ta có thể dùng lớp `LinearSVR` của Scikit-Learn để thực hiện Hồi quy với SVM tuyến tính. Đoạn mã sau huấn luyện mô hình được biểu diễn ở bên trái trong Hình 5.10 (dữ liệu huấn luyện cần được chuẩn hóa trước):

```
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X, y)
```

Để giải quyết các tác vụ hồi quy phi tuyến, ta có thể sử dụng mô hình SVM hạt nhân. Hình 5.11 biểu diễn Hồi quy SVM được huấn luyện trên một tập dữ liệu bậc hai ngẫu nhiên, sử dụng hạt nhân đa thức bậc hai. Mô hình ở đồ thị bên trái được điều chuẩn ít (giá trị `C` lớn) và mô hình ở đồ thị bên phải được điều chuẩn nhiều (giá trị `C` nhỏ).



Hình 5.11. Hồi quy SVM sử dụng hạt nhân đa thức bậc hai

Đoạn mã sau sử dụng lớp `SVR` của Scikit-Learn (có hỗ trợ thủ thuật hạt nhân) để huấn luyện mô hình bên trái trong Hình 5.11:

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X, y)
```

Lớp `SVR` là phiên bản hồi quy của `SVC` và lớp `LinearSVR` là phiên bản hồi quy của `LinearSVC`. Lớp `LinearSVR` có độ phức tạp tăng tuyến tính với kích thước tập huấn luyện (giống với `LinearSVC`) trong khi lớp `SVR` sẽ rất chậm khi tập huấn luyện có kích thước lớn (giống với `SVC`).

Ghi chú

SVM cũng có thể được sử dụng để phát hiện ngoại lai. Hãy tham khảo tài liệu của Scikit-Learn để biết thêm thông tin chi tiết.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Giải thích Mô hình

Phần này sẽ giải thích cách thực hiện dự đoán và huấn luyện với một mô hình SVM, bắt đầu từ các bộ phân loại SVM tuyến tính. Nếu bạn mới bắt đầu với Học Máy, bạn có thể bỏ qua phần này và chuyển thẳng đến các bài tập ở cuối chương, rồi quay lại sau nếu muốn hiểu sâu hơn về SVM.

Đầu tiên, hãy bàn một chút về cách ký hiệu. Ở Chương 4 ta đã sử dụng quy ước đặt tất cả các tham số của mô hình vào một vector $\boldsymbol{\theta}$, bao gồm hệ số điều chỉnh θ_0 và trọng số của các đặc trưng đầu vào $\theta_1, \ldots, \theta_n$, đồng thời thêm đặc trưng đầu vào ứng với hệ số điều chỉnh $x_0 = 1$ cho mọi mẫu. Trong chương này, ta sẽ sử dụng một quy ước khác tiện hơn (và phổ biến hơn) khi làm việc với SVM: ta gọi b là hệ số điều chỉnh và gọi $\mathbf{w}$ là vector trọng số đặc trưng. Vector đặc trưng đầu vào sẽ không bao gồm đặc trưng của hệ số điều chỉnh.

Hàm Quyết định và Dự đoán

Mô hình phân loại SVM dự đoán lớp của một mẫu $\mathbf{x}$ mới bằng cách tính hàm quyết định $\mathbf{w}^\top \mathbf{x} + b = w_1 x_1 + \cdots + w_n x_n + b$. Nếu kết quả dương, lớp dự đoán $\hat{y}$ sẽ là lớp dương (1), ngược lại sẽ là lớp âm (0); tham khảo Phương trình 5.2.

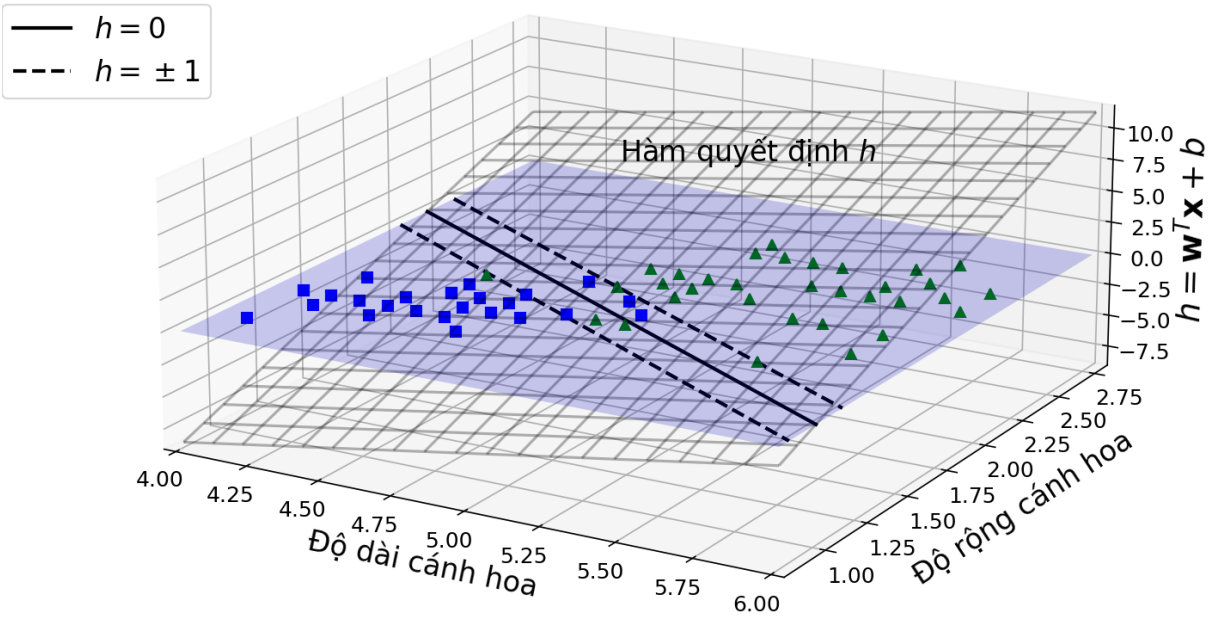
$$\hat{y} = \begin{cases} 0 & \text{nếu } \mathbf{w}^\top \mathbf{x} + b < 0, \\ 1 & \text{nếu } \mathbf{w}^\top \mathbf{x} + b \geq 0 \end{cases}$$

Phương trình 5.2. Dự đoán của bộ phân loại SVM tuyến tính

Hình 5.12 minh hoạ hàm quyết định của mô hình phía bên trái ở Hình 5.4: hàm này được biểu diễn bởi một mặt phẳng hai chiều vì tập dữ liệu này có hai đặc trưng (chiều rộng và chiều dài của cánh hoa). Ranh giới quyết định là tập hợp các điểm mà tại đó hàm quyết định có giá trị bằng 0: đó là đường thẳng giao nhau của hai mặt phẳng (được biểu diễn bằng đường nét liền in đậm).[3]

[3] Tổng quát hơn, với n đặc trưng, hàm quyết định là một *siêu mặt phẳng* (*hyperplane*) n chiều, và ranh giới quyết định là một siêu mặt phẳng (n–1) chiều.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

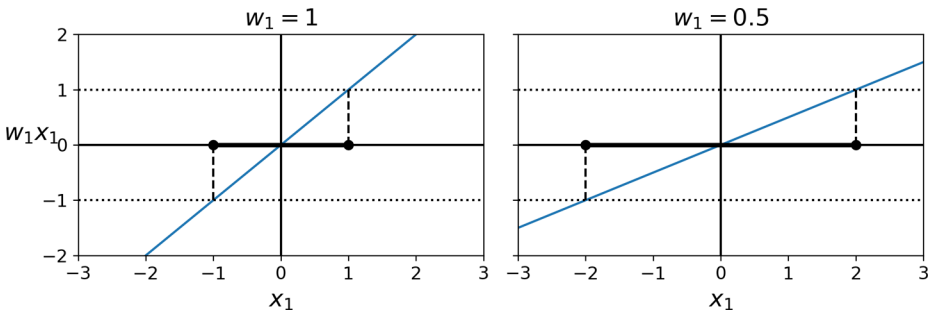


Hình 5.12. Hàm quyết định cho tập dữ liệu Iris

Các đường nét đứt biểu diễn các điểm tại đó hàm quyết định có giá trị bằng 1 hay –1: các đường thẳng này song song và cách ranh giới quyết định một khoảng bằng nhau, tạo thành biên quanh đường ranh giới. Việc huấn luyện một bộ phân loại SVM tuyến tính có nghĩa là tìm các giá trị $\mathbf{w}$ và b sao cho biên rộng nhất có thể, đồng thời tránh vi phạm biên (đối với biên cứng) hoặc giới hạn việc vi phạm biên (đối với biên mềm).

Hàm Mục tiêu

Ta thấy độ dốc của hàm quyết định có giá trị bằng chuẩn của vector trọng số, $\|\mathbf{w}\|$. Nếu giảm một nửa độ dốc, khoảng cách từ các điểm mà tại đó hàm quyết định có giá trị bằng ± 1 tới ranh giới quyết định sẽ tăng lên hai lần. Nói cách khác, việc chia độ dốc cho 2 sẽ khiến độ rộng biên tăng lên gấp đôi. Ta có thể dễ dàng biểu diễn tính chất này trên mặt phẳng hai chiều, như được minh hoạ trong Hình 5.13. Vector trọng số $\mathbf{w}$ có giá trị càng nhỏ, độ rộng biên sẽ càng lớn.



Hình 5.13. Vector trọng số có giá trị nhỏ dẫn đến biên lớn

Vì vậy, ta muốn cực tiểu hóa $\|\mathbf{w}\|$ để tìm được biên có độ rộng lớn. Nếu muốn tránh hoàn toàn vi phạm biên (biên cứng), hàm quyết định cần có giá trị lớn hơn 1 với tất cả các mẫu huấn luyện dương và có giá trị nhỏ hơn –1 với các mẫu huấn luyện âm. Nếu ta định nghĩa $t^{(i)} = -1$

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

cho các mẫu âm (nếu $y^{(i)} = 0$) và $t^{(i)} = 1$ cho các mẫu dương (nếu $y^{(i)} = 1$), khi đó điều kiện ràng buộc được biểu diễn là $t^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1$ cho tất cả các mẫu.

Từ đó, ta có thể biểu diễn hàm mục tiêu cho bộ phân loại SVM tuyến tính biên cứng dưới dạng một bài toán tối ưu có điều kiện (Phương trình 5.3).

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} \\ \text{sao cho} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 \quad \text{với } i = 1, 2, \cdots, m \end{aligned}$$

Phương trình 5.3. Hàm mục tiêu cho bộ phân loại SVM tuyến tính biên cứng

Ghi chú

Ta đang cực tiểu hóa $\frac{1}{2}\mathbf{w}^\top \mathbf{w}$, tức $\frac{1}{2}\|\mathbf{w}\|^2$, thay vì cực tiểu hóa $\|\mathbf{w}\|$. Lý do là bởi $\frac{1}{2}\|\mathbf{w}\|^2$ có đạo hàm khá đẹp và đơn giản (đó là $\mathbf{w}$), trong khi $\|\mathbf{w}\|$ không khả vi tại $\mathbf{w} = 0$. Các thuật toán tối ưu sẽ hoạt động hiệu quả hơn với những hàm khả vi.

Để thu được hàm mục tiêu cho biên mềm, ta cần thêm *biến bù* (*slack variable*) $\zeta^{(i)} \geq 0$ cho mỗi mẫu:[4] $\zeta^{(i)}$ đo lường mức độ vi phạm cho phép của mẫu thứ i. Giờ ta có hai hàm mục tiêu mâu thuẫn nhau: một hàm khiến các biến bù càng nhỏ càng tốt để giảm vi phạm biên, một hàm khiến $\frac{1}{2}\mathbf{w}^\top \mathbf{w}$ càng nhỏ càng tốt để tăng độ rộng biên. Đây là lúc ta cần sử dụng siêu tham số C để điều chỉnh sự cân bằng giữa hai hàm mục tiêu này. Từ đó, ta có bài toán tối ưu có điều kiện trong Phương trình 5.4.

$$\begin{aligned} \underset{\mathbf{w},b,\zeta}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{m} \zeta^{(i)} \\ \text{sao cho} \quad & t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) \geq 1 - \zeta^{(i)} \quad \text{và} \quad \zeta^{(i)} \geq 0 \quad \text{với } i = 1, 2, \cdots, m \end{aligned}$$

Phương trình 5.4. Hàm mục tiêu cho bộ phân loại SVM tuyến tính biên mềm

Quy hoạch Toàn phương

Bài toán biên cứng và biên mềm đều là bài toán tối ưu lồi bậc hai với ràng buộc tuyến tính. Nó còn được gọi là bài toán *Quy hoạch Toàn phương* (*Quadratic Programming* – QP). Nhiều bộ giải có sẵn giải quyết các bài toán quy hoạch toàn phương bằng cách sử dụng những kỹ thuật khác nhau, nhưng chúng nằm ngoài phạm vi của cuốn sách này.[5]

Công thức tổng quát của bài toán được biểu diễn trong Phương trình 5.5.

Lưu ý rằng biểu thức $\mathbf{Ap} \leq \mathbf{b}$ biểu diễn n_c ràng buộc: $\mathbf{p}^\top \mathbf{a}^{(i)} \leq b^{(i)}$ với $i = 1, 2, \cdots, n_c$, trong đó $\mathbf{a}^{(i)}$ là vector chứa các phần tử nằm ở hàng thứ i của ma trận $\mathbf{A}$ và $b^{(i)}$ là phần tử thứ i của $\mathbf{b}$.

Ta có thể dễ dàng chứng minh rằng nếu tham số của quy hoạch toàn phương được thiết lập như sau, ta sẽ thu được hàm mục tiêu của bộ phân loại SVM tuyến tính biên cứng:

[4] Zeta (ζ) là ký tự thứ 6 trong bảng chữ cái Hy Lạp.

[5] Để tìm hiểu thêm về bài toán quy hoạch toàn phương, bạn có thể tham khảo cuốn sách *Tối ưu Lồi* (*Convex Optimization*) của Stephen Boyd và Lieven Vandenberghe (Cambridge University Press, 2004) hoặc xem các video bài giảng của Richard Brown.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$
\begin{aligned}
\underset{\mathbf{p}}{\text{Minimize}} \quad & \frac{1}{2}\mathbf{p}^\top \mathbf{H}\mathbf{p} \quad + \quad \mathbf{f}^\top \mathbf{p} \\
\text{sao cho} \quad & \mathbf{A}\mathbf{p} \leq \mathbf{b} \\
\text{trong đó} \quad & \begin{cases}
\mathbf{p} & \text{là vector } n_p \text{ chiều } (n_p = \text{số lượng tham số}), \\
\mathbf{H} & \text{là ma trận } n_p \times n_p, \\
\mathbf{f} & \text{là vector } n_p \text{ chiều}, \\
\mathbf{A} & \text{là ma trận } n_c \times n_p \quad (n_c = \text{số lượng ràng buộc}), \\
\mathbf{b} & \text{là vector } n_c \text{ chiều.}
\end{cases}
\end{aligned}
$$

Phương trình 5.5. Bài toán Quy hoạch Toàn phương

- $n_p = n + 1$, trong đó n là số lượng đặc trưng (phần +1 là cho hệ số điều chỉnh).
- $n_c = m$, trong đó m là số lượng mẫu huấn luyện.
- $\mathbf{H}$ là ma trận đơn vị $n_p \times n_p$, ngoại trừ ô trên cùng bên trái có giá trị bằng 0 (để bỏ qua hệ số điều chỉnh).
- $\mathbf{f} = 0$, là vector n_p chiều có tất cả các giá trị bằng 0.
- $\mathbf{b} = -1$, là vector n_c chiều có tất cả các giá trị bằng –1.
- $\mathbf{a}^{(i)} = -t^{(i)}\dot{\mathbf{x}}^{(i)}$, trong đó $\dot{\mathbf{x}}^{(i)}$ là $\mathbf{x}^{(i)}$ có thêm đặc trưng ứng với hệ số điều chỉnh $\dot{\mathbf{x}}_0 = 1$.

Một cách để huấn luyện bộ phân loại SVM tuyến tính biên cứng là sử dụng bộ giải quy hoạch toàn phương có sẵn và truyền cho nó các tham số nêu trên. Vector $\mathbf{p}$ thu được sẽ chứa hệ số điều chỉnh $b = p_0$ và trọng số đặc trưng $w_i = p_i$ với $i = 1, 2, \cdots, n$. Tương tự, ta cũng có thể sử dụng các bộ giải quy hoạch toàn phương để giải bài toán biên mềm (tham khảo bài tập cuối chương).

Để sử dụng thủ thuật hạt nhân, chúng ta sẽ xem xét một bài toán tối ưu hóa có điều kiện khác.

Bài toán Đối ngẫu

Với bài toán tối ưu có điều kiện, còn được gọi là *bài toán gốc* (*primal problem*), ta có thể định nghĩa một bài toán có liên quan chặt chẽ khác, đó là *bài toán đối ngẫu* (*dual problem*). Nghiệm của bài toán đối ngẫu thường cho biết cận dưới của nghiệm của bài toán gốc, nhưng với một số điều kiện, nó có thể có cùng nghiệm với bài toán gốc. May thay, bài toán SVM lại đáp ứng các điều kiện này,[6] nên việc giải bài toán gốc hay bài toán đối ngẫu sẽ đều dẫn đến cùng một nghiệm. Phương trình 5.6 biểu diễn dạng đối ngẫu của hàm mục tiêu của SVM tuyến tính (nếu bạn muốn biết nghiệm của bài toán đối ngẫu được suy ra từ bài toán gốc như thế nào, hãy tham khảo Phụ lục C).

Khi đã tìm được vector $\hat{\alpha}$ để biểu thức này đạt cực tiểu (bằng bộ giải quy hoạch toàn phương), ta sử dụng Phương trình 5.7 để tính $\hat{\mathbf{w}}$ và $\hat{b}$ sao cho bài toán gốc đạt cực tiểu.

Bài toán đối ngẫu được giải nhanh hơn bài toán gốc khi số lượng mẫu huấn luyện nhỏ hơn số lượng đặc trưng. Quan trọng hơn, thủ thuật hạt nhân có thể được áp dụng trong bài toán đối ngẫu, nhưng trong bài toán gốc thì không. Vậy thủ thuật hạt nhân thực sự là gì?

[6] Hàm mục tiêu là hàm lồi và các bất đẳng thức ràng buộc là hàm lồi khả vi liên tục.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\top}\mathbf{x}^{(j)} \quad - \quad \sum_{i=1}^{m}\alpha^{(i)}$$
$$\text{sao cho} \quad \alpha^{(i)} \geq 0 \quad \text{với } i = 1, 2, \cdots, m$$

Phương trình 5.6. Dạng đối ngẫu của hàm mục tiêu của SVM tuyến tính

$$\hat{\mathbf{w}} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\mathbf{x}^{(i)}$$
$$\hat{b} = \frac{1}{n_s}\sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \hat{\mathbf{w}}^\top\mathbf{x}^{(i)}\right)$$

Phương trình 5.7. Từ bài toán đối ngẫu về bài toán gốc

SVM Hạt nhân

Giả sử ta muốn áp dụng phép biến đổi đa thức bậc hai cho tập huấn luyện hai chiều (chẳng hạn như tập huấn luyện moons), sau đó huấn luyện bộ phân loại SVM tuyến tính trên tập huấn luyện đã biến đổi. Phương trình 5.8 biểu diễn hàm ánh xạ đa thức bậc hai ϕ mà ta muốn áp dụng.

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} {x_1}^2 \\ \sqrt{2}x_1x_2 \\ {x_2}^2 \end{pmatrix}$$

Phương trình 5.8. Ánh xạ đa thức bậc hai

Lưu ý rằng vector đã được biến đổi từ hai chiều thành ba chiều. Giờ hãy xem điều gì sẽ xảy ra với hai vector hai chiều **a** và **b** nếu ta áp dụng ánh xạ đa thức bậc hai này và sau đó tính tích vô hướng[7] của các vector được biến đổi (Phương trình 5.9).

$$\begin{aligned}\phi(\mathbf{a})^\top\phi(\mathbf{b}) &= \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\,a_1a_2 \\ {a_2}^2 \end{pmatrix}^\top \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\,b_1b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2{b_1}^2 + 2a_1b_1a_2b_2 + {a_2}^2{b_2}^2 \\ &= (a_1b_1 + a_2b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^\top \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = \left(\mathbf{a}^\top\mathbf{b}\right)^2\end{aligned}$$

Phương trình 5.9. Thủ thuật hạt nhân với phép ánh xạ đa thức bậc hai

Ta thấy rằng tích vô hướng của hai vector đã được biến đổi có giá trị bằng bình phương tích vô

[7] Như đã giải thích trong Chương 4, tích vô hướng của hai vector **a** và **b** thường được ký hiệu là $\mathbf{a}\cdot\mathbf{b}$. Tuy nhiên, trong Học Máy, vector thường được biểu diễn dưới dạng vector cột (tức ma trận có một cột). Do đó, ta có thể tính tích vô hướng bằng cách tính $\mathbf{a}^\top\mathbf{b}$. Để đảm bảo tính nhất quán trong cuốn sách, chúng tôi sẽ sử dụng cách ký hiệu này, bỏ qua việc thực chất cách tính này sẽ cho kết quả là ma trận có một phần tử thay vì một giá trị vô hướng.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

hướng của hai vector ban đầu: $\phi(\mathbf{a})^\top \phi(\mathbf{b}) = \left(\mathbf{a}^\top \mathbf{b}\right)^2$.

Đây là ý tưởng quan trọng: nếu ta áp dụng phép biến đổi ϕ cho tất cả các mẫu huấn luyện, thì bài toán đối ngẫu (Phương trình 5.6) sẽ chứa tích vô hướng $\phi\left(\mathbf{x}^{(i)}\right)^\top \phi\left(\mathbf{x}^{(j)}\right)$. Nhưng nếu ϕ là phép biến đổi đa thức bậc hai được định nghĩa trong Phương trình 5.8, ta có thể thay thế tích vô hướng của hai vector đã được biến đổi bằng $\left(\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}\right)^2$. Vì vậy, ta không cần phải biến đổi các mẫu huấn luyện, mà chỉ cần thay thế tích vô hướng bằng bình phương giá trị của nó trong Phương trình 5.6. Kết quả sẽ hoàn toàn giống như khi ta biến đổi tập huấn luyện sau đó khớp chúng vào thuật toán SVM tuyến tính, nhưng thủ thuật này giúp cho toàn bộ quá trình tính toán hiệu quả hơn rất nhiều.

Hàm $K(\mathbf{a}, \mathbf{b}) = \left(\mathbf{a}^\top \mathbf{b}\right)^2$ là một hạt nhân đa thức bậc hai. Trong Học Máy, *hạt nhân* (*kernel*) là hàm có khả năng tính tích vô hướng $\phi(\mathbf{a})^\top \phi(\mathbf{b})$, chỉ dựa trên hai vector $\mathbf{a}$ và $\mathbf{b}$ ban đầu, mà không cần tính (hay biết) phép biến đổi ϕ. Phương trình 5.10 liệt kê một số hạt nhân được sử dụng phổ biến.

Định lý Mercer

Theo *định lý Mercer*, nếu hàm $K(\mathbf{a}, \mathbf{b})$ tuân theo một số điều kiện toán học có tên là *điều kiện Mercer* (ví dụ, K phải là hàm liên tục và đối xứng theo các đối số của nó sao cho $K(\mathbf{a}, \mathbf{b}) = K(\mathbf{b}, \mathbf{a})$, v.v.), thì tồn tại hàm ϕ ánh xạ $\mathbf{a}$ và $\mathbf{b}$ sang không gian khác (thường có nhiều chiều hơn) sao cho $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^\top \phi(\mathbf{b})$. Ta có thể sử dụng K như một hạt nhân vì ta biết ϕ có tồn tại, mặc dù ta không biết ϕ cụ thể là gì. Trong trường hợp hạt nhân RBF Gauss, ta có thể thấy ϕ ánh xạ từng mẫu huấn luyện sang một không gian vô hạn chiều, vì thế thật may mắn khi ta không cần trực tiếp thực hiện phép ánh xạ!

Lưu ý rằng một số hạt nhân hay được sử dụng (ví dụ hạt nhân sigmoid) không tuân theo tất cả các điều kiện Mercer, nhưng chúng lại hoạt động tương đối hiệu quả trong thực tế.

$$
\begin{aligned}
\text{Tuyến tính:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^\top \mathbf{b} \\
\text{Đa thức:} \quad & K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^\top \mathbf{b} + r\right)^d \\
\text{Gaussian RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \mathbf{a} - \mathbf{b}^2\right) \\
\text{Sigmoid:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^\top \mathbf{b} + r\right)
\end{aligned}
$$

Phương trình 5.10. Một số hạt nhân phổ biến

Vẫn còn một điểm mà ta cần phải đề cập thêm. Phương trình 5.7 biểu diễn cách suy ra nghiệm cho bài toán gốc từ nghiệm của bài toán đối ngẫu trong trường hợp bộ phân loại SVM tuyến tính. Nhưng nếu áp dụng thủ thuật hạt nhân, ta cần thay thế các $x^{(i)}$ bằng $\phi\left(x^{(i)}\right)$. Thực tế, $\hat{\mathbf{w}}$ sẽ có cùng số chiều với $\phi\left(x^{(i)}\right)$. Vì số chiều này có thể rất lớn hay thậm chí vô hạn nên ta không thể tính được $\hat{\mathbf{w}}$. Vậy làm sao để đưa ra dự đoán khi ta không biết $\hat{\mathbf{w}}$? Tin tốt là ta có thể áp dụng công thức của $\hat{\mathbf{w}}$ từ Phương trình 5.7 vào hàm quyết định cho một mẫu mới $\mathbf{x}^{(n)}$, và ta sẽ thu được biểu thức là tích vô hướng giữa các vector đầu vào. Điều này cho phép ta áp dụng được thủ thuật hạt nhân (Phương trình 5.11).

Lưu ý rằng vì chỉ các vector hỗ trợ có $\alpha^{(i)} \neq 0$, việc đưa ra dự đoán chỉ yêu cầu tích vô hướng của vector đầu vào mới $\mathbf{x}^{(n)}$ với các vector hỗ trợ, thay vì tất cả các mẫu huấn luyện. Tất nhiên, ta cũng cần sử dụng thủ thuật tương tự để tính hệ số điều chỉnh $\hat{b}$ (Phương trình 5.12).

Nếu bạn bắt đầu thấy khó hiểu, điều này hoàn toàn bình thường: đây là một "tác dụng phụ" của thủ thuật hạt nhân.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$
\begin{aligned}
h_{\hat{\mathbf{w}},\hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) &= \hat{\mathbf{w}}^\top \phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^\top \phi\left(\mathbf{x}^{(n)}\right) + \hat{b} \\
&= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^\top \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b} \\
&= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}) + \hat{b}
\end{aligned}
$$

Phương trình 5.11. Đưa ra dự đoán với SVM hạt nhân

$$
\begin{aligned}
\hat{b} &= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^\top \phi\left(\mathbf{x}^{(i)}\right)\right) = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi\left(\mathbf{x}^{(j)}\right)\right)^\top \phi\left(\mathbf{x}^{(i)}\right)\right) \\
&= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \left(t^{(i)} - \sum_{\substack{j=1 \\ \hat{\alpha}^{(j)}>0}}^{m} \hat{\alpha}^{(j)} t^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\right)
\end{aligned}
$$

Phương trình 5.12. Sử dụng thủ thuật hạt nhân để tính hệ số điều chỉnh

Bộ Phân loại SVM Trực tuyến

Trước khi tổng kết chương này, ta sẽ điểm nhanh qua bộ phân loại SVM trực tuyến (nhắc lại rằng học trực tuyến có nghĩa là học tăng dần, thường là khi ta có được các mẫu mới).

Với bộ phân loại SVM tuyến tính, một phương pháp để lập trình bộ phân loại SVM trực tuyến là sử dụng phương pháp Hạ Gradient (ví dụ như sử dụng `SGDClassifer`) để cực tiểu hóa hàm chi phí suy được từ bài toán gốc trong Phương trình 5.13. Thật không may, phương pháp Hạ Gradient sẽ hội tụ chậm hơn nhiều so với các phương pháp dựa trên quy hoạch toàn phương.

$$
J(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} \; + \; C \sum_{i=1}^{m} \max\left(0,\; 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\right)
$$

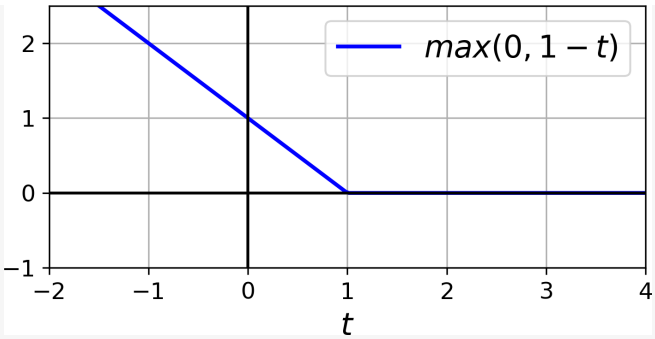
Phương trình 5.13. Hàm chi phí của bộ phân loại SVM tuyến tính

Số hạng đầu tiên trong hàm chi phí sẽ khiến mô hình có vector trọng số $\mathbf{w}$ nhỏ, dẫn đến biên lớn hơn. Số hạng thứ hai tính tổng tất cả các vi phạm biên. Vi phạm biên của một mẫu bằng 0 nếu nó nằm ngoài biên và đúng phía, nếu không nó sẽ tỷ lệ thuận với khoảng cách đến đường biên ở đúng phía. Việc cực tiểu hóa số hạng này đảm bảo rằng mô hình sẽ vi phạm biên ít nhất và với khoảng cách vi phạm nhỏ nhất có thể.

Mất mát Hinge

Hàm $\max(0,\; 1 - t)$ được gọi là hàm *mất mát hinge* (xem ảnh dưới đây). Nó có giá trị bằng 0 khi $t \geq 1$. Đạo hàm (độ dốc) của hàm này bằng –1 nếu $t < 1$ và 0 nếu $t > 1$. Hàm này không khả vi tại $t = 1$, nhưng giống với Hồi quy Lasso (tham khảo Mục 4), ta vẫn có thể sử dụng Hạ Gradient với bất kỳ *đạo hàm dưới* (*subderivative*) nào tại $t = 1$, (tức bất kỳ giá trị nào trong khoảng –1 và 0).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Ta cũng có thể lập trình được SVM hạt nhân trực tuyến, như được trình bày trong các bài báo "Incremental and Decremental Support Vector Machine Learning"[8] và "Fast Kernel Classifiers with Online and Active Learning".[9] Bộ phân loại SVM hạt nhân được lập trình bằng Matlab và C++. Đối với các bài toán phi tuyến quy mô lớn, bạn nên cân nhắc sử dụng mạng nơ-ron (sẽ được đề cập trong Tập 2).

Bài tập

1. Ý tưởng căn bản của Máy Vector Hỗ trợ là gì?
2. Vector hỗ trợ là gì?
3. Tại sao việc co giãn giá trị đầu vào lại quan trọng khi sử dụng SVM?
4. Bộ phân loại SVM có thể trả về điểm số tin cậy và xác suất khi phân loại một mẫu không?
5. Bạn nên sử dụng biểu diễn dạng gốc hay đối ngẫu cho bài toán SVM để huấn luyện mô hình trên tập dữ liệu huấn luyện có hàng triệu mẫu và hàng trăm đặc trưng?
6. Giả sử bạn vừa huấn luyện một bộ phân loại SVM với hạt nhân RBF, nhưng mô hình có vẻ dưới khớp tập huấn luyện. Bạn nên tăng hay giảm các tham số γ (`gamma`) và `C`?
7. Nên thiết lập các tham số quy hoạch toàn phương ($\mathbf{H}$, $\mathbf{f}$, $\mathbf{A}$, và $\mathbf{b}$) như thế nào để giải bài toán cho bộ phân loại SVM biên mềm với bộ giải quy hoạch toàn phương có sẵn?
8. Hãy huấn luyện mô hình `LinearSVC` trên một tập dữ liệu tách biệt tuyến tính. Sau đó huấn luyện mô hình `SVC` và `SGDClassifier` trên cùng tập dữ liệu đó. Hãy thử làm cho các mô hình thu được gần giống nhau.
9. Hãy huấn luyện bộ phân loại SVM trên tập dữ liệu MNIST. Do SVM là bộ phân loại nhị phân, bạn sẽ cần sử dụng phương pháp một-còn lại để phân loại 10 chữ số. Bạn có thể sẽ cần tinh chỉnh siêu tham số với các tập kiểm định nhỏ để tăng tốc quá trình này. Độ chính xác cao nhất bạn có thể đạt được là bao nhiêu?
10. Hãy huấn luyện bộ hồi quy SVM trên tập dữ liệu nhà ở California.

Lời giải cho các bài tập trên nằm ở Phụ lục A.

[8] Gert Cauwenberghs và Tomaso Poggio, "Incremental and Decremental Support Vector Machine Learning," *Proceedings of the 13th International Conference on Neural Information Processing Systems* (2000): 388–394.

[9] Antoine Bordes và cộng sự, "Fast Kernel Classifiers with Online and Active Learning," *Journal of Machine Learning Research* 6 (2005): 1579–1619.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 6

Cây Quyết định

Giống với SVM, *Cây Quyết định* (*Decision Tree*) cũng là thuật toán Học Máy có thể thực hiện cả hai tác vụ phân loại và hồi quy, thậm chí là cả các tác vụ đa đầu ra. Thuật toán này rất mạnh mẽ và có khả năng khớp những tập dữ liệu phức tạp. Ví dụ như ở Chương 2, ta đã khớp một mô hình `DecisionTreeRegressor` trên tập dữ liệu giá nhà California tới mức hoàn hảo (đúng hơn là quá khớp).

Cây Quyết định cũng là thành phần nền tảng của Rừng Ngẫu nhiên (tham khảo Chương 7), một trong những thuật toán học máy mạnh mẽ nhất hiện nay.

Ta sẽ bắt đầu chương này bằng việc thảo luận về cách huấn luyện, biểu diễn và dự đoán với Cây Quyết định. Sau đó ta sẽ bàn về thuật toán huấn luyện CART trong Scikit-Learn, cách điều chuẩn cây và cách sử dụng chúng cho tác vụ hồi quy. Cuối cùng, ta sẽ thảo luận về những điểm hạn chế của Cây Quyết định.

Huấn luyện và Biểu diễn một Cây Quyết định

Để hiểu rõ về Cây Quyết định, hãy cùng xây dựng một mô hình và xem cách nó đưa ra dự đoán. Đoạn mã dưới đây huấn luyện một `DecisionTreeClassifier` trên tập dữ liệu Iris (tham khảo Chương 4):

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

Ta có thể minh họa một Cây Quyết định đã được huấn luyện bằng cách sử dụng hàm `export_graphviz()` để xuất ra một tệp định nghĩa đồ thị có tên là *iris_tree.dot*:

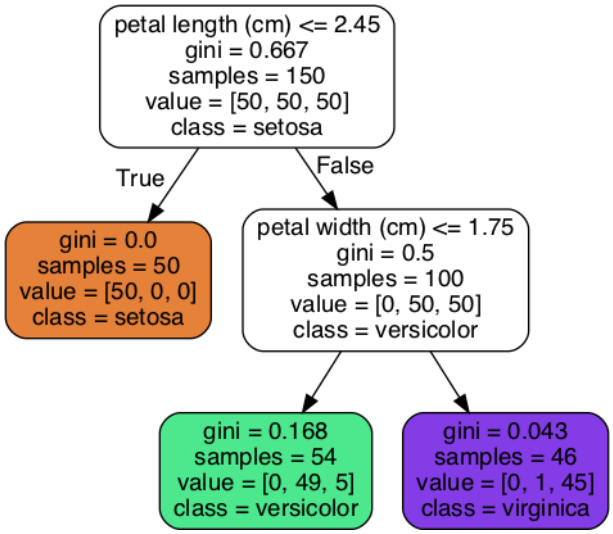
```
from sklearn.tree import export_graphviz

export_graphviz(
```

```
        tree_clf,
        out_file=image_path("iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

Sau đó, ta sử dụng lệnh `dot` từ gói Graphviz trên cửa sổ dòng lệnh để chuyển đổi tệp tin *.dot* này sang nhiều định dạng khác, chẳng hạn như PDF hoặc PNG.[1] Câu lệnh sau chuyển đổi tệp *.dot* thành một tệp ảnh *.png*:

```
$ dot -Tpng iris_tree.dot -o iris_tree.png
```

Cây Quyết định đầu tiên của ta được biểu diễn trong Hình 6.1.



Hình 6.1. Cây Quyết định Iris

Dự đoán

Hãy cùng xem cách mà cây Quyết định trong Hình 6.1 đưa ra dự đoán. Giả sử, ta cần phân loại một bông hoa Iris. Ta bắt đầu từ *nút gốc* (độ sâu bằng 0, ở trên cùng): nút này kiểm tra liệu chiều dài của cánh hoa có nhỏ hơn 2.45 cm hay không. Nếu nhỏ hơn thì ta đi xuống nút con trái của nút gốc (độ sâu 1, bên trái). Trong trường hợp này nó là một *nút lá* (tức nó không có nút con), nên không kiểm tra thêm điều kiện nào: chỉ cần nhìn vào lớp dự đoán của nút đó, ta sẽ thấy Cây Quyết định dự đoán bông hoa thuộc loại *Iris setosa* (`class=setosa`).

Tiếp tục giả sử rằng ta có một bông hoa khác, và lần này thì cánh hoa dài hơn 2.45 cm. Ta phải đi xuống nút con phải của nút gốc (độ sâu 1, bên phải). Nút này không phải là nút lá nên nó sẽ kiểm tra thêm một điều kiện nữa: chiều rộng của cánh hoa có nhỏ hơn 1.75 cm không? Nếu nhỏ hơn thì khả năng cao bông hoa thuộc loại *Iris versicolor* (độ sâu 2, bên trái). Nếu không, khả năng cao nó thuộc loại *Iris virginica* (độ sâu 2, bên phải). Việc dự đoán thực sự chỉ đơn giản như vậy thôi.

[1] Graphviz là một thư viện phần mềm minh họa đồ thị mã nguồn mở, có thể truy cập tại *http://www.graphviz.org/*.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ghi chú

Một trong những đặc điểm của Cây Quyết định là chúng không đòi hỏi ta phải chuẩn bị dữ liệu nhiều. Ta thậm chí không cần phải co giãn hoặc căn giữa giá trị của các đặc trưng.

Thuộc tính `samples` của một nút chứa số lượng mẫu huấn luyện được xử lý tại nút đó. Ví dụ, 100 mẫu huấn luyện có chiều dài cánh hoa lớn hơn 2.45 cm (độ sâu 1, bên phải), và trong số 100 mẫu đó, 54 mẫu có độ rộng cánh hoa nhỏ hơn 1.75 cm (độ sâu 2, bên trái). Thuộc tính `value` của một nút cho biết số mẫu tương ứng với mỗi lớp trong nút đó: ví dụ như nút dưới cùng bên phải không có mẫu *Iris setosa*, nhưng có 1 mẫu *Iris versicolor* và 45 mẫu *Iris virginica*. Cuối cùng, thuộc tính `gini` đo mức độ *pha tạp*: một nút là "thuần khiết" (`gini=0`) nếu mọi mẫu huấn luyện tương ứng đều thuộc cùng một lớp. Ví dụ, vì nút ở độ sâu 1 bên trái chỉ ứng với mẫu huấn luyện thuộc lớp *Iris setosa*, nó là một nút nguyên chất và có điểm số `gini` bằng 0. Phương trình 6.1 cho biết cách thuật toán huấn luyện tính điểm `gini` G_i của nút thứ i. Nút ở độ sâu 2 bên trái có điểm `gini` bằng $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$.

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

Phương trình 6.1. Độ pha tạp Gini

Trong đó:

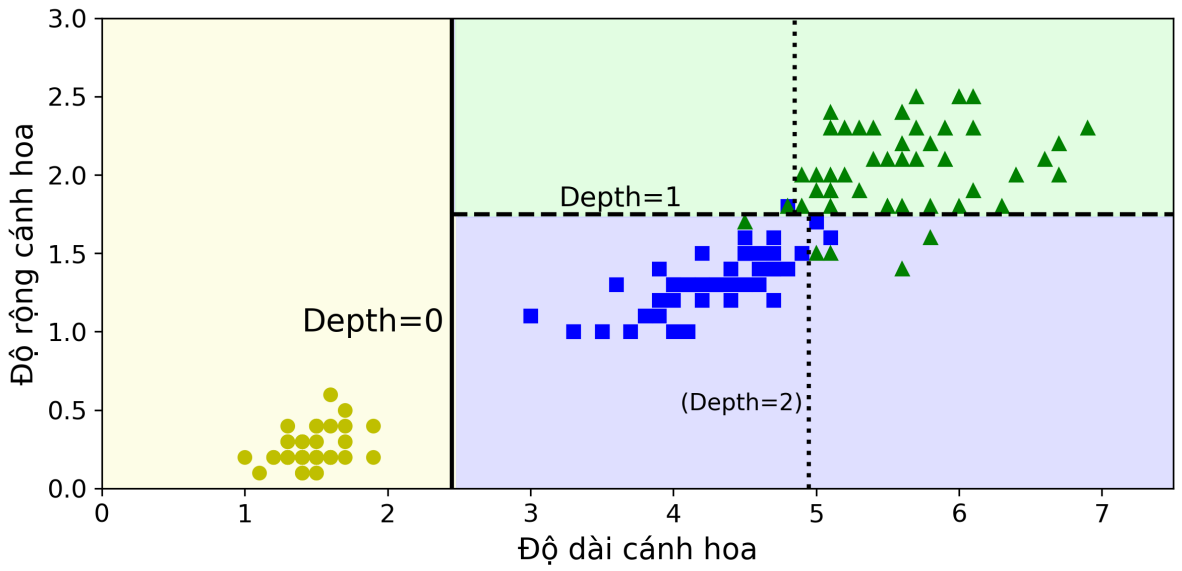
- $p_{i,k}$ là tỉ số giữa số mẫu thuộc lớp k và số mẫu huấn luyện trong nút thứ i.

Ghi chú

Scikit-Learn sử dụng thuật toán CART, và thuật toán này chỉ trả về *cây nhị phân*: các nút không phải nút lá chỉ có hai nút con (câu hỏi chỉ có câu trả lời là có/không). Tuy nhiên, các thuật toán khác như ID3 có thể tạo ra các Cây Quyết định với nút có nhiều hơn hai nút con.

Hình 6.2 minh họa ranh giới quyết định của Cây Quyết định này. Đường thẳng dọc tô đậm biểu diễn ranh giới quyết định của nút gốc (độ sâu 0): độ dài cánh hoa = 2.45 cm. Vì vùng bên trái đã là nguyên chất (chỉ có *Iris setosa*), nó không bị tách ra nữa. Tuy nhiên, vùng bên phải thì vẫn còn pha tạp nên nút ở độ sâu 1 bên phải tách nó ra ở độ rộng cánh hoa = 1.75 cm (biểu diễn bằng đường nét đứt). Vì `max_depth` được đặt bằng 2, Cây Quyết định dừng tại đây. Nếu ta đặt `max_depth` bằng 3 thì sẽ có thêm hai ranh giới quyết định cho hai nút tại độ sâu 2 (biểu diễn bằng đường chấm).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 6.2. Ranh giới Quyết định của Cây Quyết định

Diễn dịch Mô hình: Hộp Trắng và Hộp Đen

Mô hình Cây Quyết định rất dễ hiểu và quyết định mà chúng đưa ra có thể được diễn dịch dễ dàng. Các mô hình như vậy thường được gọi là các mô hình *hộp trắng* (*white box*). Ngược lại, các mô hình sau này như Rừng Ngẫu nhiên hay mạng nơ-ron thường được xem là mô hình *hộp đen* (*black box*). Mô hình hộp đen có thể đưa ra dự đoán tốt và ta có thể dễ dàng kiểm tra các phép tính dẫn đến dự đoán đó. Tuy nhiên, căn cứ mà mô hình đưa ra dự đoán lại khó có thể được giải thích bằng từ ngữ đơn giản. Ví dụ, nếu một mạng nơ-ron cho rằng một người nào đó xuất hiện trong một bức ảnh, rất khó để biết yếu tố nào góp phần vào dự đoán đó: Liệu đó là do mô hình nhận ra mắt, miệng hay mũi của họ? Nhận ra giày hay thậm chí là chiếc ghế người đó đang ngồi? Trái lại, Cây Quyết định cung cấp các quy tắc phân loại rõ ràng và đơn giản mà thậm chí có thể được sử dụng thủ công nếu cần (ví dụ như trong bài toán phân loại hoa trên).

Ước lượng Xác suất các Lớp

Cây Quyết định cũng có thể ước lượng xác suất một mẫu thuộc về một lớp k nào đó. Để tính xác suất trên, đầu tiên ta cần duyệt cây để tìm nút lá chứa mẫu đó, rồi tính tỷ lệ số lượng mẫu huấn luyện thuộc lớp k trong nút đó. Giả sử ta tìm được một bông hoa có cánh hoa dài 5 cm và rộng 1.5 cm. Nút lá tương ứng nằm ở độ sâu 2, nên Cây Quyết định sẽ đưa ra các xác suất sau: 0% cho *Iris setosa* (0/54), 90.7% cho *Iris versicolor* (49/54), và 9.3% cho *Iris virginica* (5/54). Và nếu ta yêu cầu đưa ra lớp dự đoán, mô hình sẽ dự đoán là *Iris versicolor* (lớp 1) vì lớp này có xác suất cao nhất. Hãy cùng kiểm chứng điều này:

```
>>> tree_clf.predict_proba([[5, 1.5]])
array([[0.        , 0.90740741, 0.09259259]])
>>> tree_clf.predict([[5, 1.5]])
array([1])
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hoàn hảo! Chú ý rằng các xác suất ước lượng sẽ giống hệt nhau tại bất cứ điểm nào nằm trong góc dưới bên phải của hình chữ nhật trong Hình 6.2, ví dụ như mẫu có cánh hoa dài 6 cm và rộng 1.5 cm (dù có thể thấy rõ ràng mẫu này khả năng cao thuộc lớp *Iris virginica*).

Thuật toán Huấn luyện CART

Scikit-Learn sử dụng thuật toán *Cây Phân loại và Hồi quy* (*Classification and Regression Tree* – CART) để huấn luyện Cây Quyết định (còn được gọi là "trồng" cây). Đầu tiên, thuật toán sẽ chia tập huấn luyện thành hai tập con theo đặc trưng k và ngưỡng t_k (ví dụ như "độ dài cánh hoa $\leq$ 2.45 cm"). Vậy chọn k và t_k như thế nào? Thuật toán sẽ tìm kiếm cặp (k, t_k) có thể tạo ra các tập con thuần khiết nhất (được đánh trọng số bởi kích thước tập). Phương trình 6.2 là hàm chi phí cần được cực tiểu hóa.

$$J(k, t_k) = \frac{m_{\text{trái}}}{m} G_{\text{trái}} + \frac{m_{\text{phải}}}{m} G_{\text{phải}}$$

$$\text{trong đó} \begin{cases} G_{\text{trái/phải}} \text{ đo độ pha tạp của tập con trái/phải,} \\ m_{\text{trái/phải}} \text{ là số lượng mẫu trong các tập con trái/phải.} \end{cases}$$

Phương trình 6.2. Hàm chi phí của CART cho bài toán phân loại

Một khi chia thành công tập huấn luyện thành hai tập con, thuật toán CART tiếp tục chia nhỏ các tập con với cùng logic, rồi đến các tập con nhỏ hơn, và cứ đệ quy như vậy. Thuật toán dừng lại khi đạt được chiều sâu tối đa (định nghĩa bởi siêu tham số `max_depth`), hoặc không tìm được cách chia để giảm độ pha tạp. Một vài siêu tham số khác để kiểm soát các điều kiện dừng phụ là (`min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf` và `max_leaf_nodes`).

Lưu ý

Có thể thấy, CART là một *thuật toán tham lam (greedy algorithm)*. Thuật toán này tìm kiếm tham lam cách chia tối ưu tại mức đầu tiên rồi lặp lại quá trình ở các mức sau đó. Nó không hề kiểm tra xem cách chia đó có giúp đạt độ pha tạp thấp nhất có thể tại các mức phía dưới hay không. Một thuật toán tham lam thường tìm ra nghiệm tốt nhưng không đảm bảo đó là nghiệm tối ưu.
Rất tiếc, việc tìm kiếm cây tối ưu là một bài toán NP-đầy đủ (*NP-Complete*),[2] yêu cầu thời gian tính toán $O(\exp(m))$, khiến cho bài toán trở nên bất khả tính ngay cả với các tập huấn luyện nhỏ. Chính vì thế ta nên hài lòng với một lời giải "tương đối tốt".

Độ phức tạp Tính toán

Khi dự đoán, ta cần duyệt Cây Quyết định từ gốc tới lá. Thường thì Cây Quyết định gần như cân bằng, nên việc duyệt cây yêu cầu ta đi qua khoảng $O(\log_2(m))$ nút.[3] Vì tại mỗi nút ta chỉ

[2] P là tập các bài toán có thể giải trong thời gian đa thức. NP là tập các bài toán mà lời giải thể được kiểm chứng trong thời gian đa thức. Bất kỳ bài toán NP nào cũng có thể được giảm lược xuống bài toán NP-khó trong thời gian đa thức. Một bài toán NP-đầy đủ vừa là NP và NP-khó. Một câu hỏi lớn chưa có lời giải trong toán học là liệu P = NP hay không. Nếu P $\neq$ NP (rất có khả năng), thì không có thuật toán nào có thể giải được bài toán NP-đầy đủ trong thời gian đa thức (ngoại trừ trên máy tính lượng tử).

[3] $\log_2$ là hàm logarit cơ số 2: $\log_2(m) = \log(m)/\log(2)$.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

cần kiểm tra giá trị của một đặc trưng, nên độ phức tạp tính toán tổng thể là $O(\log_2(m))$, không phụ thuộc vào số lượng đặc trưng. Do đó việc dự đoán rất nhanh, ngay cả với các tập huấn luyện lớn.

Khi huấn luyện, nếu `max_features` không được đặt, thuật toán sẽ so sánh tất cả đặc trưng của tất cả các mẫu tại mỗi nút. Việc so sánh này có độ phức tạp $O(n \times m \log_2(m))$. Với các tập huấn luyện nhỏ (ít hơn vài nghìn mẫu), Scikit-Learn có thể tăng tốc huấn luyện bằng việc tiền sắp xếp dữ liệu (đặt `presort=True`), nhưng làm vậy với các tập huấn luyện lớn sẽ khiến quá trình huấn luyện chậm đi đáng kể.

Độ Pha tạp Gini hay Entropy?

Theo mặc định, độ pha tạp Gini được sử dụng, nhưng ta cũng có thể chọn độ pha tạp *entropy* bằng cách gán siêu tham số `criterion` bằng `"entropy"`. Khái niệm entropy xuất phát từ nhiệt động lực học. Đó là phép đo sự hỗn loạn của các phân tử: entropy tiến đến 0 khi các phân tử đứng yên và có trật tự. Sau này entropy được sử dụng trong rất nhiều lĩnh vực, bao gồm *lý thuyết thông tin* (*information theory*) của Shannon, dùng để đo lượng thông tin trung bình của một bản tin:[4] entropy bằng không khi tất cả các bản tin giống hệt nhau. Trong Học Máy, entropy thường được sử dụng như một phép đo độ pha tạp: entropy của một tập bằng 0 khi tập đó chỉ chứa các mẫu thuộc một lớp. Phương trình 6.3 định nghĩa entropy của nút thứ i. Ví dụ, nút trái ở độ sâu 2 trong Hình 6.1 có entropy bằng $-(49/54)\log_2(49/54) - (5/54)\log_2(5/54) \approx 0.445$.

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2\left(p_{i,k}\right)$$

Phương trình 6.3. Entropy

Vậy, ta nên sử dụng độ pha tạp Gini hay entropy? Nhìn chung chúng không tạo ra khác biệt quá lớn: kết quả thu được đều là các cây gần giống nhau. Do việc tính toán nhanh hơn một chút, độ pha tạp Gini thường được dùng làm lựa chọn mặc định. Tuy nhiên, cần lưu ý rằng độ pha tạp Gini có xu hướng cô lập lớp xuất hiện nhiều nhất trong riêng một nhánh cây, trong khi entropy có xu hướng tạo ra các cây cân bằng hơn một chút.[5]

Các Siêu tham số Điều chuẩn

Cây Quyết định đặt rất ít giả định về dữ liệu huấn luyện (khác với các mô hình tuyến tính giả định dữ liệu là tuyến tính). Nếu không có ràng buộc, cấu trúc cây sẽ tự thích ứng với dữ liệu huấn luyện để khớp dữ liệu rất sát, và tất nhiên rất có thể sẽ quá khớp. Một mô hình như vậy thường được gọi là *mô hình phi tham số* (*nonparametric model*), không phải vì chúng không hề có tham số (thực tế thường có rất nhiều) mà vì số lượng tham số không được quyết định trước khi huấn luyện, nên cấu trúc mô hình có thể khớp với dữ liệu một cách tự do. Ngược lại, một *mô hình tham số* (*parametric model*) như mô hình tuyến tính, có số lượng tham số xác định trước nên số bậc tự do bị giới hạn, giảm thiểu nguy cơ quá khớp (nhưng lại tăng nguy cơ dưới khớp).

Để tránh việc quá khớp dữ liệu huấn luyện, ta cần giới hạn số bậc tự do của Cây Quyết định khi huấn luyện. Như đã biết, việc này được gọi là điều chuẩn. Các siêu tham số điều chuẩn phụ

[4] Sự giảm entropy thường được gọi là *mức tăng thông tin* (*information gain*).
[5] Tham khảo bài phân tích rất thú vị của Sebastian Raschka để tìm hiểu thêm.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

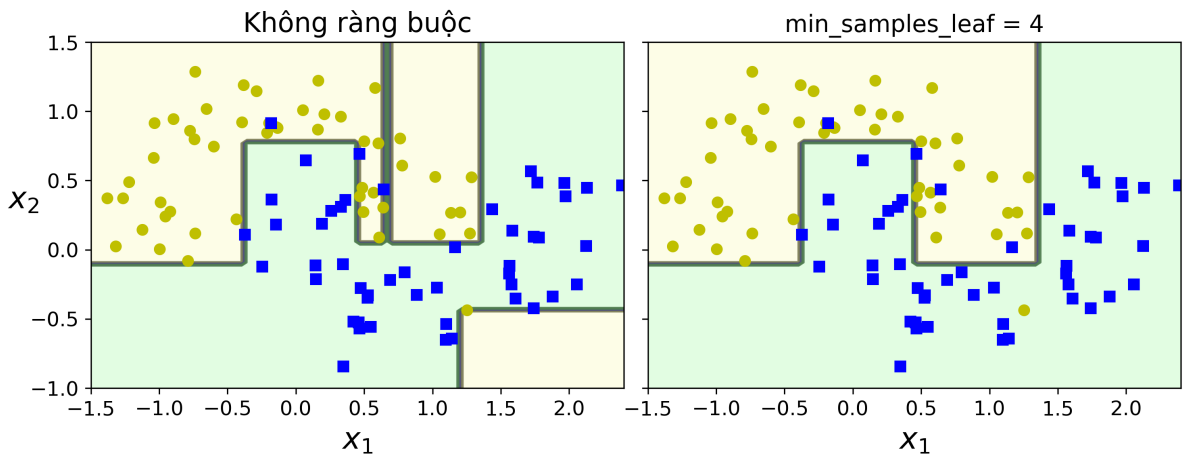
thuộc vào thuật toán đang được sử dụng, nhưng nhìn chung ít nhất ta có thể giới hạn độ sâu tối đa của cây. Trong Scikit-Learn, việc này được điều khiển bởi siêu tham số `max_depth` (giá trị mặc định là `None`, tức không giới hạn). Việc giảm `max_depth` sẽ điều chuẩn mô hình và do đó giảm thiểu nguy cơ quá khớp.

Lớp `DecisionTreeClassifier` có một vài siêu tham số khác dành cho việc giới hạn kích thước cây: `min_samples_split` (số lượng mẫu tối thiểu một nút phải có trước khi phân chia nút đó), `min_samples_leaf` (số lượng mẫu tối thiểu một nút phải có), `min_weight_fraction_leaf` (giống `min_samples_leaf` nhưng được biểu diễn dưới dạng tỷ lệ của tổng các mẫu có trọng số), `max_leaf_nodes` (số lượng nút lá tối đa), và `max_features` (số lượng đặc trưng tối đa được đánh giá khi phân chia tại các nút). Việc tăng các siêu tham số `min_*` hoặc giảm các siêu tham số `max_*` sẽ điều chuẩn mô hình.

Ghi chú

Các thuật toán khác huấn luyện Cây Quyết định mà không có ràng buộc, rồi *cắt tỉa* (*prune*) những nút không cần thiết. Một nút có toàn bộ nút con là nút lá sẽ được cho là không cần thiết nếu mức cải thiện độ pha tạp mà nó mang lại không có ý nghĩa thống kê. Các phép kiểm định thống kê tiêu chuẩn như *kiểm định* χ^2 (kiểm định chi-bình phương) được dùng để ước lượng xác suất mà sự cải thiện hoàn toàn là do ngẫu nhiên (điều này được gọi là *giả thuyết gốc – null hypothesis*). Xác suất này được gọi là *trị số p*, và nếu nó cao hơn một mức ngưỡng cho trước (một siêu tham số, thường là 5%) thì nút này được cho là không cần thiết và các nút con của nó sẽ bị loại bỏ. Việc cắt tỉa được tiếp tục cho đến khi mọi nút không cần thiết đã được loại bỏ.

Hình 6.3 minh họa hai Cây Quyết định được huấn luyện trên tập dữ liệu moons (đã giới thiệu trong Chương 5). Bên trái là Cây Quyết định được huấn luyện với các siêu tham số mặc định (tức không bị ràng buộc), và bên phải là cây được huấn luyện với `min_samples_leafs=4`. Có thể thấy rõ rằng mô hình bên trái đang quá khớp, và mô hình bên phải khái quát hóa tốt hơn.



Hình 6.3. Điều chuẩn bằng `min_samples_leafs`

Hồi quy

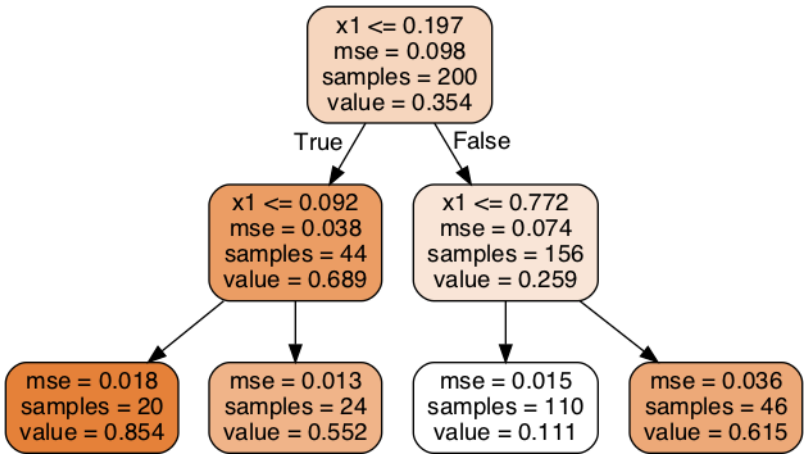
Cây Quyết định cũng có thể giải quyết các tác vụ hồi quy. Hãy xây dựng một cây hồi quy bằng lớp `DecisionTreeRegressor` của Scikit-Learn và huấn luyện nó trên một tập dữ liệu bậc hai bị

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

nhiễu với `max_depth=2`:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(max_depth=2)
tree_reg.fit(X, y)
```

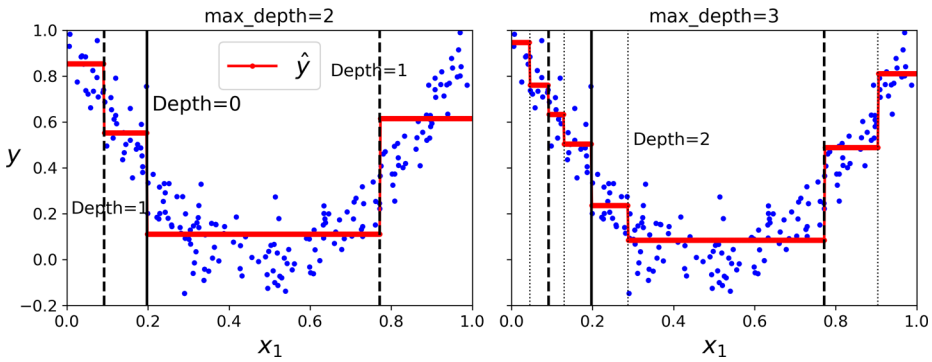
Kết quả là cây được biểu diễn trong Hình 6.4.



Hình 6.4. Cây Quyết định Hồi quy

Cây này trông khá giống với cây phân loại mà ta đã xây dựng trước đó. Điểm khác biệt chính là thay vì dự đoán lớp tại mỗi nút, nó sẽ dự đoán một giá trị. Ví dụ, giả sử ta muốn dự đoán một mẫu mới với $x_1 = 0.6$. Ta sẽ duyệt cây bắt đầu từ gốc, và cuối cùng sẽ duyệt tới nút lá với dự đoán `value=0.111`. Dự đoán này là giá trị mục tiêu trung bình của 110 mẫu huấn luyện ứng với nút lá này, và dẫn đến trung bình bình phương sai số bằng 0.015 cho 110 mẫu đó.

Dự đoán của mô hình này được biểu diễn ở biểu đồ bên trái trong Hình 6.5. Nếu đặt `max_depth=3`, ta sẽ thu được dự đoán ở biểu đồ bên phải. Chú ý rằng giá trị dự đoán ở mỗi vùng luôn là giá trị mục tiêu trung bình của các mẫu nằm trong vùng đó. Thuật toán phân chia các vùng sao cho hầu hết các mẫu huấn luyện nằm gần giá trị dự đoán đó nhất có thể.



Hình 6.5. Dự đoán của hai mô hình Cây Quyết định Hồi quy

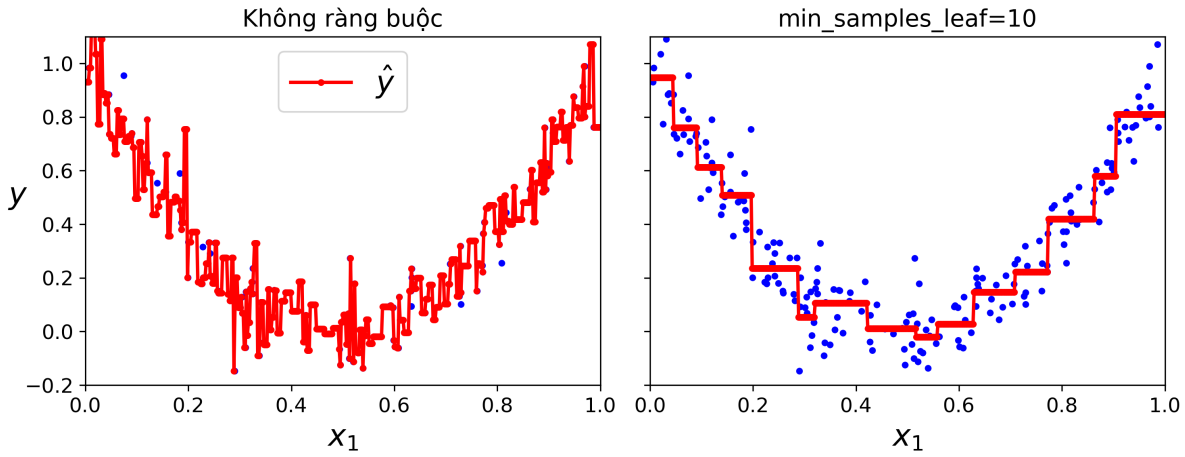
Thuật toán CART hoạt động gần tương tự như trước, chỉ thay việc cực tiểu hóa độ pha tạp thành cực tiểu hóa MSE. Phương trình 6.4 mô tả hàm chi phí của thuật toán CART.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$J(k, t_k) = \frac{m_{\text{trái}}}{m}\text{MSE}_{\text{trái}} + \frac{m_{\text{phải}}}{m}\text{MSE}_{\text{phải}} \quad \text{trong đó} \quad \begin{cases} \text{MSE}_{\text{nút}} = \sum_{i \in \text{nút}} \left(\hat{y}_{\text{nút}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{nút}} = \frac{1}{m_{\text{nút}}} \sum_{i \in \text{nút}} y^{(i)} \end{cases}$$

Phương trình 6.4. Hàm chi phí của CART cho bài toán hồi quy

Giống như với các tác vụ phân loại, Cây Quyết định cũng dễ bị quá khớp khi giải quyết các tác vụ hồi quy. Nếu không sử dụng điều chuẩn (tức giữ nguyên các tham số mặc định), ta sẽ thu được các dự đoán ở biểu đồ bên trái trong Hình 6.6. Các dự đoán này rõ ràng đang quá khớp tập huấn luyện một cách tồi tệ. Chỉ cần đặt `min_samples_leaf=10`, ta sẽ có một mô hình phù hợp hơn nhiều, được biểu diễn ở biểu đồ bên phải.

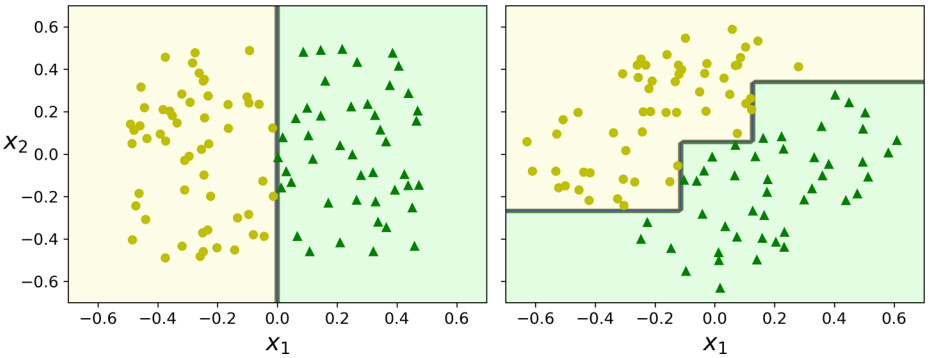


Hình 6.6. Điều chuẩn Cây Quyết định Hồi quy

Sự Bất ổn

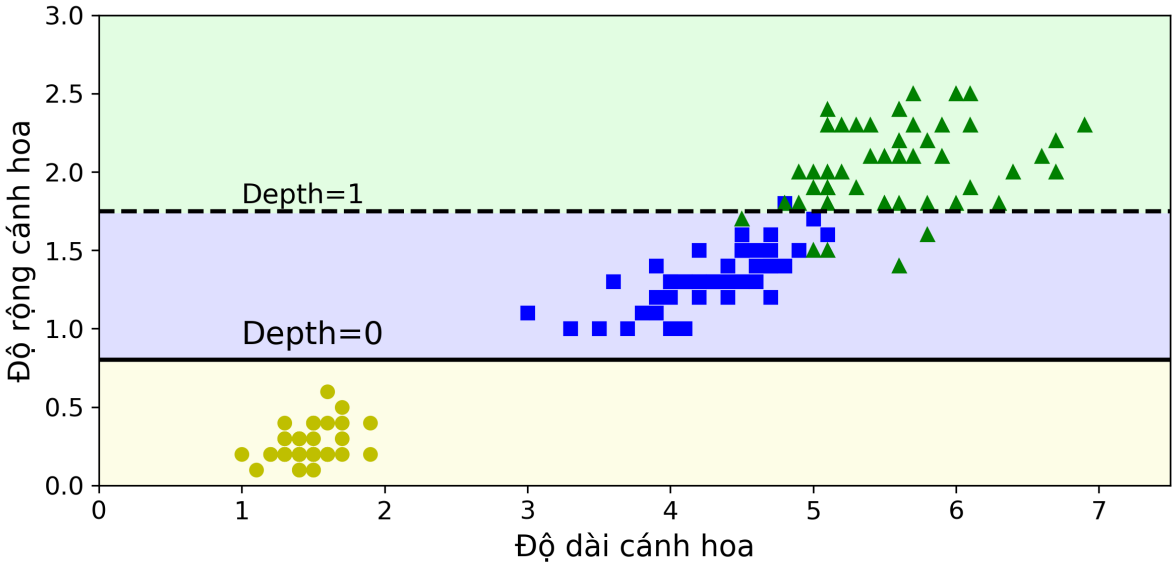
Hy vọng bạn đã hiểu rằng Cây Quyết định có rất nhiều điểm mạnh: chúng đơn giản, dễ hiểu, dễ sử dụng, linh hoạt và mạnh mẽ. Tuy nhiên, chúng vẫn có một vài hạn chế. Đầu tiên, có thể thấy, Cây Quyết định rất thích các ranh giới quyết định trực giao (tất cả các đường phân chia đều vuông góc với một trục), và điều này khiến chúng nhạy cảm với phép xoay tập dữ liệu. Ví dụ, Hình 6.7 biểu diễn một tập dữ liệu tách biệt tuyến tính đơn giản: trong hình bên trái, cây Quyết định có thể phân chia tập dữ liệu một cách dễ dàng, còn ở hình bên phải, khi tập dữ liệu bị xoay 45°, ranh giới quyết định trở nên phức tạp một cách không cần thiết. Dù cả hai Cây Quyết định đều khớp tập huấn luyện một cách hoàn hảo, khả năng cao là mô hình bên phải sẽ không khái quát hóa tốt. Một cách xử lý vấn đề này là sử dụng Phân tích Thành phần Chính (*Principal Component Analysis* – tham khảo Chương 8). Phương pháp này thường sẽ giúp tập dữ liệu có "góc xoay" tốt hơn.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 6.7. Sự nhạy cảm với phép xoay tập dữ liệu

Tổng quát hơn, nhược điểm chính của Cây Quyết định là sự nhạy cảm với các thay đổi nhỏ trong tập huấn luyện. Ví dụ, nếu loại bỏ bông hoa *Iris versicolor* lớn nhất trong tập huấn luyện (bông có cánh dài 4.8 cm và rộng 1.8 cm) rồi huấn luyện một Cây Quyết định mới, ta có thể thu được mô hình được biểu diễn trong Hình 6.8. Có thể thấy nó trông rất khác so với Cây Quyết định ở trên (Hình 6.2). Thực chất, vì Scikit-Learn sử dụng thuật toán huấn luyện có tính ngẫu nhiên,[6] các mô hình thu được có thể rất khác nhau kể cả khi sử dụng cùng dữ liệu huấn luyện (trừ khi ta đặt siêu tham số `random_state`).



Hình 6.8. Sự nhạy cảm với các chi tiết trong tập huấn luyện

Rừng Ngẫu nhiên có thể giảm thiểu sự bất ổn này bằng cách lấy trung bình dự đoán của nhiều cây, và ta sẽ thấy điều này ở chương tiếp theo.

Bài tập

1. Chiều sâu của Cây Quyết định là khoảng bao nhiêu nếu nó được huấn luyện (không bị giới hạn) trên tập dữ liệu chứa một triệu mẫu?

[6] Thuật toán lựa chọn một tập đặc trưng ngẫu nhiên để đánh giá tại mỗi nút.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

2. Độ pha tạp Gini của nút con thường thấp hơn hay cao hơn độ pha tạp Gini của nút cha? Nó *thường* thấp/cao hơn, hay *luôn* thấp/cao hơn?

3. Nếu Cây Quyết định đang quá khớp tập huấn luyện thì có nên thử giảm `max_depth` không?

4. Nếu Cây Quyết định đang dưới khớp tập huấn luyện thì có nên thử co giãn đặc trưng đầu vào không?

5. Nếu việc huấn luyện Cây Quyết định trên tập huấn luyện chứa 1 triệu mẫu mất một tiếng, sẽ mất khoảng bao lâu để huấn luyện một Cây Quyết định khác trên tập huấn luyện chứa 10 triệu mẫu?

6. Nếu tập huấn luyện chứa 100,000 mẫu, việc đặt `presort=True` có tăng tốc quá trình huấn luyện không?

7. Huấn luyện và tinh chỉnh một Cây Quyết định trên tập dữ liệu moons theo các bước sau:

   a. Dùng `make_moons(n_samples=10000, noise=0.4)` để sinh tập dữ liệu moons.

   b. Dùng `train_test_split()` để chia tập dữ liệu thành một tập huấn luyện và một tập kiểm tra.

   c. Dùng tìm kiếm dạng lưới cùng kiểm định chéo (sử dụng lớp `GridSearchCV`) để tìm giá trị siêu tham số tốt cho một `DecisionTreeClassifier`. Gợi ý: hãy thử nhiều giá trị khác nhau cho `max_leaf_nodes`.

   d. Huấn luyện mô hình trên tập huấn luyện đầy đủ với các siêu tham số tìm được, và đo chất lượng của mô hình trên tập kiểm tra. Bạn sẽ thu được độ chính xác khoảng 85% đến 87%.

8. Trồng một khu rừng theo các bước sau:

   a. Tiếp tục từ bài tập trước, sinh 1,000 tập con của tập huấn luyện, mỗi tập chứa 100 mẫu được chọn ngẫu nhiên. Gợi ý: bạn có thể dùng lớp `ShuffleSplit` của Scikit-Learn để thực hiện việc này.

   b. Huấn luyện một Cây Quyết định cho mỗi tập con với các siêu tham số tốt nhất tìm được ở bài tập trước. Đánh giá 1,000 Cây Quyết định này trên tập kiểm tra. Vì chúng được huấn luyện trên các tập dữ liệu nhỏ hơn, khả năng cao các cây này sẽ hoạt động tệ hơn Cây Quyết định đầu tiên, chỉ đạt được độ chính xác khoảng 80%.

   c. Giờ là lúc thực hiện phép màu. Với mỗi mẫu kiểm tra, sinh dự đoán của 1,000 Cây Quyết định và chỉ giữ lại dự đoán xuất hiện nhiều nhất (bạn có thể dùng hàm `mode()` của SciPy để làm việc này). Phương pháp này cho ta *dự đoán biểu quyết theo đa số* (*majority-vote prediction*) trên tập kiểm tra.

   d. Đánh giá các dự đoán đó trên tập kiểm tra: bạn sẽ thu được độ chính xác cao hơn một chút (từ 0.5 đến 1.5%) so với mô hình đầu tiên. Chúc mừng, bạn đã huấn luyện xong một bộ phân loại Rừng Ngẫu nhiên!

Lời giải của các bài tập trên nằm ở Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 7

Học Ensemble và Rừng Ngẫu nhiên

Giả sử, ta đặt một câu hỏi phức tạp cho hàng nghìn người được chọn ngẫu nhiên, và tổng hợp các câu trả lời của họ. Trong nhiều trường hợp, ta sẽ thấy câu trả lời có được thường tốt hơn câu trả lời của một chuyên gia. Hiện tượng này được gọi là *trí tuệ đám đông*. Tương tự, nếu kết hợp một nhóm bộ dự đoán (chẳng hạn như các bộ phân loại và hồi quy), dự đoán tổng hợp nhận được thường tốt hơn dự đoán riêng lẻ của bộ dự đoán tốt nhất. Một nhóm các bộ phân loại được gọi là *ensemble*, và vì thế kỹ thuật trên còn được gọi là *Học Ensemble*. Một thuật toán Học Ensemble được gọi là *phương pháp Ensemble*.

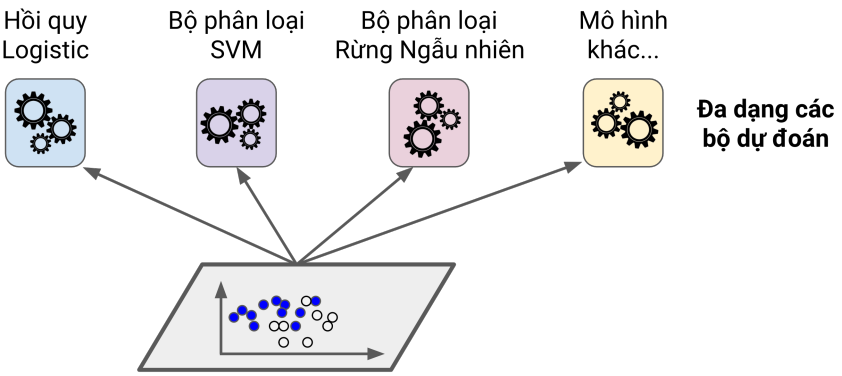
Ví dụ, ta có thể huấn luyện một nhóm bộ phân loại Cây Quyết định, mỗi bộ được huấn luyện trên một tập con ngẫu nhiên của tập huấn luyện. Để đưa ra dự đoán, ta tổng hợp dự đoán của tất cả các cây, sau đó đưa ra kết quả là lớp được chọn nhiều nhất (tham khảo bài tập cuối cùng trong Chương 6). Một ensemble gồm các Cây Quyết định như vậy được gọi là *Rừng Ngẫu nhiên* (*Random Forest*), và tuy đơn giản nhưng đây lại là một trong những thuật toán Học Máy hiệu quả nhất hiện nay.

Như đã thảo luận tại Chương 2, sau khi đã xây dựng được một vài bộ dự đoán tốt, ta thường sử dụng các phương pháp Ensemble để kết hợp chúng thành một hệ thống dự đoán tốt hơn. Trên thực tế, những giải pháp giành chiến thắng trong các cuộc thi Học Máy thường bao gồm một số phương pháp Ensemble (nổi bật nhất là cuộc thi Netflix Prize).

Trong chương này, chúng ta sẽ thảo luận về các phương pháp Ensemble phổ biến nhất, bao gồm *bagging*, *boosting*, và *stacking*. Ngoài ra, ta cũng sẽ tìm hiểu thêm về thuật toán Rừng Ngẫu nhiên.

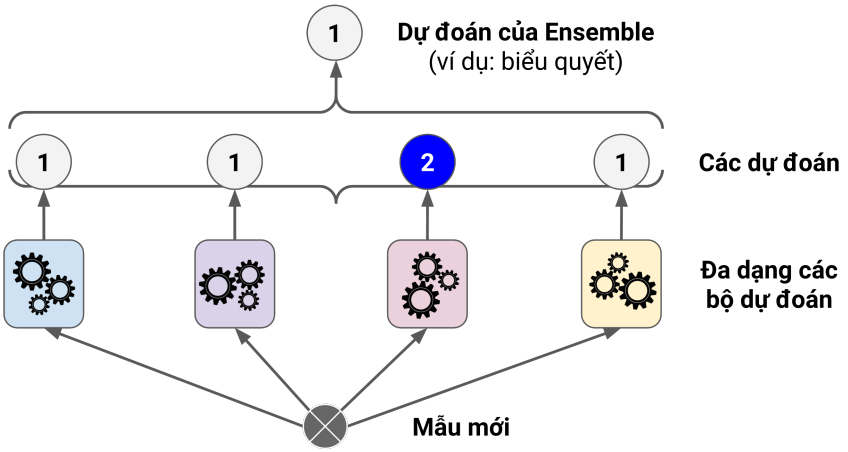
Bộ phân loại Biểu quyết

Giả sử ta đã huấn luyện xong một vài bộ phân loại, mỗi bộ phân loại có độ chính xác khoảng 80%. Các bộ phân loại có thể là Hồi quy Logistic, SVM, Rừng Ngẫu nhiên, K-Điểm Gần nhất hoặc một vài mô hình khác nữa (tham khảo Hình 7.1).



Hình 7.1. Huấn luyện các bộ phân loại khác nhau

Một cách đơn giản để có được một bộ phân loại vượt trội hơn là tổng hợp các dự đoán của mỗi bộ phân loại riêng lẻ, sau đó đưa ra kết quả dựa trên lớp được biểu quyết nhiều nhất. Bộ phân loại biểu quyết theo đa số này được gọi là bộ phân loại *biểu quyết cứng* (*hard voting classifier* – tham khảo Hình 7.2).

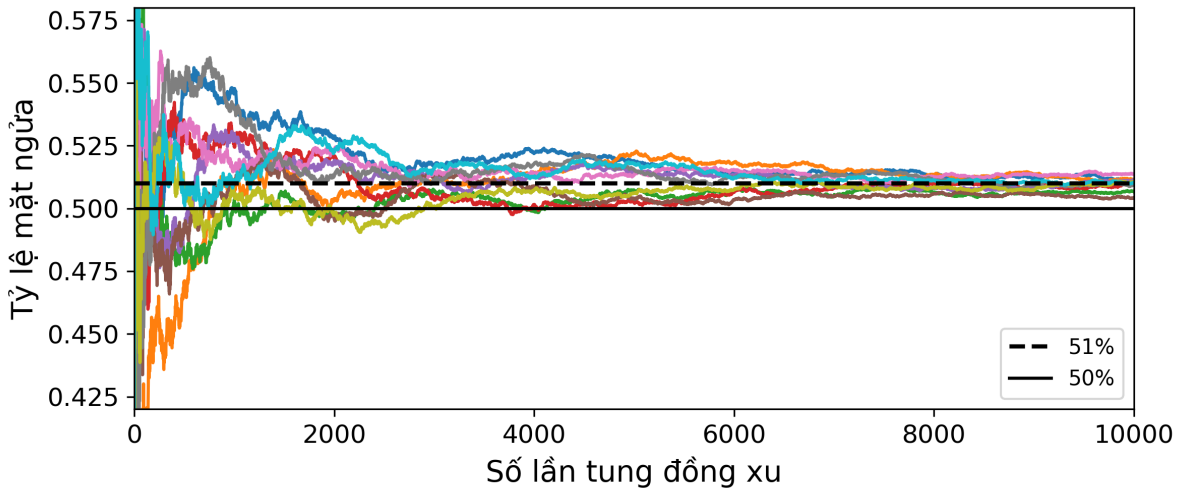


Hình 7.2. Dự đoán của bộ phân loại biểu quyết cứng

Một điều ngạc nhiên là bộ phân loại biểu quyết cứng thường có độ chính xác cao hơn bộ phân loại riêng lẻ tốt nhất trong ensemble. Trên thực tế, ngay cả khi mỗi bộ phân loại trong ensemble là một *bộ học yếu* (*weak learner* – nghĩa là nó chỉ hoạt động tốt hơn một chút so với dự đoán ngẫu nhiên), thì ensemble của chúng vẫn có thể là một *bộ học mạnh* (*strong learner* – đạt độ chính xác cao), miễn là ta có đủ số lượng bộ học yếu và chúng phải đủ khác biệt so với nhau.

Tại sao lại như vậy? Sự so sánh sau đây có thể giúp làm sáng tỏ bí ẩn này. Giả sử ta có một đồng xu không cân đối với xác suất 51% cho mặt ngửa và 49% cho mặt sấp. Nếu tung đồng xu này 1,000 lần, ta sẽ có xấp xỉ 510 lần mặt ngửa và 490 lần mặt sấp, khi đó mặt ngửa chiếm đa số. Tính toán một chút, ta sẽ thấy rằng xác suất để mặt ngửa đạt đa số sau 1,000 lần tung là gần 75%. Càng tung đồng xu nhiều lần, xác suất này càng cao (ví dụ như với 10,000 lần tung, xác suất này lên đến 97%). Điều này có thể được giải thích bởi *luật số lớn* (*law of large numbers*): càng tung đồng xu nhiều lần, tỉ lệ thu được mặt ngửa sẽ càng tiến gần xác suất xuất hiện mặt ngửa (51%). Hình 7.3 biểu diễn 10 loạt tung đồng xu này. Có thể thấy rằng khi số lần tung tăng lên, tỷ lệ mặt ngửa tiến tới 51%. Cuối cùng, toàn bộ 10 loạt tung đều kết thúc rất gần 51% với kết quả luôn ổn định trên mức 50%.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 7.3. Luật số lớn

Tương tự, giả sử ta xây dựng được một ensemble chứa 1,000 bộ phân loại với độ chính xác riêng lẻ chỉ là 51% (tốt hơn một chút so với dự đoán ngẫu nhiên). Nếu đưa ra dự đoán dựa trên đa số, ta có thể đạt được độ chính xác lên đến 75%! Tuy nhiên, điều này chỉ đúng nếu tất cả bộ phân loại hoàn toàn độc lập và lỗi dự đoán của chúng không tương quan với nhau. Điều kiện này rõ ràng là không thể thỏa mãn bởi chúng đều được huấn luyện trên cùng một tập dữ liệu. Khả năng cao là chúng sẽ mắc cùng một loại lỗi, dẫn đến nhiều dự đoán sai dù được biểu quyết theo đa số và làm giảm độ chính xác của ensemble.

Mẹo

Phương pháp Ensemble hoạt động tốt nhất khi các bộ dự đoán càng độc lập với nhau càng tốt. Một cách để có được các bộ phân loại đa dạng là huấn luyện chúng với những thuật toán khác nhau. Điều này làm tăng khả năng các bộ phân loại mắc phải những lỗi rất khác nhau, nhờ đó cải thiện độ chính xác của ensemble.

Đoạn mã sau khởi tạo và huấn luyện một bộ phân loại biểu quyết với Scikit-Learn, là ensemble của ba bộ phân loại khác nhau (tập huấn luyện là tập dữ liệu moons được giới thiệu trong Chương 5):

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```


Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hãy xem độ chính xác của từng bộ phân loại trên tập kiểm tra:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

Có thể thấy, bộ phân loại biểu quyết hoạt động tốt hơn một chút so với mỗi bộ phân loại riêng lẻ.

Nếu tất cả các bộ phân loại đều có thể ước tính xác suất của từng lớp (tức là chúng đều có phương thức `predict_proba()`), ta có thể yêu cầu Scikit-Learn lấy trung bình xác suất từng lớp trên tất cả các bộ phân loại riêng lẻ, rồi đưa ra dự đoán là lớp có xác suất trung bình cao nhất. Phương pháp này được gọi là *biểu quyết mềm* (*soft voting*), và thường đạt chất lượng cao hơn so với biểu quyết cứng bởi nó gán trọng số lớn hơn cho các biểu quyết có độ tin cậy cao. Trong Scikit-Learn, ta chỉ cần thay `voting="hard"` bằng `voting="soft"` và đảm bảo rằng tất cả các bộ phân loại riêng lẻ đều có thể ước tính xác suất của lớp. Điều này không đúng với khởi tạo mặc định của lớp `SVC`, vì vậy ta cần đặt siêu tham số `probability` của lớp này thành `True` (khi đó `SVC` sẽ sử dụng kiểm định chéo để ước tính xác suất lớp, khiến cho quá trình huấn luyện lâu hơn, nhưng sẽ có phương thức `predict_proba()`). Nếu sửa đoạn mã trên thành biểu quyết mềm, ta sẽ thấy rằng bộ phân loại biểu quyết đạt độ chính xác lên đến hơn 91.2%.

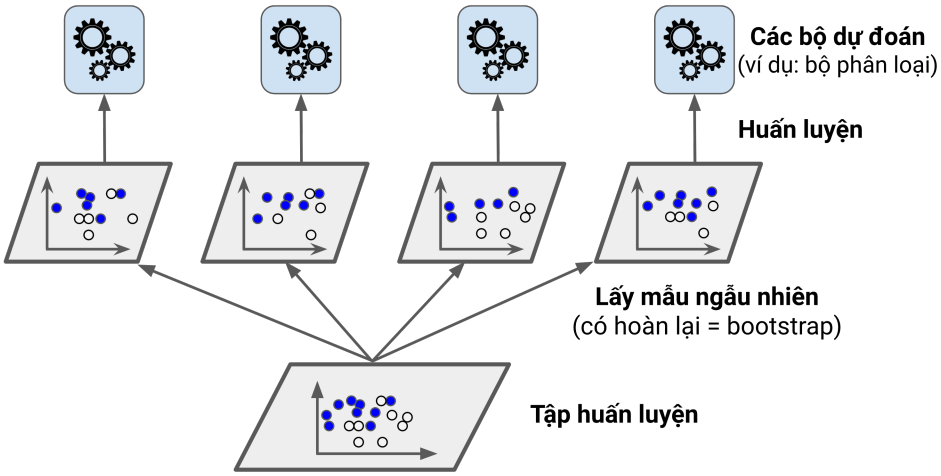
Bagging và Pasting

Một cách để có được các bộ phân loại đa dạng là sử dụng các thuật toán huấn luyện khác nhau như đã thảo luận ở trên. Một hướng tiếp cận khác là sử dụng cùng một thuật toán huấn luyện cho mỗi bộ dự đoán nhưng huấn luyện chúng trên các tập con ngẫu nhiên của tập huấn luyện. Nếu ta lấy mẫu *có* hoàn lại, phương pháp này được gọi là *bagging*[1] (viết tắt của *bootstrap aggregating* – tạm dịch: *tổng hợp bootstrap*[2]). Ngược lại, nếu ta thực hiện lấy mẫu *không* hoàn lại, phương pháp này sẽ được gọi là *pasting*.[3]

Nói cách khác, cả hai phương pháp bagging và pasting cho phép các mẫu huấn luyện được sử dụng nhiều lần cho các bộ dự đoán, nhưng chỉ có phương pháp bagging cho phép các mẫu được sử dụng nhiều lần cho cùng một bộ dự đoán. Quá trình lấy mẫu và huấn luyện này được mô tả trong Hình 7.4.

[1] Leo Breiman, "Bagging Predictors," *Machine Learning* 24, no. 2 (1996): 123–140.
[2] Trong thống kê, lấy mẫu có hoàn lại được gọi là *bootstrapping*.
[3] Leo Breiman, "Pasting Small Votes for Classification in Large Databases and On-Line," *Machine Learning* 36, no. 1–2 (1999): 85–103.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 7.4. Phương pháp bagging và pasting cho phép huấn luyện các bộ dự đoán riêng lẻ trên các tập con ngẫu nhiên của tập huấn luyện

Sau khi tất cả các bộ dự đoán riêng lẻ đã được huấn luyện, ensemble có thể đưa ra dự đoán trên một mẫu mới bằng cách tổng hợp kết quả của các bộ dự đoán riêng lẻ. Hàm tổng hợp thường là *yếu vị thống kê* (*statistical mode* – tức lấy dự đoán xuất hiện nhiều nhất, tương tự như bộ phân loại biểu quyết cứng) cho tác vụ phân loại, hoặc lấy trung bình cho tác vụ hồi quy. Mỗi bộ dự đoán riêng lẻ thường có độ chệch cao hơn so với khi chúng được huấn luyện trên tập huấn luyện gốc, tuy nhiên việc tổng hợp sẽ giảm cả độ chệch lẫn phương sai.[4] Nhìn chung, kết quả thu được là một ensemble có độ chệch tương đương với một bộ dự đoán riêng lẻ được huấn luyện trên tập dữ liệu gốc nhưng có phương sai thấp hơn.

Như ta có thể thấy trong Hình 7.4, các bộ dự đoán riêng lẻ có thể được huấn luyện song song, thông qua các lõi CPU khác nhau hoặc thậm chí các server khác nhau. Tương tự, việc đưa ra dự đoán cũng có thể được thực hiện một cách song song. Đây là một trong những lý do tại sao bagging và pasting rất phổ biến: hai phương pháp này có thể mở rộng tốt.

Bagging và Pasting trong Scikit-Learn

Scikit-Learn cung cấp một API đơn giản cho cả phương pháp bagging và pasting với lớp `BaggingClassifier` (hoặc `BaggingRegressor` cho tác vụ hồi quy). Đoạn mã dưới đây huấn luyện một ensemble gồm 500 bộ phân loại Cây Quyết định:[5] mỗi bộ phân loại được huấn luyện trên 100 mẫu được lấy mẫu ngẫu nhiên có hoàn lại từ tập huấn luyện (với bagging, nếu muốn dùng pasting ta cần đặt `bootstrap=False`). Tham số `n_jobs` chỉ định số lõi CPU mà Scikit-Learn có thể sử dụng để huấn luyện và dự đoán (`-1` cho phép Scikit-Learn sử dụng tất cả các lõi):

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

[4] Độ chệch và phương sai được giới thiệu trong Chương 4.

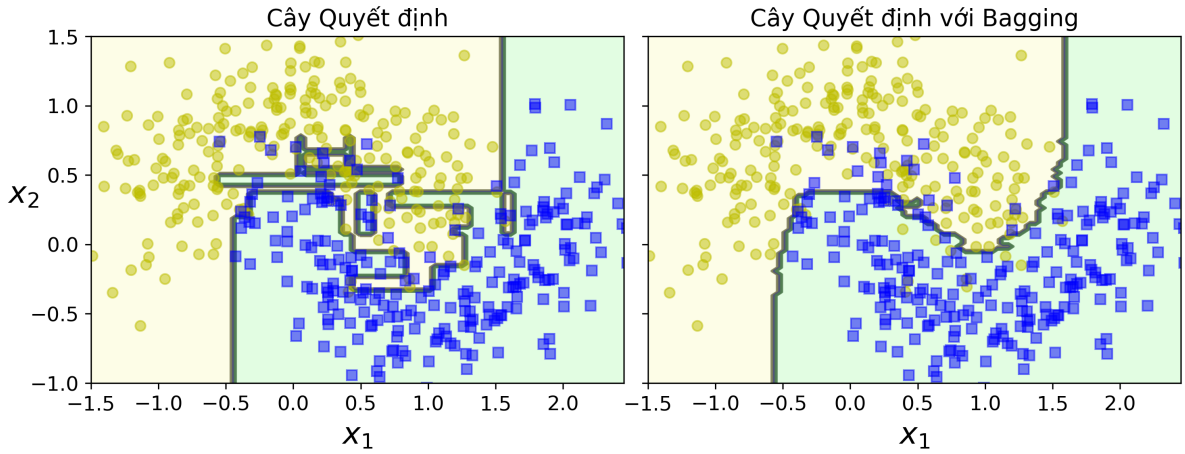
[5] `max_samples` có thể được gán bằng số thực từ 0.0 tới 1.0, trong trường hợp này số lượng lấy mẫu bằng tích giữa số mẫu trong tập huấn luyện và `max_samples`.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ghi chú

`BaggingClassifier` tự động thực hiện biểu quyết mềm thay vì biểu quyết cứng nếu bộ phân loại cơ sở có thể ước tính được xác suất lớp (tức có phương thức `predict_proba()`); trường hợp này đúng với bộ phân loại Cây Quyết định.

Hình 7.5 so sánh ranh giới quyết định của một Cây Quyết định và một ensemble gồm 500 cây theo phương pháp bagging (từ đoạn mã trên), cả hai đều được huấn luyện trên tập dữ liệu moons. Có thể thấy, các dự đoán của ensemble có tính khái quát cao hơn nhiều so với một Cây Quyết định riêng lẻ: ensemble có độ chệch tương đương nhưng có phương sai thấp hơn (có số lỗi trên tập huấn luyện tương đương nhưng có ranh giới quyết định mượt hơn).



Hình 7.5. Cây Quyết định riêng lẻ (trái) và ensemble bagging của 500 cây quyết định (phải)

Bagging, do Bootstrap, sử dụng các tập dữ liệu con đa dạng hơn để huấn luyện các bộ dự đoán, vì vậy bagging sẽ có độ chệch cao hơn một chút so với pasting. Tuy nhiên, sự đa dạng này đồng nghĩa với việc các bộ dự đoán trở nên ít tương quan hơn, dẫn đến giảm phương sai của ensemble. Nhìn chung, bagging thường tạo ra các mô hình ensemble tốt hơn, nên vì thế được sử dụng phổ biến hơn. Nếu bạn có thời gian cùng nhiều tài nguyên CPU, bạn có thể sử dụng kiểm định chéo để đánh giá cả bagging và pasting, từ đó đưa ra lựa chọn tốt nhất.

Đánh giá Out-of-Bag

Với bagging, một vài mẫu có thể được sử dụng nhiều lần để huấn luyện một bộ dự đoán, trong khi một số mẫu khác lại không được sử dụng. Theo mặc định, một thực thể `BaggingClassifier` lấy mẫu có hoàn lại m mẫu huấn luyện (`bootstrap=True`), trong đó m là kích thước của tập huấn luyện. Điều này có nghĩa là trung bình chỉ có khoảng 63% số mẫu huấn luyện được sử dụng cho mỗi bộ dự đoán.[6] 37% số mẫu huấn luyện còn lại không được sử dụng được gọi là các mẫu *out-of-bag* (oob). Lưu ý rằng con số 37% này không nhất thiết đúng cho tất cả các bộ dự đoán.

Vì không nhìn thấy các mẫu oob trong quá trình huấn luyện, nên bộ dự đoán có thể được đánh giá trên các mẫu này mà không cần một tập kiểm định riêng biệt. Ta có thể đánh giá một ensemble bằng cách lấy kết quả đánh giá oob trung bình của mỗi bộ dự đoán.

[6] Khi m tăng lên, tỉ lệ này tiến gần tới $1-\exp(-1) \approx 63.212\%$.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Trong Scikit-Learn, ta có thể đặt `oob_score=True` khi khởi tạo một thực thể `BaggingClassifier` để yêu cầu đánh giá oob tự động sau khi huấn luyện. Đoạn mã sau đây mô tả quy trình đó. Kết quả đánh giá được truy xuất thông qua biến `oob_score_`:

```
>>> bag_clf = BaggingClassifier(
...     DecisionTreeClassifier(), n_estimators=500,
...     bootstrap=True, n_jobs=-1, oob_score=True)
...
>>> bag_clf.fit(X_train, y_train)
>>> bag_clf.oob_score_
0.90133333333333332
```

Theo đánh giá oob trên, thực thể `BaggingClassifier` này có thể đạt độ chính xác tới 90.1% trên tập kiểm tra. Ta kiểm chứng kết quả trên như sau:

```
>>> from sklearn.metrics import accuracy_score
>>> y_pred = bag_clf.predict(X_test)
>>> accuracy_score(y_test, y_pred)
0.91200000000000003
```

Ta có được độ chính xác 91.2% trên tập kiểm tra, rất gần với kết quả của đánh giá oob.

Hàm quyết định oob cho mỗi mẫu huấn luyện cũng có thể được truy xuất thông qua biến `oob_decision_function_`. Trong trường hợp này, vì bộ ước lượng cơ sở có phương thức `predict_proba()`, hàm quyết định sẽ trả về xác suất các lớp cho mỗi mẫu. Ví dụ, đánh giá oob ước lượng rằng mẫu huấn luyện đầu tiên có 68.25% xác suất thuộc lớp dương (và 31.75% thuộc lớp âm):

```
>>> bag_clf.oob_decision_function_
array([[0.31746032, 0.68253968],
       [0.34117647, 0.65882353],
       [1.        , 0.        ],
       ...
       [1.        , 0.        ],
       [0.03108808, 0.96891192],
       [0.57291667, 0.42708333]])
```

Patch Ngẫu nhiên và Không gian con Ngẫu nhiên

Lớp `BaggingClassifier` cũng hỗ trợ lấy mẫu đặc trưng. Việc lấy mẫu này được kiểm soát bởi hai siêu tham số: `max_features` và `bootstrap_features`. Các siêu tham số này hoạt động tương tự như `max_samples` và `bootstrap`, nhưng cho việc lấy mẫu đặc trưng thay vì dữ liệu. Vì vậy, mỗi bộ dự đoán sẽ được huấn luyện trên một tập con ngẫu nhiên của các đặc trưng đầu vào.

Kỹ thuật này đặc biệt hữu ích trong trường hợp dữ liệu đầu vào có nhiều chiều (dữ liệu ảnh chẳng hạn). Việc lấy mẫu cả dữ liệu và đặc trưng được gọi là phương pháp *Patch Ngẫu nhiên*.[7] Giữ lại tất cả dữ liệu (đặt `bootstrap=False` và `max_samples=1.0`) nhưng lấy mẫu đặc trưng (đặt

[7] Gilles Louppe and Pierre Geurts, "Ensembles on Random Patches," *Lecture Notes in Computer Science* 7523 (2012): 346–361.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

`bootstrap_features=True` và/hoặc `max_features` nhỏ hơn `1.0`) được gọi là phương pháp *Không gian con Ngẫu nhiên (Random Subspaces)*.[8]

Lấy mẫu đặc trưng thậm chí còn giúp các bộ dự đoán trở nên đa dạng hơn bằng cách đánh đổi một chút độ chệch để có phương sai thấp hơn.

Rừng Ngẫu nhiên

Như đã đề cập, Rừng Ngẫu nhiên[9] là ensemble của các Cây Quyết định, thường được huấn luyện thông qua phương pháp bagging (hoặc đôi khi là pasting) với `max_samples` thường được đặt bằng với kích thước bộ huấn luyện. Thay vì phải tạo một `BaggingClassifier` và truyền vào một `DecisionTreeClassifier`, sử dụng lớp `RandomForestClassifier` sẽ tiện lợi và tối ưu hơn cho thuật toán Cây Quyết định[10] (tương tự, ta có lớp `RandomForestRegressor` dành cho những tác vụ hồi quy). Đoạn mã sau sử dụng tất cả lõi CPU sẵn có có để huấn luyện một bộ phân loại Rừng Ngẫu nhiên với 500 cây quyết định (mỗi cây có tối đa 16 nút):

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Trừ một số ngoại lệ, `RandomForestClassifier` có tất cả các siêu tham số của `DecisionTreeClassifier` (để kiểm soát cách các cây được phát triển) cùng với tất cả siêu tham số của `BaggingClassifier`.[11]

Ngoài ra, Rừng Ngẫu nhiên còn đưa thêm sự ngẫu nhiên vào quá trình xây dựng cây; thay vì tìm đặc trưng tốt nhất khi tách một nút (tham khảo Chương 6), thuật toán này tìm đặc trưng tốt nhất trong một tập đặc trưng con ngẫu nhiên. Kết quả ta sẽ có các cây đa dạng hơn, (một lần nữa) bằng cách đánh đổi độ chệch cao hơn để thu về phương sai thấp hơn, từ đó tạo ra mô hình ensemble tốt hơn. `BaggingClassifier` dưới đây gần như tương đương với `RandomForestClassifier` ở trên:

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(splitter="random", max_leaf_nodes=16),
    n_estimators=500, max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Cây Siêu Ngẫu nhiên

Khi phát triển cây trong Rừng Ngẫu nhiên, tại mỗi nút ta chỉ xét một tập đặc trưng con ngẫu nhiên để tách nút đó (như đã thảo luận ở trên). Ta có thể làm cho các cây trở nên ngẫu nhiên hơn bằng cách sử dụng thêm các ngưỡng ngẫu nhiên cho mỗi đặc trưng thay vì tìm kiếm các ngưỡng tốt nhất có thể (như các Cây Quyết định thông thường).

[8] Tin Kam Ho, "The Random Subspace Method for Constructing Decision Forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, no. 8 (1998): 832–844.

[9] Tin Kam Ho, "Random Decision Forests," *Proceedings of the Third International Conference on Document Analysis and Recognition* 1 (1995): 278.

[10] Lớp `BaggingClassifier` vẫn hữu ích nếu ta muốn một tập hợp các mô hình không chỉ gồm các Cây Quyết định.

[11] Một vài ngoại lệ đáng lưu ý: không có `splitter` (buộc phải là `"random"`), không có `presort` (buộc phải là `False`), không có `max_samples` (buộc phải bằng 1.0) và không có `base_estimator` (buộc phải là `DecisionTreeClassifier` với các siêu tham số được cung cấp).

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Một rừng chứa những cây cực kỳ ngẫu nhiên như vậy được gọi là một ensemble của các *Cây Siêu Ngẫu nhiên (Extremely Randomized Trees)*[12] (trong tiếng Anh được viết gọn là *Extra-Trees*). Một lần nữa, kỹ thuật này đánh đổi độ chệch cao hơn để có phương sai thấp hơn. Huấn luyện Cây Siêu Ngẫu nhiên cũng nhanh hơn nhiều so với Rừng Ngẫu nhiên thông thường, vì tìm ngưỡng tốt nhất cho mỗi đặc trưng ở mỗi nút là một trong những tác vụ mất nhiều thời gian nhất khi xây dựng cây.

Ta có thể sử dụng lớp `ExtraTreesClassifier` trong Scikit-Learn để tạo bộ phân loại Cây Siêu Ngẫu nhiên. API của lớp này giống với lớp `RandomForestClassifier`. Tương tự, lớp `ExtraTreesRegressor` cũng có API giống với lớp `RandomForestRegressor`.

Mẹo

Rất khó để nói trước liệu `RandomForestClassifier` sẽ hoạt động tốt hơn hay kém hơn `ExtraTreesClassifier`. Thông thường, cách duy nhất để biết là thử cả hai và so sánh chúng thông qua kiểm định chéo (và tinh chỉnh các siêu tham số bằng thuật toán tìm kiếm dạng lưới).

Độ Quan trọng của Đặc trưng

Một ưu điểm khác của Rừng Ngẫu nhiên là dễ dàng đo lường được độ quan trọng tương đối của mỗi đặc trưng. Scikit-Learn tính toán độ quan trọng của đặc trưng bằng cách xét xem các nút sử dụng đặc trưng đó có thể giảm độ pha tạp đi bao nhiêu (lấy trung bình trên tất cả các cây trong rừng). Chính xác hơn, đó là trung bình có trọng số, trong đó trọng số của mỗi nút bằng số lượng mẫu huấn luyện trong nút đó (tham khảo Chương 6).

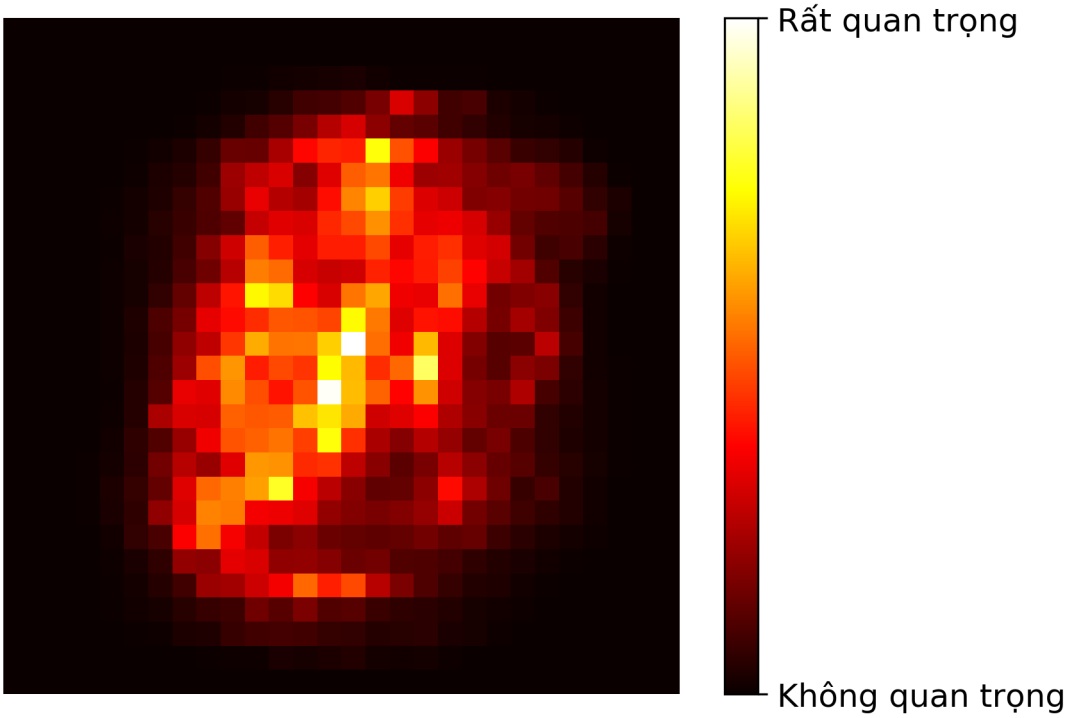
Scikit-Learn tính độ quan trọng này một cách tự động cho mỗi đặc trưng sau khi huấn luyện, tiếp đó co giãn kết quả sao cho tổng của tất cả độ quan trọng bằng 1. Ta có thể thu được kết quả này thông qua thuộc tính `feature_importances_`. Ví dụ, đoạn mã sau huấn luyện một bộ phân loại `RandomForestClassifier` trên tập dữ liệu Iris (đã được giới thiệu ở Chương 4) và đưa ra độ quan trọng cho mỗi đặc trưng. Dường như những đặc trưng quan trọng nhất là độ dài (44%) và độ rộng (42%) của cánh hoa, trong khi độ dài và độ rộng của đài hoa không quan trọng lắm (có giá trị tương ứng là 11% và 2%):

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Tương tự, khi huấn luyện bộ phân loại Rừng Ngẫu nhiên trên tập dữ liệu MNIST (được giới thiệu ở Chương 3) và vẽ đồ thị độ quan trọng của mỗi điểm ảnh, ta sẽ thu được ảnh trong Hình 7.6.

[12] Pierre Geurts et al., "Extremely Randomized Trees," *Machine Learning* 63, no. 1 (2006): 3–42.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 7.6. Độ quan trọng của các điểm ảnh trong MNIST (theo bộ phân loại Rừng Ngẫu nhiên)

Rừng Ngẫu nhiên rất tiện lợi trong việc nắm bắt nhanh những đặc trưng thực sự quan trọng, đặc biệt khi ta cần trích chọn đặc trưng.

Boosting

Boosting (ban đầu được gọi là *boosting giả thuyết – hypothesis boosting*) là bất cứ phương pháp Ensemble nào có thể kết hợp một vài bộ học yếu thành một bộ học mạnh. Ý tưởng chung của hầu hết các phương pháp boosting là huấn luyện tuần tự các bộ dự đoán, bộ dự đoán sau cố gắng sửa lỗi cho bộ dự đoán trước. Có nhiều phương pháp boosting nhưng hiện nay phổ biến nhất là *AdaBoost*[13] (viết tắt của *Adaptive Boosting* – tạm dịch: *Boosting Thích ứng*) và *Gradient Boosting*. Bây giờ, hãy bắt đầu với AdaBoost!

AdaBoost

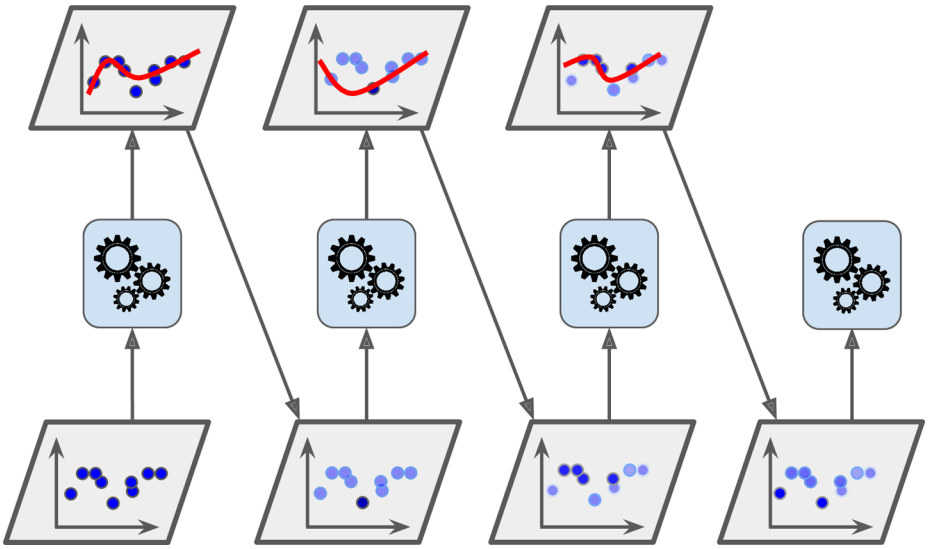
Một cách để bộ dự đoán sau có thể sửa lỗi cho bộ dự đoán trước là tập trung hơn vào các mẫu huấn luyện mà bộ dự đoán trước bị dưới khớp. Điều này giúp bộ dự đoán sau chú ý vào những trường hợp khó. Kỹ thuật này có tên gọi là AdaBoost.

Ví dụ, khi huấn luyện một bộ phân loại AdaBoost, đầu tiên thuật toán huấn luyện một bộ phân loại cơ sở (như Cây Quyết định) và sử dụng bộ phân loại này để dự đoán trên tập huấn luyện. Sau đó thuật toán sẽ tăng trọng số của các mẫu huấn luyện bị dự đoán sai. Tiếp theo thuật toán huấn luyện bộ phân loại thứ hai, sử dụng trọng số được cập nhật, sau đó lặp lại việc dự

[13] Yoav Freund and Robert E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences* 55, no. 1 (1997): 119–139.

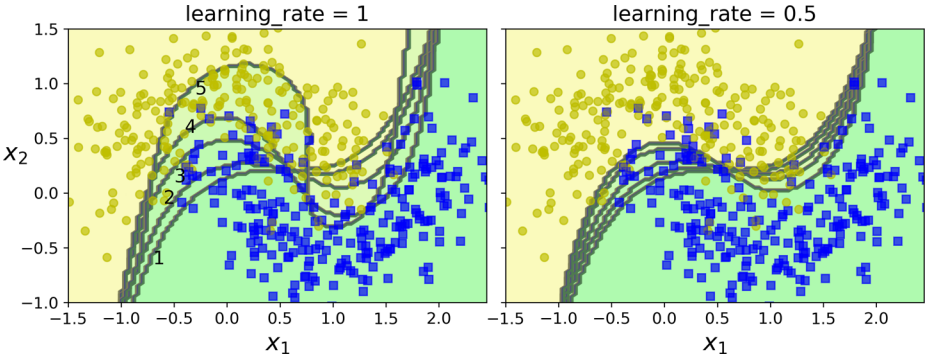
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

đoán trên tập huấn luyện, cập nhật trọng số cho mẫu dữ liệu và cứ tiếp tục như thế (tham khảo Hình 7.7).



Hình 7.7. Huấn luyện tuần tự AdaBoost với trọng số mẫu được cập nhật

Hình 7.8 biểu diễn ranh giới quyết định của năm bộ dự đoán liên tiếp trên tập dữ liệu moons (trong ví dụ này, mỗi bộ dự đoán là một bộ phân loại SVM được điều chuẩn mạnh với hạt nhân RBF[14]). Bộ phân loại đầu tiên dự đoán sai nhiều mẫu và trọng số của những mẫu đó được tăng lên. Vì vậy, bộ phân loại thứ hai đạt được kết quả tốt hơn trên những mẫu dữ liệu này, và tương tự như vậy cho các bộ phân loại sau. Biểu đồ bên phải minh họa cùng một chuỗi các bộ dự đoán nhưng với tốc độ học giảm một nửa (tức trọng số của các mẫu bị phân loại sai tăng chậm một nửa tại mỗi vòng lặp). Có thể thấy, kỹ thuật học tuần tự này có một vài điểm tương đồng với Hạ Gradient, trừ việc thay vì điều chỉnh các tham số của một bộ dự đoán riêng lẻ để cực tiểu hóa hàm chi phí, AdaBoost kết hợp nhiều bộ dự đoán thành ensemble, dần dần cải thiện kết quả dự đoán tốt hơn.



Hình 7.8. Ranh giới quyết định của các bộ dự đoán liên tiếp

Một khi huấn luyện xong, cách mà ensemble đưa ra các dự đoán rất giống với phương pháp bagging hoặc pasting, ngoại trừ việc các bộ dự đoán có trọng số khác nhau tuỳ thuộc vào độ chính xác tổng thể trên tập huấn luyện đã đánh trọng số.

[14] Việc này chỉ mang tính minh họa. Thường thì SVM không phải là bộ dự đoán cơ sở tốt cho AdaBoost vì SVM chậm và có xu hướng không ổn định khi được dùng với thuật toán này.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lưu ý

Một điểm trừ lớn của kỹ thuật học tuần tự này là không thể học song song (hoặc chỉ có thể song song hóa một phần). Ta chỉ có thể huấn luyện bộ dự đoán mới sau khi hoàn thành quá trình huấn luyện và đánh giá bộ dự đoán trước đó. Do đó, kỹ thuật này không mở rộng tốt bằng phương pháp bagging và pasting.

Hãy cùng tìm hiểu kỹ hơn thuật toán AdaBoost. Ban đầu, trọng số của mỗi mẫu $w^{(i)}$ được khởi tạo bằng $1/m$. Bộ dự đoán đầu tiên được huấn luyện, và tỉ lệ lỗi có trọng số r_1 được tính trên tập huấn luyện (tham khảo Phương trình 7.1).

$$r_j = \frac{\sum_{\hat{y}_j^{(i)} \neq y^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}} \quad \text{trong đó } \hat{y}_j^{(i)} \text{ là dự đoán thứ } j \text{ của bộ dự đoán cho mẫu thứ } i.$$

Phương trình 7.1. Tỉ lệ lỗi có trọng số của bộ dự đoán thứ j

Trọng số α_j của bộ dự đoán sau đó được tính theo Phương trình 7.2, trong đó η là tốc độ học (mặc định bằng 1).[15] Bộ dự đoán càng chính xác, trọng số của nó càng cao. Nếu chỉ dự đoán ngẫu nhiên, trọng số sẽ gần 0. Tuy nhiên, nếu dự đoán sai nhiều (tức độ chính xác thấp hơn cả khi dự đoán ngẫu nhiên), thì trọng số sẽ âm.

$$\alpha_j = \eta \times log \frac{1 - r_j}{r_j}$$

Phương trình 7.2. Trọng số bộ dự đoán

Sau đó, thuật toán AdaBoost cập nhật trọng số của mẫu theo Phương trình 7.3. Công thức này sẽ tăng trọng số của các mẫu bị phân loại sai.

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{nếu } \hat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{nếu } \hat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$
$$\text{với } i = 1, 2, \cdots, m$$

Phương trình 7.3. Quy tắc cập nhật trọng số

Sau đó, tất cả trọng số của mẫu được chuẩn hóa (chia cho $\sum_{i=1}^{m} w^{(i)}$).

Cuối cùng, một bộ dự đoán mới được huấn luyện với các trọng số đã cập nhật, và toàn bộ quá trình được lặp lại (tính trọng số của bộ dự đoán mới, cập nhật trọng số của mẫu, huấn luyện một bộ dự đoán khác, và cứ như vậy). Thuật toán dừng lại khi đạt đến số lượng bộ dự đoán mong muốn, hoặc khi tìm được một bộ dự đoán hoàn hảo.

Khi dự đoán, AdaBoost chỉ đơn giản tính đầu ra của từng bộ dự đoán, rồi gán thêm trọng số α_j của bộ dự đoán đó. Nhãn lớp dự đoán là nhãn có tổng trọng số lớn nhất (tham khảo Phương trình 7.4).

[15] Thuật toán AdaBoost gốc không sử dụng tốc độ học.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$\hat{y}(\mathbf{x}) = \underset{k}{argmax} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j \quad \text{trong đó } N \text{ là số bộ dự đoán.}$$

Phương trình 7.4. AdaBoost dự đoán

Scikit-Learn sử dụng phiên bản đa lớp của AdaBoost, được gọi là *SAMME*[16] (viết tắt của *Stagewise Additive Modeling với hàm mất mát Multiclass Exponential*). Khi chỉ có hai lớp, SAMME sẽ tương đương với AdaBoost. Nếu các bộ dự đoán có thể ước lượng xác suất (tức có phương thức `predict_proba()`), Scikit-Learn có thể sử dụng một biến thể của SAMME, đó là *SAMME.R* (*R* là viết tắt của "Real"). Biến thể này dựa trên xác suất các lớp thay vì dự đoán đưa ra và nhìn chung hoạt động tốt hơn.

Đoạn mã dưới đây huấn luyện bộ phân loại AdaBoost dựa trên 200 *Gốc cây Quyết định* sử dụng lớp `AdaBoostClassifier` trong Scikit-Learn (tương tự, cũng có lớp `AdaBoostRegressor`). Gốc cây Quyết định là một Cây Quyết định có `max_depth=1`, hay nói cách khác, là cây bao gồm một nút quyết định duy nhất cùng với hai nút lá. Đây là bộ ước lượng cơ sở mặc định của lớp `AdaBoostClassifier`:

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5)
ada_clf.fit(X_train, y_train)
```

Mẹo

Nếu AdaBoost quá khớp tập huấn luyện, bạn có thể thử giảm số bộ ước lượng, hay tăng điều chuẩn bộ ước lượng cơ sở.

Gradient Boosting

Một thuật toán boosting phổ biến khác đó là *Gradient Boosting*.[17] Cũng giống AdaBoost, Gradient Boosting hoạt động bằng cách lần lượt thêm các bộ dự đoán vào ensemble, mỗi bộ sửa lỗi cho bộ trước đó. Tuy nhiên, thay vì tác động vào trọng số của mẫu tại tất cả các vòng lặp như AdaBoost, phương pháp này cố gắng khớp bộ dự đoán mới vào *sai số phần dư* (*residual error*) của bộ dự đoán trước đó.

Hãy cùng xem một ví dụ hồi quy đơn giản, sử dụng Cây Quyết định làm bộ dự đoán cơ sở (tất nhiên, Gradient Boosting cũng hoạt động tốt với các tác vụ phân loại). Đây được gọi là *Gradient Tree Boosting*, hoặc *Gradient Boosted Regression Trees* (GBRT). Đầu tiên, hãy khớp `DecisionTreeRegressor` vào tập huấn luyện (ví dụ, tập huấn luyện bậc hai có nhiễu):

```
from sklearn.tree import DecisionTreeRegressor
```

[16] Đọc "Multi-Class AdaBoost," *Statistics and Its Interface* 2, no. 3 (2009): 349–360 của Ji Zhu và cộng sự, để biết thêm chi tiết.

[17] Gradient Boosting được giới thiệu lần đầu trong bài báo "Arcing the Edge" của Leo Breiman và được phát triển sâu hơn trong bài báo năm 1999 "Greedy Function Approximation: A Gradient Boosting Machine" của Jerome H. Friedman.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)
```

Tiếp theo, ta huấn luyện `DecisionTreeRegressor` thứ hai trên sai số phần dư của bộ hồi quy đầu tiên:

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Sau đó huấn luyện bộ hồi quy thứ ba trên sai số phần dư của bộ thứ hai:

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Giờ ta đã có một ensemble chứa ba cây. Ensemble này có thể dự đoán trên mẫu mới đơn giản bằng cách tính tổng dự đoán của tất cả các cây.

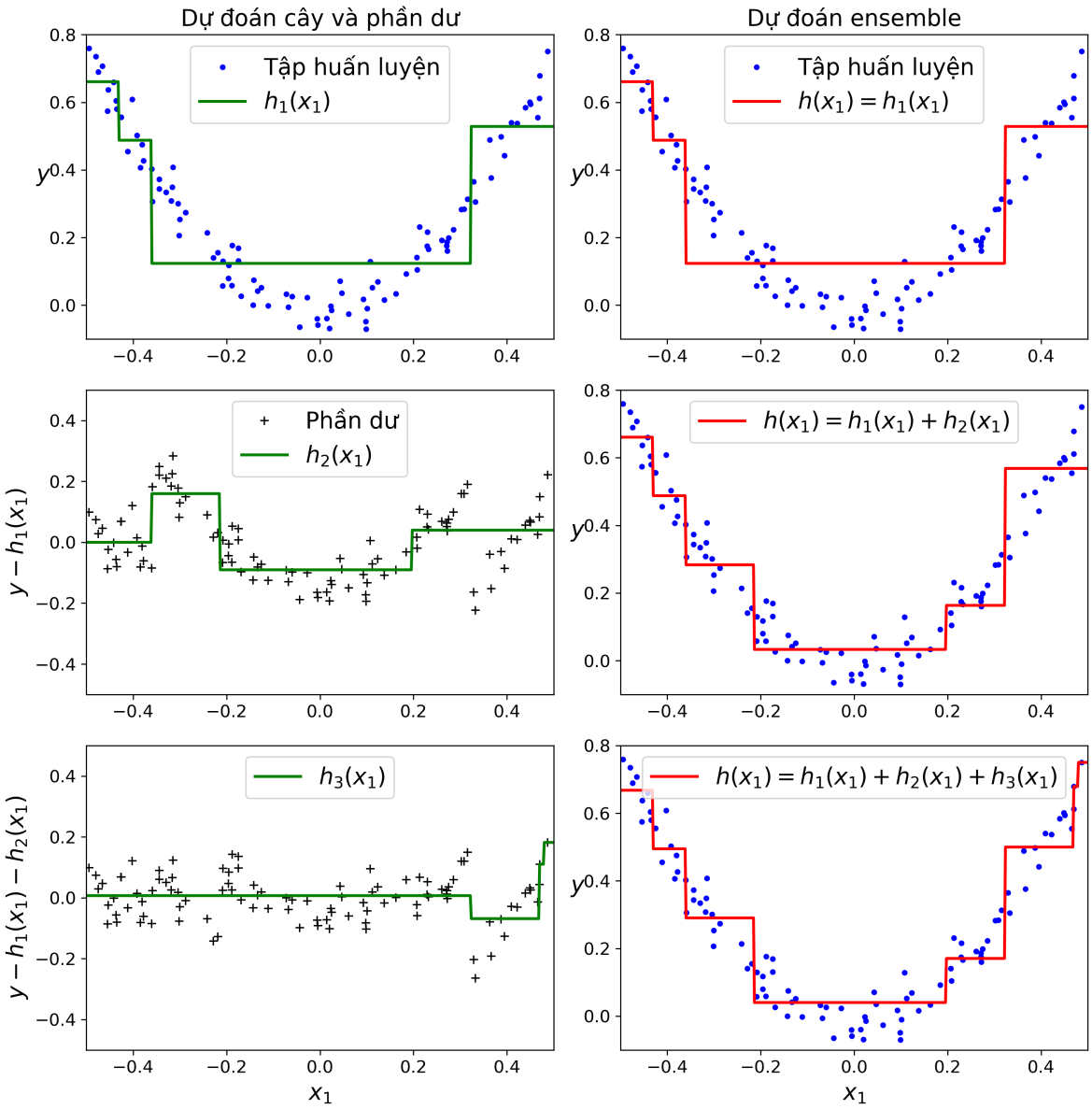
```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

Hình 7.9 biểu diễn các dự đoán của ba cây trên ở cột bên trái và các dự đoán của ensemble ở cột bên phải. Ở hàng đầu tiên, ensemble chỉ có một cây nên có dự đoán giống hệt dự đoán của cây thứ nhất. Hàng thứ hai có một cây mới được huấn luyện trên sai số phần dư của cây thứ nhất. Ở phía bên phải ta có thể thấy dự đoán của ensemble bằng tổng các dự đoán của hai cây. Tương tự, ở hàng thứ ba có một cây khác được huấn luyện trên sai số phần dư của cây thứ hai. Có thể thấy rằng dự đoán của ensemble dần được cải thiện khi nhiều cây được thêm vào.

Một cách đơn giản hơn để huấn luyện ensemble GBRT là sử dụng lớp `GradientBoostingRegressor` của Scikit-Learn. Tương tự như lớp `RandomForestRegressor`, lớp này có các siêu tham số điều khiển tốc độ phát triển của Cây Quyết định (như `max_depth`, `min_samples_leaf`), và điều khiển việc huấn luyện ensemble, như số lượng cây (`n_estimators`). Đoạn mã sau tạo một ensemble giống hệt như phía trên:

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)
gbrt.fit(X, y)
```

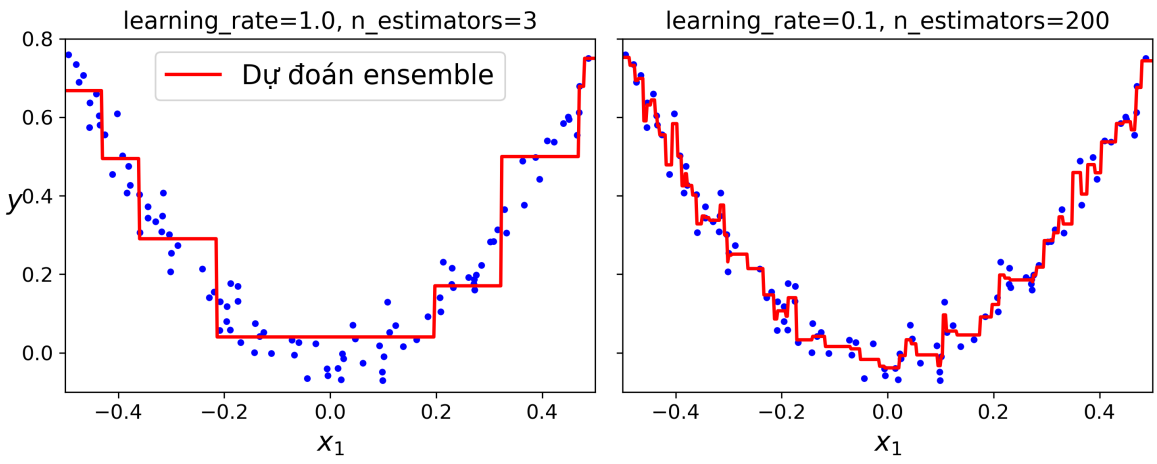
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 7.9. Minh hoạ Gradient Boosting: bộ dự đoán thứ nhất (hình trên cột trái) được huấn luyện bình thường, tiếp theo các bộ dự đoán sau đó (hình giữa và dưới cột trái) được huấn luyện trên sai số phần dư của bộ dự đoán trước đó; cột bên phải biểu diễn dự đoán của ensemble được tạo ra

Siêu tham số `learning_rate` điều khiển sự đóng góp của các cây. Nếu giá trị này thấp, chẳng hạn như bằng 0.1, ensemble sẽ cần nhiều cây hơn để khớp tập huấn luyện, nhưng thường sẽ khái quát hóa tốt hơn. Đây là một kỹ thuật điều chuẩn gọi là *shrinkage*. Hình 7.10 biểu diễn hai ensemble GBRT được huấn luyện với tốc độ học thấp: mô hình bên trái không có đủ cây để khớp tập huấn luyện, trong khi mô hình bên phải có quá nhiều cây và bị quá khớp.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



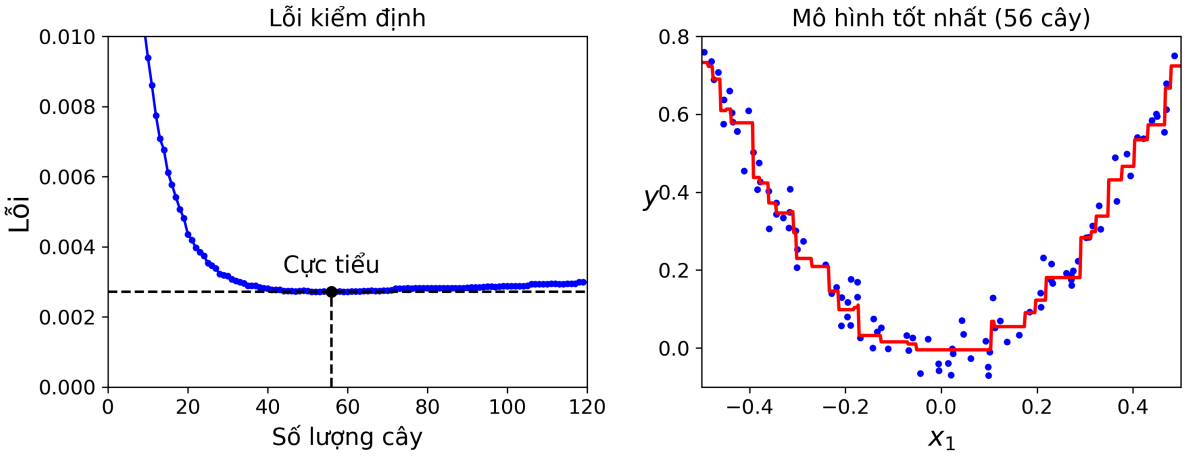
Hình 7.10. Ensemble GBRT không có đủ (trái) và có quá nhiều (phải) bộ dự đoán

Để tìm số lượng cây tối ưu, ta có thể sử dụng kỹ thuật dừng sớm (tham khảo Chương 4). Một cách đơn giản để thực hiện kỹ thuật này là sử dụng phương thức `staged_predict()`: phương thức này trả về tất cả dự đoán của ensemble tại các giai đoạn huấn luyện (khi ensemble có một cây, hai cây, v.v.) Đoạn mã dưới đây huấn luyện một ensemble GBRT với 120 cây, sau đó tính lỗi kiểm định tại mỗi giai đoạn huấn luyện để tìm số lượng cây tối ưu, và cuối cùng huấn luyện một ensemble GBRT khác với số cây tối ưu đó:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2,n_estimators=bst_n_estimators)
gbrt_best.fit(X_train, y_train)
```

Lỗi kiểm định được biểu diễn ở biểu đồ bên trái trong Hình 7.11, và dự đoán của mô hình tốt nhất được biểu diễn ở biểu đồ bên phải.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 7.11. Tinh chỉnh số lượng cây sử dụng dừng sớm

Ta cũng có thể thực hiện dừng sớm bằng cách thật sự dừng huấn luyện sớm (thay vì huấn luyện một lượng lớn cây rồi quay lại tìm số cây tối ưu). Bằng cách đặt `warm_start=True`, Scikit-Learn sẽ giữ lại các cây đang có khi phương thức `fit()` được gọi, cho phép ta huấn luyện gia tăng. Đoạn mã sau dừng huấn luyện khi lỗi kiểm định không được cải thiện trong 5 vòng lặp liên tiếp:

```
gbrt = GradientBoostingRegressor(max_depth=2, warm_start=True)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1, 120):
    gbrt.n_estimators = n_estimators
    gbrt.fit(X_train, y_train)
    y_pred = gbrt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break  # early stopping
```

Lớp `GradientBoostingRegressor` cũng hỗ trợ siêu tham số `subsample` để chỉ định tỷ lệ mẫu huấn luyện được sử dụng khi huấn luyện mỗi cây. Ví dụ, nếu `subsample=0.25`, mỗi cây sẽ được huấn luyện trên 25% số mẫu được lựa chọn ngẫu nhiên từ tập huấn luyện. Có thể nói rằng kỹ thuật này đánh đổi độ chệch lớn để có phương sai nhỏ, và cũng tăng tốc huấn luyện đáng kể. Đây được gọi là *Gradient Boosting Ngẫu nhiên (Stochastic Gradient Boosting)*.

Ghi chú

Có thể sử dụng Gradient Boosting với các hàm chi phí khác bằng cách chỉnh siêu tham số `loss` (xem tài liệu của Scikit-Learn để biết thêm chi tiết).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Một điều đáng đề cập là mã nguồn đã được tối ưu của Gradient Boosting hiện có sẵn ở thư viện Python nổi tiếng XGBoost, viết tắt của Extreme Gradient Boosting. Thư viện này ban đầu được phát triển bởi Tianqi Chen, thuộc Cộng đồng Học Máy (Sâu) Phân tán – *Distributed (Deep) Machine Learning Community (DMLC)*, nhắm tới tốc độ, tính mở rộng và di động cực cao. Trên thực tế, XGBoost thường là một mô hình quan trọng trong nhiều thắng lợi tại các cuộc thi Học Máy. API của XGBoost cũng khá giống với API của Scikit-Learn:

```
import xgboost

xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)
```

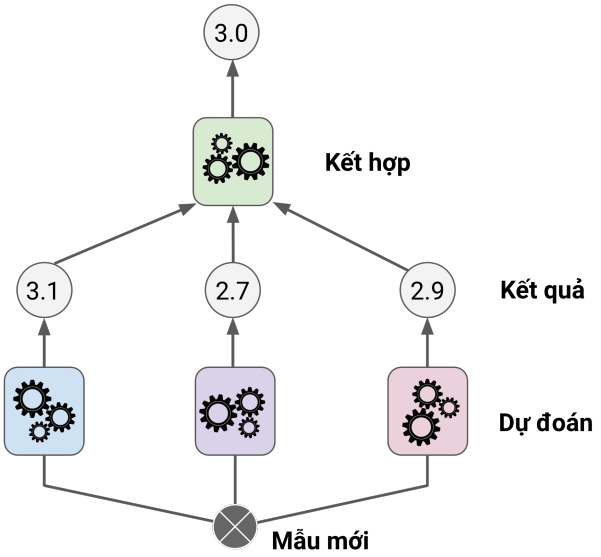
XGBoost cũng cung cấp một số tính năng thú vị, chẳng hạn như dừng sớm:

```
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)], early_stopping_rounds=2)
y_pred = xgb_reg.predict(X_val)
```

Chúng tôi khuyến khích độc giả dùng thử thư viện này!

Stacking

Phương pháp Ensemble cuối cùng được trình bày trong chương này là *stacking* (viết tắt của *stacked generalization – khái quát hóa theo stack*).[18] Phương pháp dựa trên ý tưởng rất đơn giản: thay vì sử dụng các hàm đơn giản (như biểu quyết cứng), một mô hình được huấn luyện để kết hợp đầu ra của tất cả các bộ dự đoán trong ensemble. Hình 7.12 mô tả một ensemble thực hiện tác vụ hồi quy trên một mẫu mới. Mỗi bộ dựa đoán đưa ra một giá trị khác nhau (3.1, 2.7, và 2.9), và bộ dự đoán cuối cùng (gọi là *bộ kết hợp – blender*, hay *bộ học meta – meta learner*) nhận các giá trị dự đoán đó làm đầu vào và đưa ra dự đoán cuối cùng (3.0).

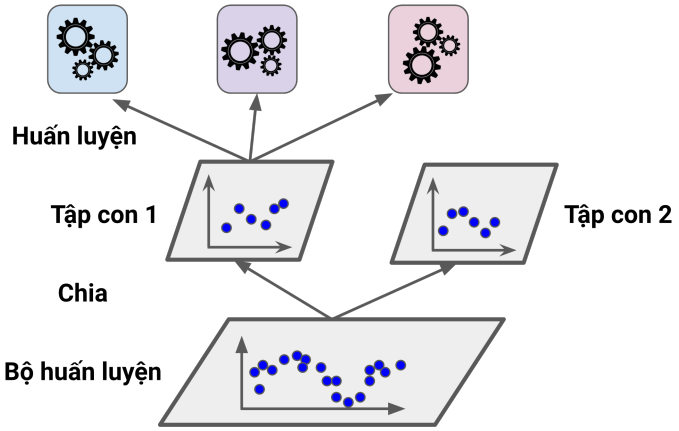


Hình 7.12. Kết hợp các dự đoán bằng cách sử dụng bộ dự đoán kết hợp

[18] David H. Wolpert, "Stacked Generalization," *Neural Networks* 5, no. 2 (1992): 241–259.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Để huấn luyện bộ kết hợp, cách phổ biến đó là sử dụng tập giữ lại (*hold-out*).[19] Đầu tiên, tập huấn luyện được chia thành hai tập con. Tập đầu tiên được sử dụng để huấn luyện các bộ phân loại trong tầng thứ nhất (xem Hình 7.13).

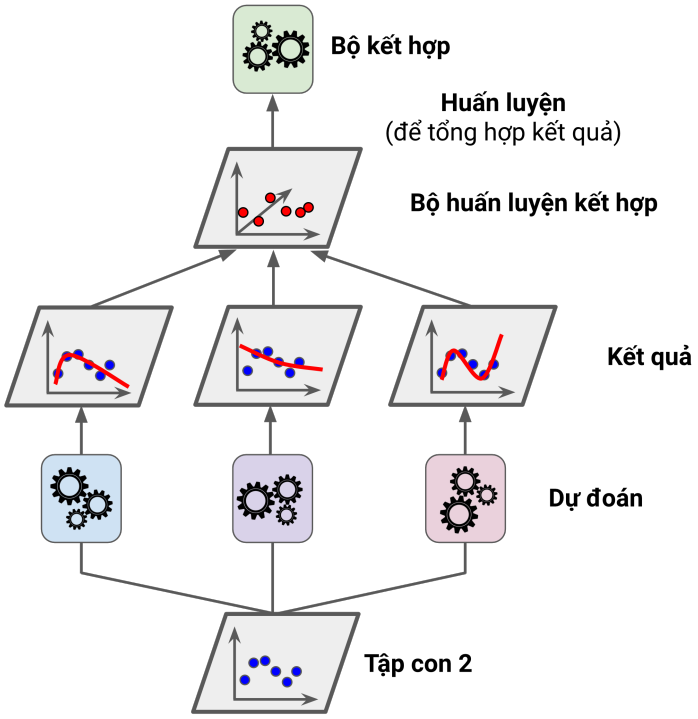


Hình 7.13. Huấn luyện tầng thứ nhất

Tiếp đến, các bộ phân loại trong tầng thứ nhất đưa ra dự đoán cho các mẫu trong tập còn lại (tức tập giữ lại, xem Hình 7.14). Điều này đảm bảo rằng các dự đoán là "sạch", vì các bộ dự đoán không nhìn thấy các mẫu dữ liệu này trong quá trình huấn luyện. Với mỗi mẫu trong tập giữ lại, sẽ có 3 giá trị dự đoán. Ta có thể tạo ra một tập huấn luyện mới bằng cách thay thế các đặc trưng cũ trong tập giữ lại bằng các giá trị dự đoán (tập huấn luyện mới lúc này có đặc trưng 3 chiều), và giữ lại các giá trị mục tiêu. Bộ kết hợp được huấn luyện trên tập dữ liệu mới này, học cách dự đoán giá trị mục tiêu khi biết trước các dự đoán của tầng đầu tiên.

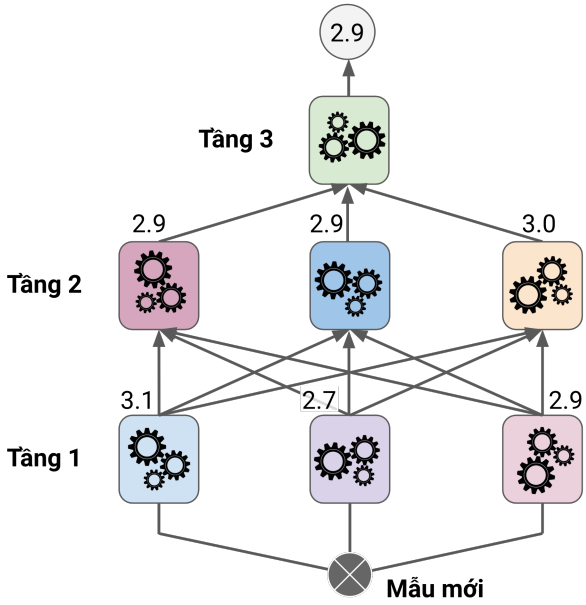
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

[19] Ngoài ra, cũng có thể sử dụng kỹ thuật dự đoán out-of-fold. Trong một vài ngữ cảnh, cách này được gọi là *stacking*, trong khi cách sử dụng tập giữ lại được gọi là *blending*. Nhiều người coi hai thuật ngữ này là một.



Hình 7.14. Huấn luyện bộ kết hợp

Thực tế, ta có thể huấn luyện các bộ kết hợp khác nhau theo cách này (ví dụ, Hồi quy Tuyến tính, Rừng Ngẫu nhiên, v.v.), để thu được một tầng các bộ kết hợp. Thủ thuật ở đây là chia tập huấn luyện thành ba tập con: tập đầu tiên được sử dụng để huấn luyện tầng đầu tiên, tập thứ hai được sử dụng để huấn luyện tầng thứ hai (sử dụng các dự đoán từ các bộ dự đoán ở tầng đầu tiên), và tập thứ ba được sử dụng để huấn luyện tầng thứ ba (sử dụng các dự đoán từ các bộ dự đoán ở tầng thứ hai). Khi thực hiện xong, ta có thể đưa ra dự đoán cho một mẫu mới bằng cách lần lượt đi qua từng tầng, như minh hoạ trong Hình 7.15.



Hình 7.15. Dự đoán của ensemble stacking đa tầng

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Không may, Scikit-Learn chưa hỗ trợ stacking một cách trực tiếp, nhưng nó cũng không quá khó để tự lập trình (xem bài tập phía dưới). Một phương án khác là sử dụng các thư viện mã nguồn mở như `DESlib`.

Bài tập

1. Giả sử ta huấn luyện 5 mô hình khác nhau trên cùng một tập huấn luyện và tất cả các mô hình đều đạt precision 95%. Liệu ta có thể kết hợp các mô hình để đạt được kết quả cao hơn không? Nếu có, hãy mô tả cách kết hợp đó. Nếu không, hãy giải thích tại sao?

2. Điểm khác biệt giữa bộ phân loại biểu quyết cứng và mềm là gì?

3. Liệu ta có thể tăng tốc quá trình huấn luyện ensemble bagging bằng cách chạy phân tán trên nhiều máy chủ được không? Các ensemble khác như pasting, boosting, Rừng Ngẫu nhiên, hay stacking thì sao?

4. Phương pháp đánh giá out-of-bag có những ưu điểm gì?

5. Điều gì khiến Cây Siêu Ngẫu nhiên có tính ngẫu nhiên cao hơn Rừng Ngẫu nhiên thông thường? Sự ngẫu nhiên bổ sung này có ích như thế nào? Cây Siêu Ngẫu nhiên chậm hay nhanh hơn Rừng Ngẫu nhiên thông thường?

6. Nếu ensemble AdaBoost dưới khớp tập huấn luyện, ta có thể điều chỉnh những siêu tham số nào và điều chỉnh như thế nào?

7. Nếu ensemble Gradient Boosting quá khớp tập huấn luyện, thì ta nên tăng hay giảm tốc độ học?

8. Nạp tập dữ liệu MNIST (được giới thiệu trong Chương 3), và chia thành tập huấn luyện, tập kiểm định và tập kiểm tra (ví dụ, tập huấn luyện có 50,000 mẫu, tập kiểm định có 10,000 mẫu, và tập kiểm tra có 10,000 mẫu). Sau đó, huấn luyện các bộ phân loại khác nhau, như Rừng Ngẫu nhiên, Cây Siêu Ngẫu nhiên, và SVM. Tiếp theo, hãy kết hợp chúng theo cách biểu quyết cứng hoặc mềm để thu được một ensemble có chất lượng tốt hơn từng bộ phân loại riêng lẻ trên tập kiểm định. Ensemble này có chất lượng trên tập kiểm tra tốt hơn bao nhiêu khi so sánh với từng bộ phân loại riêng lẻ?

9. Chạy từng bộ phân loại riêng lẻ trong bài tập trước để đưa ra các dự đoán trên tập kiểm định. Sau đó, tạo một tập huấn luyện mới với các dự đoán sau: mỗi mẫu huấn luyện gồm một vector chứa các dự đoán từ tất cả các bộ phân loại riêng lẻ cho một ảnh, và nhãn vẫn là nhãn của ảnh đó. Huấn luyện một bộ phân loại trên tập huấn luyện mới này. Tới đây, bạn đã huấn luyện xong một bộ kết hợp, và cùng với các bộ phân loại, bạn đã có một ensemble stacking hoàn chỉnh. Bây giờ, hãy đánh giá chất lượng của ensemble trên tập kiểm tra. Với mỗi ảnh trong tập kiểm tra, hãy đưa ra dự đoán của tất cả các bộ phân loại riêng lẻ, sau đó đưa chúng qua bộ kết hợp để thu được dự đoán của ensemble. Hãy so sánh ensemble stacking này với ensemble biểu quyết được huấn luyện ở bài tập trước.

Lời giải cho các bài tập trên được cung cấp trong Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 8

Giảm Chiều

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Nhiều bài toán Học Máy đòi hỏi ta phải xử lý hàng nghìn hoặc thậm chí hàng triệu đặc trưng trên mỗi mẫu. Việc có nhiều đặc trưng như vậy không chỉ làm chậm đáng kể quá trình huấn luyện mà còn khiến cho việc tìm nghiệm tốt khó khăn hơn rất nhiều. Vấn đề này thường được gọi là *lời nguyền chiều* (*curse of dimensionality*).

May mắn thay, trong nhiều bài toán thực tế, ta có thể giảm đáng kể số lượng đặc trưng, từ đó biến bài toán bất khả tính thành khả tính. Ví dụ, xét các ảnh trong tập dữ liệu MNIST (được giới thiệu trong Chương 3): hầu hết các điểm ảnh ở viền đều có màu trắng, do đó ta có thể loại bỏ hoàn toàn các điểm ảnh này ra khỏi tập huấn luyện mà không để mất quá nhiều thông tin. Hình 7.6 một lần nữa xác nhận những điểm ảnh nói trên không hề quan trọng đối với tác vụ phân loại. Ngoài ra, hai điểm ảnh lân cận thường có độ tương quan cao: nếu ta hợp nhất chúng thành một (ví dụ bằng cách lấy trung bình cường độ điểm ảnh), thì lượng thông tin mất đi cũng sẽ không đáng kể.

Lưu ý

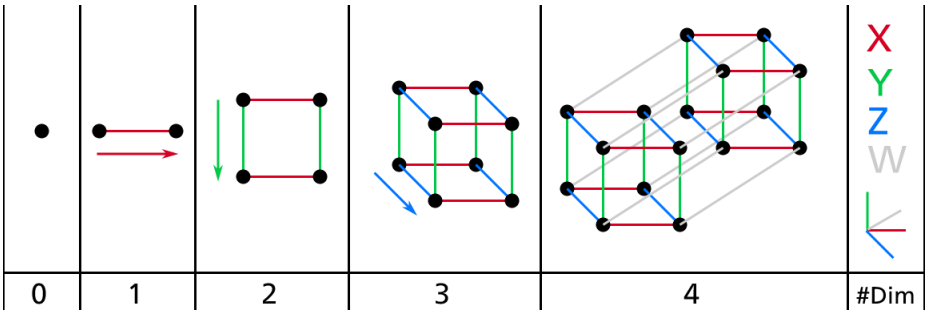
Việc giảm chiều sẽ làm mất một lượng thông tin nhất định (tương tự như việc nén ảnh thành định dạng JPEG sẽ làm giảm chất lượng ảnh), vì vậy mặc dù phương pháp này có thể tăng tốc độ huấn luyện, chất lượng hệ thống có thể sẽ giảm đi đôi chút. Việc giảm chiều cũng khiến cho pipeline trở nên phức tạp và khó bảo trì hơn. Do đó nếu quá trình huấn luyện diễn ra không quá chậm, ta nên thử huấn luyện hệ thống với dữ liệu gốc trước khi tính đến việc giảm chiều. Trong một vài trường hợp, việc giảm chiều của dữ liệu huấn luyện có thể lọc nhiễu và loại bỏ các chi tiết không cần thiết, từ đó tăng chất lượng của hệ thống. Nhưng nhìn chung các trường hợp đó rất hiếm, và phương pháp này sẽ chỉ tăng tốc quá trình huấn luyện.

Ngoài việc tăng tốc độ huấn luyện, giảm chiều cũng rất hữu ích cho tác vụ trực quan hóa dữ liệu (*data visualization* hay *DataViz*). Việc giảm xuống còn hai (hoặc ba) chiều giúp ta vẽ được một biểu đồ cô đọng của một tập huấn luyện nhiều chiều, từ đó cung cấp cho ta những thông tin quan trọng thông qua việc phát hiện các khuôn mẫu (ví dụ các cụm dữ liệu) một cách trực quan. Hơn nữa, việc trực quan hóa dữ liệu cũng rất cần thiết trong việc giải thích kết quả cho những người không làm về khoa học dữ liệu – cụ thể là những người sẽ đưa ra quyết định dựa trên kết quả mà ta cung cấp.

Trong chương này, chúng ta sẽ thảo luận về lời nguyền chiều và hiểu được những gì diễn ra trong không gian nhiều chiều. Sau đó, chúng ta sẽ xem xét hai hướng tiếp cận chính để giảm chiều (phép chiếu và Học Đa tạp – *Manifold Learning*), và ba trong số các kỹ thuật giảm chiều phổ biến nhất: PCA, PCA hạt nhân và LLE.

Lời nguyền Chiều

Chúng ta đã quá quen với việc sống trong không gian ba chiều,[1] vì vậy việc hình dung một không gian nhiều chiều sẽ tương đối khó khăn. Ngay cả một siêu khối 4D cơ bản[2] cũng đã rất khó để hình dung (tham khảo Hình 8.1), chứ chưa nói đến một ellipsoid 200 chiều uốn cong trong không gian 1,000 chiều.



Hình 8.1. Điểm, đoạn thẳng, hình vuông, khối lập phương và tesseract (là các siêu khối từ 0-chiều tới 4-chiều)

Có nhiều vấn đề thể hiện các đặc tính rất khác biệt trong không gian nhiều chiều. Ví dụ, nếu ta chọn ngẫu nhiên một điểm trong một hình vuông đơn vị (có kích thước 1×1), xác suất điểm này cách đường biên ít hơn 0.001 đơn vị chỉ là 0.4% (nói cách khác, khả năng một điểm ngẫu nhiên sẽ đạt "cực trị" theo một chiều bất kỳ là rất thấp). Tuy nhiên trong một siêu khối đơn vị 10,000 chiều, xác suất này lên đến hơn 99.999999%. Hầu hết các điểm trong một siêu khối nhiều chiều đều nằm rất gần với đường biên.[3]

Một điểm khác biệt còn phiền phức hơn: khoảng cách trung bình giữa hai điểm bất kỳ trong một hình vuông đơn vị là xấp xỉ 0.52. Nếu ta chọn hai điểm ngẫu nhiên trong một hình lập phương đơn vị, khoảng cách trung bình sẽ gần bằng 0.66. Nhưng với hai điểm được chọn ngẫu nhiên trong một siêu khối 1,000,000 chiều thì sao? Khoảng cách trung bình, mặc dù khó tin, sẽ xấp xỉ 408.25 (gần bằng $\sqrt{1,000,000/6}$)! Điều này thật khó hiểu: làm thế nào mà hai điểm lại có thể cách xa nhau như vậy khi mà cả hai đều nằm trong cùng một siêu khối đơn vị? Thực chất, có rất nhiều khoảng trống tồn tại trong một không gian nhiều chiều. Kết quả là tập dữ liệu nhiều chiều có thể rất thưa: các mẫu huấn luyện rất có thể sẽ nằm cách xa nhau. Điều này cũng có nghĩa là các mẫu mới sẽ cách xa các mẫu huấn luyện, khiến việc dự đoán trở nên kém tin cậy hơn đáng kể so với tập dữ liệu ít chiều, vì việc dự đoán phải dựa vào các phép ngoại suy lớn hơn. Nói ngắn gọn, số chiều của tập huấn luyện càng lớn thì nguy cơ quá khớp càng cao.

Trên lý thuyết, một giải pháp cho lời nguyền chiều là tăng kích thước của tập huấn luyện sao cho mật độ các mẫu huấn luyện đủ dày. Không may là để đạt được mật độ này trong thực tế, số lượng mẫu huấn luyện cần tăng theo hàm mũ của số chiều. Với chỉ 100 đặc trưng (ít hơn rất nhiều so với MNIST), ta sẽ cần nhiều mẫu huấn luyện hơn tổng số nguyên tử trong toàn bộ vũ trụ khả kiến để các mẫu huấn luyện có thể cách nhau trung bình 0.1 đơn vị, giả sử rằng chúng trải đều theo tất cả các chiều.

[1] Bốn chiều nếu bạn tính cả chiều thời gian, và nhiều chiều hơn nữa nếu bạn tin theo lý thuyết dây (*string theory*)

[2] Xem một khối xoay tesseract được chiếu lên không gian 3-chiều tại *https://homl.info/30*. Ảnh trên Wikipedia được tạo bởi NerdBoy1392 (Creative Commons BY-SA 3.0). Sao chép lại từ *https://en.wikipedia.org/wiki/Tesseract*.

[3] Một điều thú vị: bất kỳ ai mà bạn biết đều có thể "đạt cực trị" theo ít nhất một chiều (ví dụ như lượng đường họ bỏ vào cà phê), nếu bạn xét đủ số chiều.

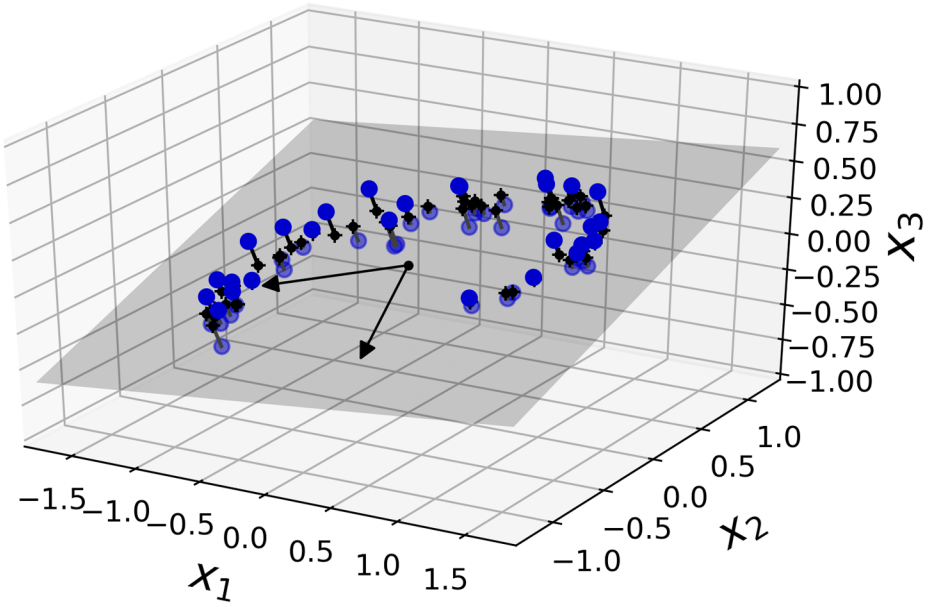
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các Phương pháp Giảm Chiều chính

Trước khi đi sâu vào các thuật toán giảm chiều cụ thể, hãy xem xét hai phương pháp giảm chiều chính: phép chiếu và Học Đa tạp.

Phép chiếu

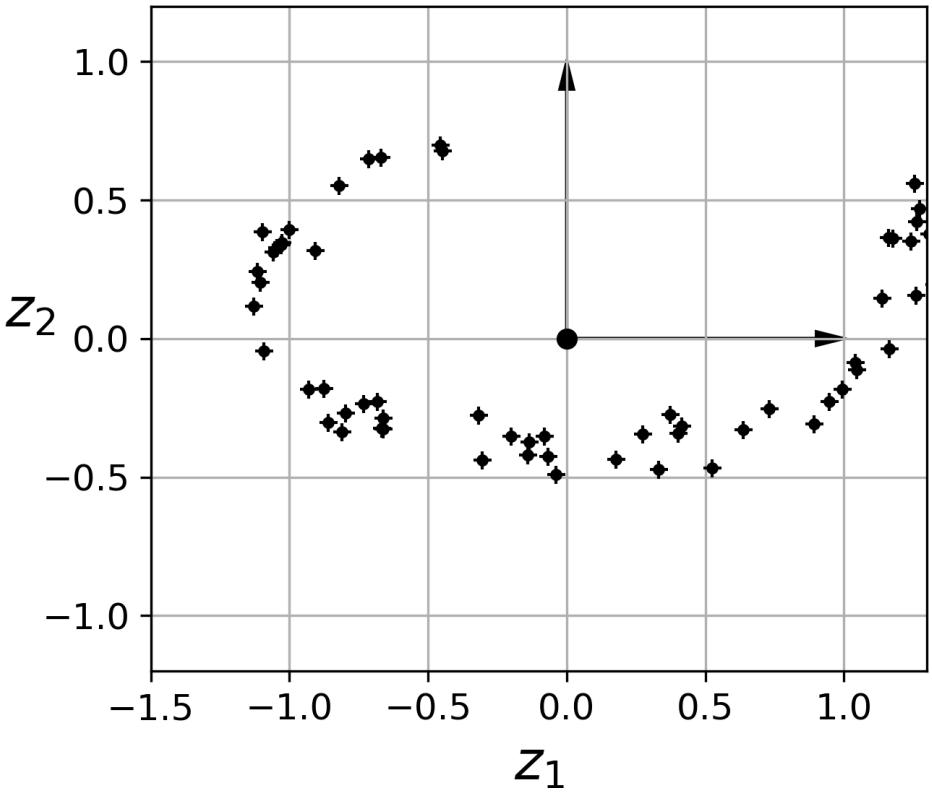
Trong hầu hết các bài toán thực tế, các mẫu huấn luyện *không* được trải đều theo tất cả các chiều. Nhiều đặc trưng có giá trị gần như không đổi, trong khi những đặc trưng khác có tính tương quan cao (như đã thảo luận trước đó với tập dữ liệu MNIST). Kết quả là tất cả các mẫu huấn luyện đều nằm trong (hoặc gần) một *không gian con* ít chiều hơn hẳn so với không gian nhiều chiều gốc. Điều này nghe có vẻ rất trừu tượng, vì thế hãy xem xét một ví dụ. Trong Hình 8.2 ta có thể thấy một tập dữ liệu ba chiều được biểu diễn bởi các hình tròn.



Hình 8.2. Tập dữ liệu ba chiều nằm gần không gian con hai chiều

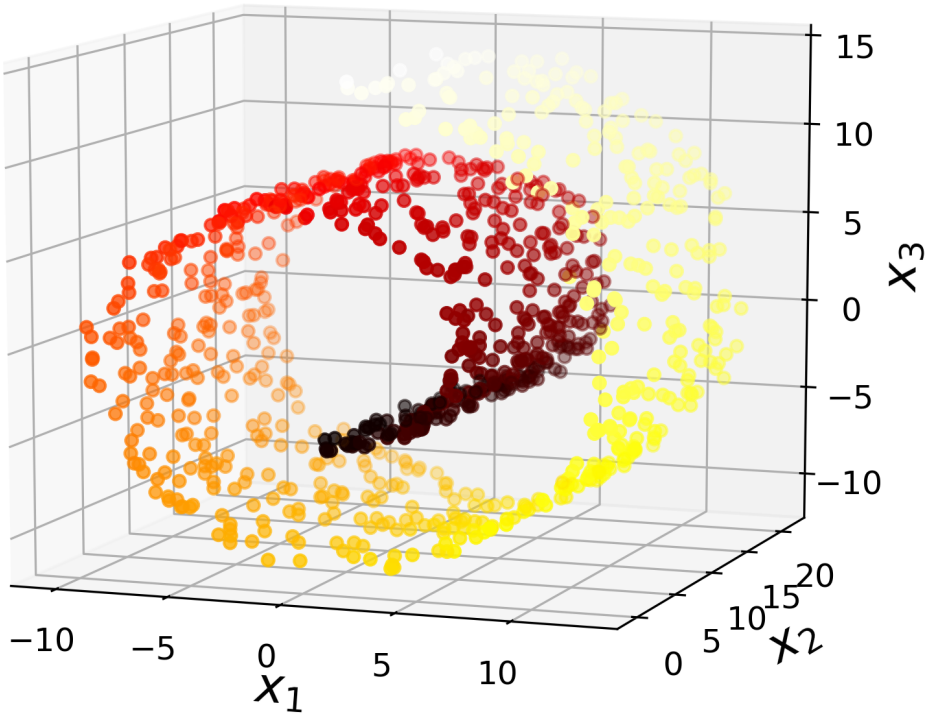
Chú ý rằng tất cả các mẫu huấn luyện đều nằm gần một mặt phẳng: đây là không gian con ít chiều (hai chiều) của không gian nhiều chiều (ba chiều). Nếu chiếu mọi mẫu huấn luyện vuông góc với không gian con này (được biểu diễn bởi các đường ngắn nối các mẫu với mặt phẳng), ta có được tập huấn luyện hai chiều mới được biểu diễn trong Hình 8.3. Vậy là chúng ta vừa giảm chiều của tập huấn luyện từ ba xuống hai chiều. Chú ý rằng các trục tương ứng với các đặc trưng mới là z_1 và z_2 (tọa độ của các hình chiếu trên mặt phẳng).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 8.3. Dữ liệu hai chiều mới sau phép chiếu

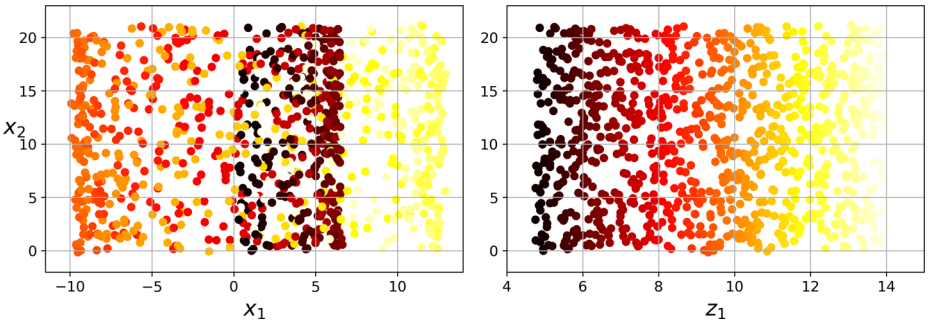
Tuy nhiên, không phải lúc nào phép chiếu cũng là hướng tiếp cận tốt nhất để giảm chiều. Trong nhiều trường hợp không gian con có thể bị xoắn và xoay, chẳng hạn trong như tập dữ liệu đồ chơi *Swiss roll* nổi tiếng được biểu diễn trong Hình 8.4.



Hình 8.4. Tập dữ liệu Swiss roll

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chỉ đơn thuần chiếu lên một mặt phẳng (ví dụ như loại bỏ x_3) sẽ ép các lớp của Swiss roll lại với nhau như đồ thị bên trái trong Hình 8.5. Điều ta muốn là trải phẳng cuộn này ra để có được tập dữ liệu hai chiều ở đồ thị bên phải trong Hình 8.5.



Hình 8.5. Ép các lớp bằng cách chiếu lên mặt phẳng (bên trái) so với trải phẳng tập Swiss roll (bên phải)

Học Đa tạp

Tập Swiss roll là một ví dụ của *đa tạp* (*manifold*) hai chiều. Nói một cách đơn giản, đa tạp hai chiều là một hình hai chiều có thể được uốn và xoắn trong không gian nhiều chiều. Tổng quát hơn, một đa tạp d-chiều là một phần của không gian n-chiều (với $d < n$), có tính tương đương cục bộ với một siêu phẳng d-chiều. Với tập Swiss roll thì $d = 2$ và $n = 3$: tập dữ liệu này tương đương cục bộ với một mặt phẳng hai chiều, nhưng được cuộn lại trong không gian ba chiều.

Nhiều thuật toán giảm chiều hoạt động bằng cách mô hình hóa sự đa tạp của các mẫu huấn luyện, và phương pháp này được gọi là *Học Đa tạp*. Nó dựa vào *giả định đa tạp*, hay còn được gọi là *giả thuyết đa tạp*, cho rằng hầu hết các tập dữ liệu nhiều chiều trong thực tế nằm gần với một đa tạp có số chiều thấp hơn nhiều. Giả định này thường xuyên đúng trong thực tiễn.

Hãy xem xét lại tập dữ liệu MNIST: tất cả các hình ảnh chứa chữ số viết tay đều có một số điểm tương đồng. Chúng được tạo bởi các đường nét liền, phần viền ảnh có màu trắng và hầu hết các chữ số đều được căn giữa. Nếu ta sinh các hình ảnh một cách ngẫu nhiên, chỉ một phần rất rất nhỏ trong số chúng sẽ trông giống chữ số viết tay. Nói cách khác, số bậc tự do khi tạo một ảnh chứa chữ số sẽ thấp hơn đáng kể so với số bậc tự do khi tạo một hình ảnh bất kỳ. Những ràng buộc này có xu hướng ép tập dữ liệu thành một đa tạp ít chiều hơn.

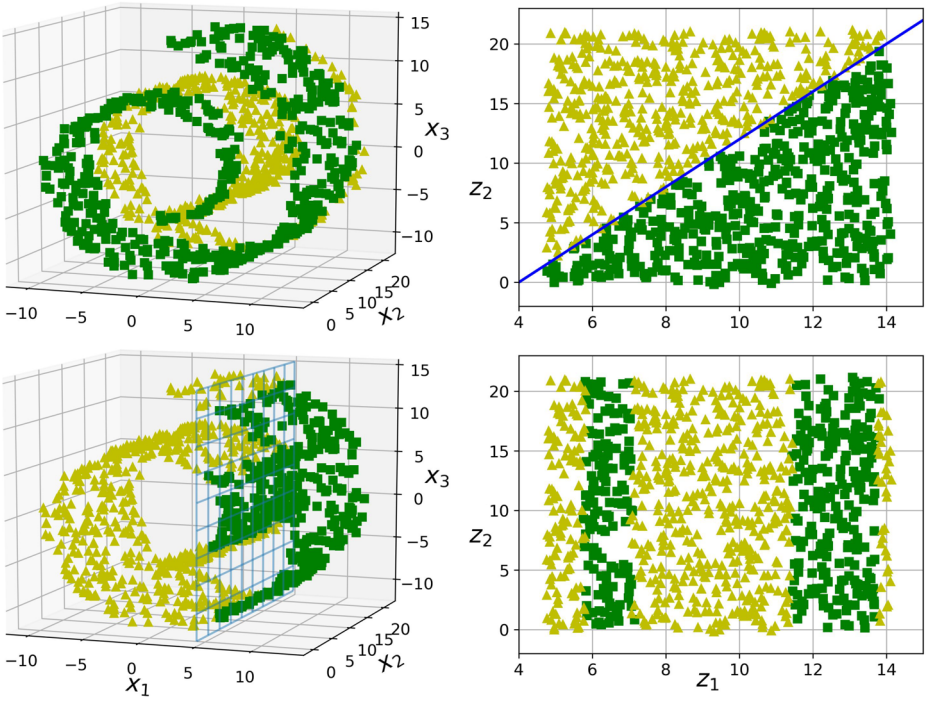
Giả định đa tạp thường đi kèm với một giả định ngầm khác: tác vụ hiện tại (như phân loại hay hồi quy) sẽ đơn giản hơn nếu được biểu diễn trong không gian ít chiều hơn của đa tạp. Ví dụ, ở hàng trên cùng trong Hình 8.6, tập Swiss roll được phân chia thành hai lớp: trong không gian ba chiều (biểu đồ bên trái) thì ranh giới quyết định tương đối phức tạp, tuy nhiên trong không gian đa tạp hai chiều được trải phẳng (biểu đồ bên phải) thì ranh giới quyết định chỉ là một đường thẳng.

Dù vậy, giả định ngầm này không phải lúc nào cũng đúng. Ví dụ, ở hàng dưới cùng trong Hình 8.6, ranh giới quyết định nằm ở $x_1 = 5$. Ranh giới quyết định này trông rất đơn giản trong không gian ba chiều ban đầu (một mặt phẳng thẳng đứng), nhưng lại trở nên phức tạp hơn trong đa tạp được trải phẳng (một tập hợp 4 đường thẳng độc lập).

Tóm lại, việc giảm số chiều của tập huấn luyện trước khi huấn luyện mô hình thường sẽ tăng tốc quá trình huấn luyện, nhưng không phải lúc nào ta cũng sẽ có được nghiệm tốt hơn hoặc đơn giản hơn, mà điều này phụ thuộc vào tập dữ liệu.

Hy vọng bạn đã hiểu rõ về lời nguyền chiều và cách mà các thuật toán giảm chiều có thể chống lại nó, đặc biệt là khi giả định đa tạp là đúng. Phần còn lại của chương này sẽ trình bày một vài thuật toán giảm chiều phổ biến nhất.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 8.6. Ranh giới quyết định không phải lúc nào cũng đơn giản hơn ở không gian ít chiều

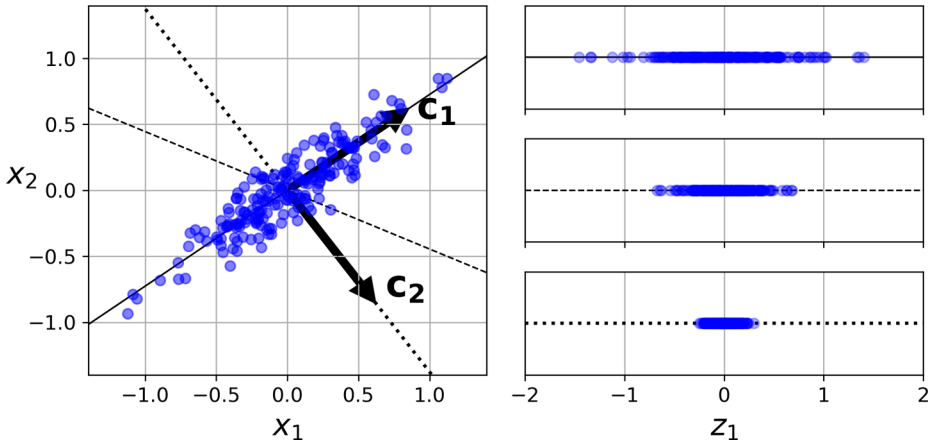
PCA

Phân tích Thành phần Chính (*Principal Component Analysis* – PCA) là thuật toán giảm chiều phổ biến nhất cho tới nay. Đầu tiên nó xác định siêu phẳng nằm gần dữ liệu nhất, sau đó chiếu các điểm dữ liệu lên siêu phẳng đó, như được biểu diễn trong Hình 8.2.

Bảo toàn Phương sai

Trước khi chiếu tập huấn luyện lên một siêu phẳng ít chiều hơn, ta cần phải chọn siêu phẳng phù hợp. Để lấy ví dụ, hãy xem xét tập dữ liệu hai chiều đơn giản được biểu diễn ở đồ thị bên trái trong Hình 8.7, cùng với ba trục khác nhau (các siêu phẳng một chiều). Đồ thị bên phải biểu diễn kết quả của phép chiếu tập dữ liệu lên mỗi trục này. Có thể thấy rằng phép chiếu lên đường nét liền bảo toàn được phương sai tối đa, trong khi phép chiếu lên đường chấm chỉ bảo toàn được một lượng phương sai rất nhỏ và phép chiếu lên đường nét đứt chỉ bảo toàn toàn được một lượng phương sai tương đối.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 8.7. Lựa chọn không gian con để thực hiện phép chiếu

Lựa chọn hợp lý ở đây là chiếu lên trục mà ở đó phương sai được bảo toàn tối đa, bởi lượng thông tin mất đi sẽ ít hơn so với các phép chiếu lên các trục khác. Một cách khác để giải thích lựa chọn này đó là trục nói trên cực tiểu hóa trung bình bình phương khoảng cách giữa tập dữ liệu gốc và phép chiếu của tập dữ liệu trên trục đó. Đây là ý tưởng đơn giản phía sau PCA.[4]

Thành phần Chính

PCA xác định trục có lượng phương sai lớn nhất trong tập huấn luyện. Trong Hình 8.7 thì trục đó là đường nét liền. Nó cũng xác định trục thứ hai trực giao với trục thứ nhất và chiếm đa phần lượng phương sai còn lại. Trong ví dụ trên thì trục thứ hai chỉ có thể là đường chấm. Nếu đây là tập dữ liệu có nhiều chiều hơn, PCA cũng sẽ xác định thêm trục thứ ba, trực giao với cả hai trục trước đó và trục thứ tư, thứ năm, cứ như vậy cho đến khi số trục bằng với số chiều của tập dữ liệu.

Trục *thứ i* được gọi là *thành phần chính* (*principle component*) *thứ i* của dữ liệu. Trong Hình 8.7, thành phần chính đầu tiên là trục chứa vector $\mathbf{c_1}$, và thành phần chính thứ hai là trục chứa vector $\mathbf{c_2}$. Trong Hình 8.2, hai thành phần chính đầu tiên là các trục trực giao chứa hai mũi tên nằm trên mặt phẳng, còn thành phần chính thứ ba là trục trực giao với mặt phẳng đó.

Ghi chú

Với mỗi thành phần chính, PCA tìm vector đơn vị có trung bình bằng không và cùng hướng với thành phần chính. Vì một trục có hai vector đơn vị ngược chiều, hướng của các vector đơn vị thu được từ PCA sẽ không ổn định: chỉ xáo trộn tập huấn luyện một chút có thể khiến PCA trả về các vector đơn vị ngược chiều với các vector ban đầu. Tuy nhiên, thường thì trục của các vector này vẫn không thay đổi. Trong một số trường hợp, một cặp vector đơn vị có thể xoay hoặc hoán đổi (nếu phương sai dọc theo hai trục này gần bằng nhau), nhưng nhìn chung mặt phẳng mà chúng xác lập sẽ không thay đổi.

Vậy làm thế nào để tìm các thành phần chính của tập huấn luyện? Rất may là có một kỹ thuật phân tích ma trận tiêu chuẩn tên là *Phân tích Giá trị Suy biến* (*Singular Value Decomposition* – SVD) có thể phân tích ma trận tập huấn luyện $\mathbf{X}$ thành tích của ba ma trận $\mathbf{U\Sigma V}^T$, trong đó $\mathbf{V}$ chứa các vector đơn vị định nghĩa tất cả các thành phần chính cần tìm, như được trình bày trong Phương trình 8.1.

[4] Karl Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, no. 11 (1901): 559-572, *https://homl.info/pca*.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$\mathbf{V} = \begin{pmatrix} | & | & & | \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_n} \\ | & | & & | \end{pmatrix}$$

Phương trình 8.1. Ma trận thành phần chính

Đoạn mã Python sau đây sử dụng hàm `svd()` của NumPy để thu được tất cả thành phần chính của tập huấn luyện, sau đó trích xuất hai vector đơn vị định nghĩa hai thành phần chính đầu tiên:

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

Lưu ý

PCA giả định rằng tập dữ liệu được căn giữa ở gốc toạ độ. Các lớp PCA của Scikit-Learn đã ngầm đảm nhận việc căn giữa này. Nếu bạn tự lập trình PCA (ở ví dụ trên) hay sử dụng thư viện khác, đừng quên căn giữa dữ liệu trước.

Chiếu Xuống d Chiều

Khi đã xác định tất cả các thành phần chính, ta có thể giảm chiều của tập dữ liệu xuống d chiều bằng cách chiếu lên siêu mặt phẳng được định nghĩa bởi d thành phần chính đầu tiên. Việc lựa chọn siêu mặt phẳng này đảm bảo phép chiếu sẽ giữ được phương sai nhiều nhất có thể. Ví dụ, trong Hình 8.2 tập dữ liệu 3D được chiếu xuống mặt phẳng 2D được định nghĩa bởi hai thành phần chính đầu tiên, bảo toàn được một phần lớn phương sai của tập dữ liệu. Kết quả là, hình chiếu 2D nhìn rất giống với tập dữ liệu 3D ban đầu.

Để chiếu tập huấn luyện lên siêu mặt phẳng và thu được tập dữ liệu $\mathbf{X}_{\text{chiếu-}d}$ có d chiều, ta tính tích giữa ma trận tập huấn luyện $\mathbf{X}$ với ma trận $\mathbf{W}_d$, được định nghĩa là ma trận chứa d cột đầu tiên của $\mathbf{V}$, như trình bày trong Phương trình 8.2.

$$\mathbf{X}_{\text{chiếu-}d} = \mathbf{X}\mathbf{W}_d$$

Phương trình 8.2. Chiếu tập huấn luyện xuống d chiều

Đoạn mã Python dưới đây chiếu tập huấn luyện lên mặt phẳng được định nghĩa bởi hai thành phần chính đầu tiên:

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

Vậy là xong. Bây giờ ta đã biết cách giảm chiều của bất cứ tập dữ liệu nào xuống bất cứ số chiều nào mà vẫn bảo toàn phương sai nhiều nhất có thể.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Sử dụng Scikit-Learn

Lớp `PCA` của Scikit-Learn sử dụng phép phân rã SVD để thực hiện PCA như được trình bày ở đầu chương. Đoạn mã sau áp dụng PCA để giảm chiều dữ liệu xuống 2 chiều (chú ý rằng việc căn giữa dữ liệu đã được thực hiện một cách tự động):

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

Sau khi khớp `PCA` vào tập dữ liệu, thuộc tính `components_` chứa chuyển vị của $\mathbf{W}_d$ (ví dụ, vector đơn vị định nghĩa thành phần chính đầu tiên sẽ là `pca.components_.T[:, 0]`).

Tỉ lệ Phương sai được Giải thích

Một thông tin hữu ích khác là *tỉ lệ phương sai được giải thích* (*explained variance ratio*) của mỗi thành phần chính, có trong biến `explained_variance_ratio_`. Nó cho biết tỉ lệ phần trăm phương sai của tập dữ liệu nằm dọc theo mỗi thành phần chính. Ví dụ, hãy xem tỉ lệ phương sai được giải thích của hai thành phần đầu tiên của tập dữ liệu 3D được trình bày ở Hình 8.2:

```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

Kết quả đầu ra cho biết 84.2% phương sai của tập dữ liệu nằm dọc theo thành phần chính thứ nhất và 14.6% nằm dọc theo thành phần chính thứ hai. Còn lại dưới 1.2% nằm trong thành phần chính thứ ba nên ta có thể kết luận thành phần chính thứ ba chứa rất ít thông tin.

Chọn Số chiều Hợp lý

Thay vì chọn số chiều một cách ngẫu nhiên, ta có thể chọn số chiều sao cho lượng phương sai bảo toàn là đủ lớn (95%). Dĩ nhiên trừ khi ta muốn giảm chiều để trực quan hóa dữ liệu, trong trường hợp này số chiều thường được giảm xuống 2 hoặc 3.

Đoạn mã bên dưới thực hiện PCA mà không giảm chiều, sau đó tính số chiều nhỏ nhất sao cho giữ được ít nhất 95% phương sai của tập huấn luyện:

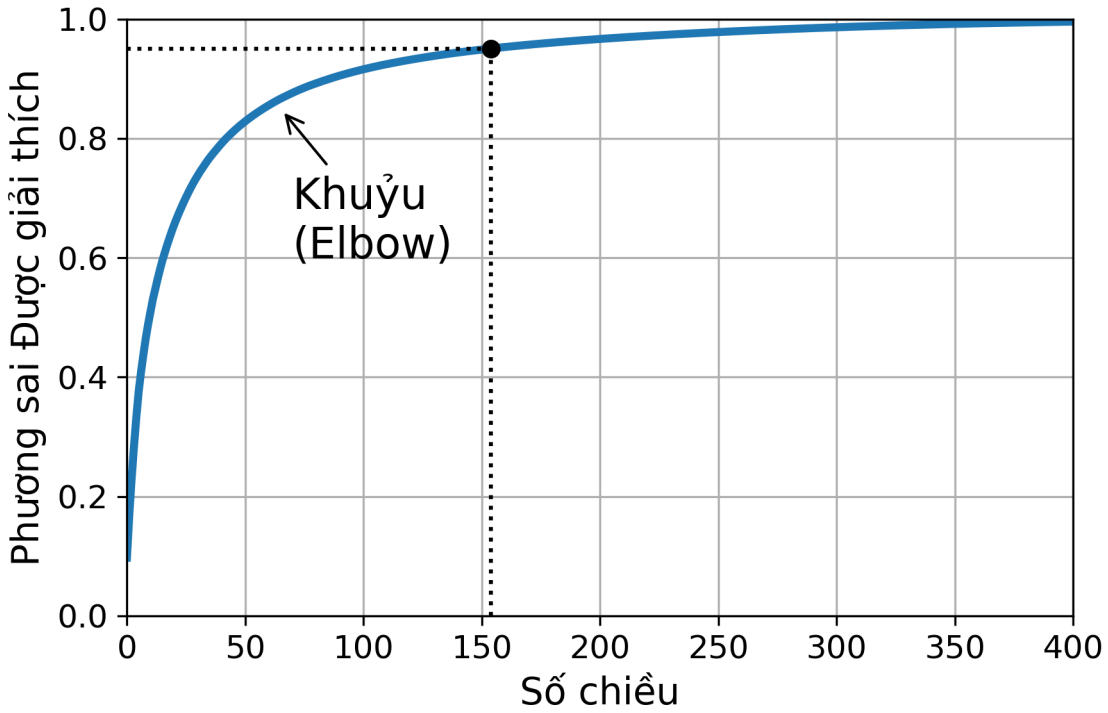
```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

Ta có thể thết lập giá trị `n_components=d` và chạy lại PCA. Nhưng có một cách tốt hơn: thay vì ghi rõ số lượng thành phần chính muốn giữ lại, ta có thể thiết lập `n_components` là một giá trị số thực (float) giữa 0.0 và 1.0 để chỉ định tỉ lệ phương sai mà ta muốn bảo toàn:

```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X_train)
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Một phương án khác là vẽ biểu đồ phương sai được giải thích như một hàm theo số chiều (đơn giản là vẽ kết quả hàm `cumsum`, tham khảo Hình 8.8). Thường sẽ có một khuỷu (*elbow*) trong đường cong, nơi mà phương sai được giải thích ngừng tăng nhanh. Trong trường hợp này, việc giảm số chiều xuống khoảng 100 sẽ không làm mất quá nhiều phương sai được giải thích.



Hình 8.8. Phương sai được giải thích như một hàm của số chiều

PCA cho Tác vụ Nén

Sau khi giảm chiều, tập huấn luyện sẽ chiếm ít dung lượng hơn. Để ví dụ, hãy thử áp dụng PCA vào tập dữ liệu MNIST sao cho giữ được 95% phương sai. Ta sẽ thấy rằng mỗi mẫu chỉ chứa khoảng 150 đặc trưng thay vì 784 đặc trưng như ban đầu. Do vậy, không những ta giữ được phần lớn phương sai, tập dữ liệu bây giờ chiếm còn chưa đến 20% kích thước ban đầu! Đây là một tỉ lệ nén hợp lý và ta có thể thấy việc giảm kích thước này làm tăng tốc độ của thuật toán phân loại (như bộ phân loại SVM) một cách đáng kể.

Ta có thể giải nén bộ giữ liệu về 784 chiều bằng phép biến đổi nghịch đảo của phép chiếu PCA. Tuy nhiên, ta sẽ không có được dữ liệu gốc vì phép chiếu đã làm mất một vài thông tin (khoảng 5% phương sai bị mất), nhưng kết quả sẽ xấp xỉ với dữ liệu gốc. Trung bình bình phương khoảng cách giữa dữ liệu gốc và dữ liệu được khôi phục lại (nén sau đó giải nén) được gọi là *lỗi khôi phục* (*reconstruction error*).

Đoạn mã sau nén tập dữ liệu MNIST xuống 154 chiều, sau đó dùng `inverse_transform()` để giải nén trở lại 784 chiều:

```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hình 8.9 biểu diễn các chữ số từ tập huấn liệu ban đầu (bên trái) và các chữ số tương ứng sau khi nén và giải nén. Có thể thấy chất lượng ảnh giảm một chút nhưng các chữ số vẫn được giữ nguyên vẹn.

Hình 8.9. Phép nén tập dữ liệu MNIST bảo toàn 95% phương sai

Công thức của phép biến đổi nghịch đảo được trình bày trong Phương trình 8.3.

$$\mathbf{X}_{\text{khôi-phục}} = \mathbf{X}_{\text{chiếu-}d}\mathbf{W}_d{}^T$$

Phương trình 8.3. Phép biến đổi nghịch đảo PCA, khôi phục số chiều ban đầu

PCA Ngẫu nhiên

Nếu ta đặt siêu tham số `svd_solver` là `"randomized"`, Scikit-Learn sẽ sử dụng một thuật toán ngẫu nhiên có tên là *PCA Ngẫu nhiên* (*Randomized PCA*) để nhanh chóng tìm ra giá trị xấp xỉ của d thành phần chính đầu tiên. Độ phức tạp của thuật toán này là $O(m \times d^2) + O(d^3)$ so với $O(m \times n^2) + O(n^3)$ của SVD thông thường, nhanh hơn đáng kể khi d nhỏ hơn rất nhiều so với n:

```
rnd_pca = PCA(n_components=154, svd_solver="randomized")
X_reduced = rnd_pca.fit_transform(X_train)
```

Giá trị mặc định của `svd_solver` là `"auto"`: Scikit-Learn sẽ tự động dùng thuật toán PCA ngẫu nhiên nếu m hoặc n lớn hơn 500 và d nhỏ hơn 80% của m hoặc n, ngược lại SVD thông thường sẽ được sử dụng. Nếu muốn buộc Scikit-Learn sử dụng SVD đầy đủ, ta có thể đặt siêu tham số `svd_solver` thành `"full"`.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

PCA Gia tăng

Một vấn đề với các cách thực thi PCA phía trên là thuật toán cần bộ nhớ đủ lớn để chứa toàn bộ tập huấn luyện. Rất may là các thuật toán *PCA Gia tăng* (*Incremental PCA* – IPCA) đã được phát triển, cho phép ta chia tập huấn luyện thành các minibatch và áp dụng PCA Gia tăng lần lượt với các minibatch đó. Phương pháp này khá hữu dụng đối với các tập huấn luyện lớn hay khi ta muốn áp dụng PCA trực tuyến (hoạt động liên tục với các mẫu dữ liệu mới).

Đoạn mã sau chia tập dữ liệu MNIST thành 100 minibatch (sử dụng hàm `array_split()` của NumPy) và đưa chúng vào lớp `IncrementalPCA` của Scikit-Learn[5] để giảm chiều của tập MNIST xuống còn 154 chiều (giống như trước). Lưu ý rằng ta phải gọi phương thức `partial_fit()` với mỗi minibatch thay vì phương thức `fit()` với toàn bộ tập huấn luyện:

```
from sklearn.decomposition import IncrementalPCA

n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
for X_batch in np.array_split(X_train, n_batches):
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

Một cách khác là ta có thể dùng lớp `memmap` của NumPy, cho phép thao tác trên một mảng lớn được lưu trong tệp nhị phân tại ổ đĩa như thể nó đang nằm trong bộ nhớ. Do lớp này chỉ thực sự nạp dữ liệu vào bộ nhớ khi cần thiết, `IncrementalPCA` sẽ luôn sử dụng một phần nhỏ của mảng, dung lượng bộ nhớ nhờ đó sẽ được kiểm soát. Điều này cũng cho phép ta gọi phương thức `fit()` thông thường, như trong đoạn mã sau:

```
X_mm = np.memmap(filename, dtype="float32", mode="readonly", shape=(m, n))

batch_size = m // n_batches
inc_pca = IncrementalPCA(n_components=154, batch_size=batch_size)
inc_pca.fit(X_mm)
```

PCA Hạt nhân

Trong Chương 5, ta đã bàn về thủ thuật hạt nhân, một kĩ thuật toán học ngầm ánh xạ các mẫu dữ liệu vào không gian rất nhiều chiều (gọi là *không gian đặc trưng – feature space*), cho phép ta thực hiện các tác vụ phân loại và hồi quy phi tuyến với Máy Vector Hỗ trợ. Nhắc lại rằng một ranh giới quyết định tuyến tính trong không gian đặc trưng nhiều chiều sẽ tương ứng với một ranh giới quyết định phi tuyến phức tạp trong *không gian ban đầu* (*original space*).

Hóa ra thủ thuật này cũng có thể được áp dụng cho PCA, cho phép ta thực hiện các phép chiếu phi tuyến phức tạp để giảm chiều dữ liệu. Phương pháp này được gọi là *PCA Hạt nhân* (*Kernel PCA* – kPCA).[6]

Đoạn mã sau sử dụng lớp `KernelPCA` của Scikit-Learn để sử dụng kPCA với hạt nhân RBF (tham khảo Chương 5 để biết thêm chi tiết về hạt nhân RBF và các loại hạt nhân khác):

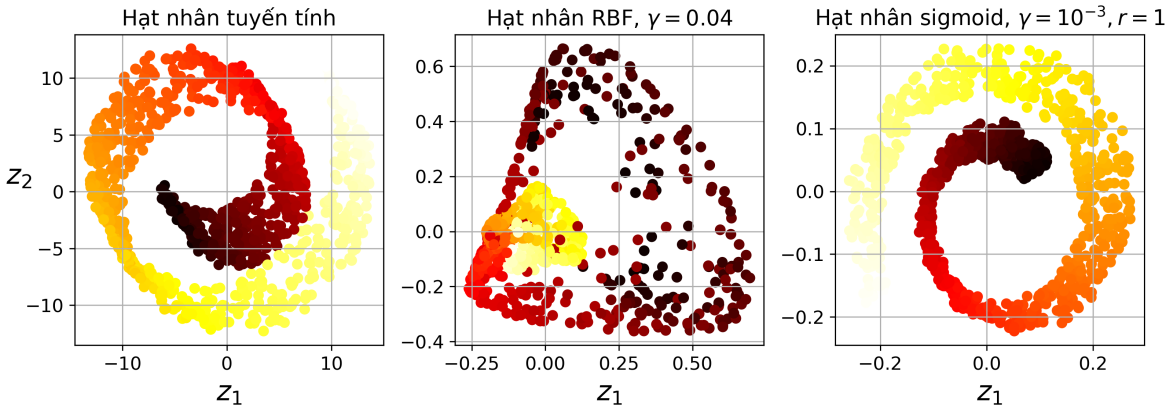
[5] Scikit-Learn sử dụng thuật toán được mô tả trong bài báo của David A. Ross và cộng sự, "Incremental Learning for Robust Visual Tracking," *International Journal of Computer Vision* 77, no. 1–3 (2008): 125–141.

[6] Bernhard Schölkopf và cộng sự, "Kernel Principal Component Analysis," trong *Lecture Notes in Computer Science* 1327 (Berlin: Springer, 1997): 583–588.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
from sklearn.decomposition import KernelPCA

rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```

Hình 8.10 biểu diễn tập dữ liệu Swiss roll được giảm xuống còn hai chiều với hạt nhân tuyến tính (tương đương với việc sử dụng lớp `PCA` đơn thuần), hạt nhân RBF và hạt nhân sigmoid.



Hình 8.10. Tập Swiss roll được giảm xuống hai chiều bằng kPCA với các hạt nhân khác nhau

Lựa chọn Hạt nhân và Tinh chỉnh Siêu tham số

Bởi kPCA là thuật toán học không giám sát, do đó không có phép đo chất lượng mặc định nào có thể giúp ta chọn hạt nhân và các giá trị siêu tham số tốt nhất. Dù vậy, giảm chiều dữ liệu thường là bước chuẩn bị cho một tác vụ học có giám sát (ví dụ như phân loại), nên ta có thể sử dụng tìm kiếm dạng lưới để chọn hạt nhân và siêu tham số đạt được chất lượng cao nhất trên tác vụ đó. Đoạn mã sau tạo một pipeline gồm hai bước: đầu tiên giảm chiều dữ liệu bằng kPCA, rồi áp dụng Hồi quy Logistic để phân loại. Sau đó `GridSearchCV` được dùng để tìm hạt nhân và giá trị `gamma` tốt nhất cho kPCA nhằm đạt được độ chính xác cao nhất khi phân loại ở cuối pipeline:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([
        ("kpca", KernelPCA(n_components=2)),
        ("log_reg", LogisticRegression())
    ])

param_grid = [{
        "kpca__gamma": np.linspace(0.03, 0.05, 10),
        "kpca__kernel": ["rbf", "sigmoid"]
    }]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

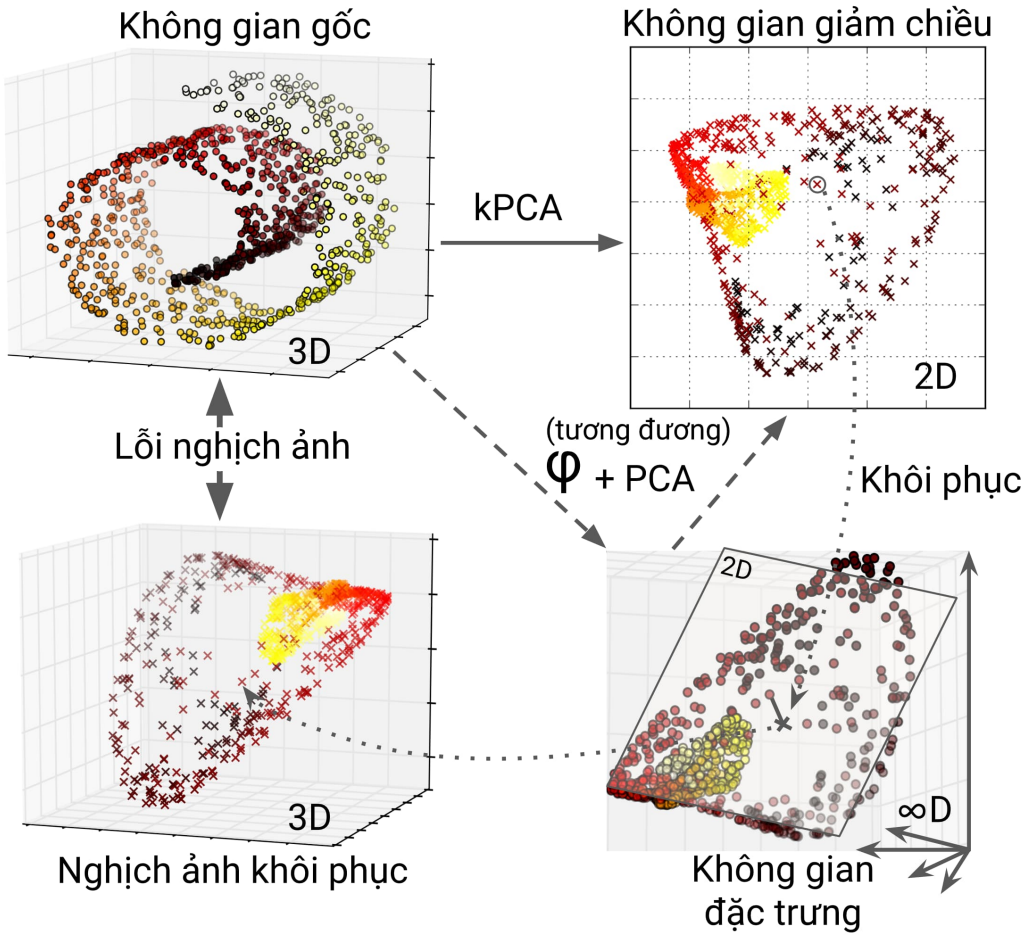
Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hạt nhân và các siêu tham số tốt nhất được lưu ở biến `best_params_`:

```
>>> print(grid_search.best_params_)
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```

Một phương pháp khác, hoàn toàn không giám sát, đó là lựa chọn hạt nhân và siêu tham số sao cho lỗi khôi phục nhỏ nhất. Lưu ý rằng việc khôi phục sẽ không dễ dàng như khi sử dụng PCA tuyến tính, điều này được giải thích như sau. Hình 8.11 biểu diễn tập dữ liệu ba chiều Swiss roll (biểu đồ trên-trái) và tập dữ liệu hai chiều thu được sau khi áp dụng kPCA với hạt nhân RBF (biểu đồ trên-phải). Nhờ có thủ thuật hạt nhân, phép biến đổi này tương đương toán học với việc sử dụng *ánh xạ đặc trưng* (*feature map*) φ để ánh xạ tập huấn luyện vào không gian đặc trưng vô số chiều (biểu đồ dưới-phải), rồi chiếu tập huấn luyện đã biến đổi xuống còn hai chiều bằng PCA tuyến tính.

Chú ý rằng khi đảo ngược bước áp dụng PCA tuyến tính cho một mẫu bất kỳ trong không gian đã giảm chiều, điểm được khôi phục sẽ nằm trong không gian đặc trưng thay vì không gian ban đầu (ví dụ như điểm được biểu diễn bằng dấu X trong biểu đồ). Bởi không gian đặc trưng có vô số chiều, ta không thể tính toán điểm này và do đó không thể tính lỗi khôi phục thực sự. May mắn thay, ta vẫn có thể tìm một điểm trong không gian ban đầu có ánh xạ gần với điểm được khôi phục. Điểm này được gọi là *nghịch ảnh khôi phục* (*reconstruction pre-image*). Một khi có được nghịch ảnh này, ta có thể tính bình phương khoảng cách tới mẫu dữ liệu ban đầu. Sau đó ta có thể lựa chọn hạt nhân và siêu tham số sao cho lỗi nghịch ảnh khôi phục nhỏ nhất.



Hình 8.11. PCA Hạt nhân và lỗi nghịch ảnh khôi phục

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bạn có thể đang thắc mắc cách thực hiện phương pháp khôi phục này. Một giải pháp đó là huấn luyện một mô hình hồi quy có giám sát với tập huấn luyện chứa các mẫu đã giảm chiều và nhãn là các mẫu ban đầu. Scikit-Learn sẽ tự động thực hiện việc này nếu ta đặt `fit_inverse_transform=True`, như trong đoạn mã sau:[7]

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433,
                    fit_inverse_transform=True)
X_reduced = rbf_pca.fit_transform(X)
X_preimage = rbf_pca.inverse_transform(X_reduced)
```

Ghi chú

Theo mặc định, `fit_inverse_transform=False`, do đó lớp `KernelPCA` sẽ không có phương thức `inverse_transform()`. Phương thức này sẽ chỉ được tạo khi đặt `fit_inverse_transform=True`.

Sau đó ta có thể tính lỗi nghịch ảnh khôi phục:

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(X, X_preimage)
32.786308795766132
```

Bây giờ, ta có thể sử dụng tìm kiếm dạng lưới với phép kiểm định chéo để tìm hạt nhân và các giá trị siêu tham số nhằm cực tiểu hóa lỗi khôi phục.

LLE

Embedding Tuyến tính Cục bộ (*Locally Linear Embedding* – LLE)[8] là một kỹ thuật *giảm chiều phi tuyến* (*nonlinear dimensionality reduction* – NLDR) hoạt động rất tốt. Đây là một kỹ thuật Học Đa tạp không phụ thuộc vào phép chiếu giống như các thuật toán trước đây. Nói một cách ngắn gọn, LLE hoạt động bằng cách đo mối quan hệ tuyến tính giữa mỗi mẫu huấn luyện với các mẫu lân cận gần nhất của nó, sau đó tìm kiếm một biểu diễn ít chiều của tập huấn luyện sao cho các mối quan hệ cục bộ này được bảo toàn một cách tốt nhất (chi tiết sẽ được trình bày sau). Cách tiếp cận này hoạt động rất tốt, đặc biệt khi không có quá nhiều nhiễu, trong việc trải phẳng các đa tạp xoắn.

Đoạn mã dưới đây sử dụng lớp `LocallyLinearEmbedding` trong Scikit-Learn để trải phẳng tập dữ liệu Swiss roll ra:

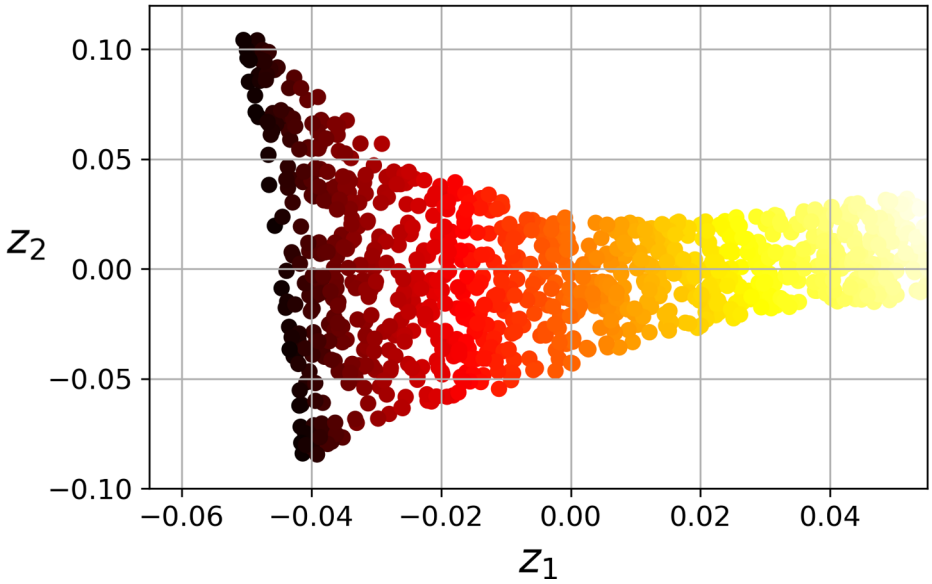
```
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
X_reduced = lle.fit_transform(X)
```

[7] Nếu ta đặt `fit_inverse_transform=True`, Scikit-Learn sẽ sử dụng thuật toán (dựa trên Hồi quy Ridge Hạt nhân) được mô tả trong bài báo của Gokhan H. Bakır và cộng sự, "Learning to Find Pre-Images", *Proceedings of the 16th International Conference on Neural Information Processing Systems* (2004): 449–456.

[8] Sam T. Roweis và Lawrence K. Saul, "Nonlinear Dimensionality Reduction by Locally Linear Embedding" *Science* 290, no. 5500 (2000): 2323–2326.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tập dữ liệu 2D thu được được minh hoạ trong Hình 8.12. Có thể thấy, Swiss roll hoàn toàn trải ra và khoảng cách cục bộ giữa các mẫu được bảo toàn rất tốt. Tuy nhiên, khoảng cách này lại không được bảo toàn trên quy mô lớn hơn: phần bên trái của Swiss roll được trải dãn ra, trong khi phần bên phải bị dồn lại. Tuy vậy, LLE đã mô hình hóa đa tạp này khá tốt.



Hình 8.12. Swiss roll được trải phẳng bằng LLE

LLE hoạt động như sau: Với mỗi mẫu huấn luyện $\mathbf{x}^{(i)}$, thuật toán sẽ xác định k điểm gần nhất (trong đoạn mã trên $k = 10$), sau đó tái tạo lại $\mathbf{x}^{(i)}$ như một tổ hợp tuyến tính của các điểm lân cận trên. Cụ thể hơn, thuật toán sẽ tìm các trọng số $w_{i,j}$ sao cho bình phương khoảng cách giữa $\mathbf{x}^{(i)}$ và $\sum_{j=1}^{m} w_{i,j}\mathbf{x}^{(j)}$ nhỏ nhất có thể, với giả định rằng $w_{i,j} = 0$ khi $\mathbf{x}^{(j)}$ không phải là một trong k điểm lân cận gần $\mathbf{x}^{(i)}$ nhất. Vì thế, bước đầu của LLE sẽ giải một bài toán tối ưu có ràng buộc được miêu tả trong Phương trình 8.4, với $\mathbf{W}$ là ma trận trọng số chứa tất cả trọng số $w_{i,j}$. Điều kiện ràng buộc thứ hai chỉ đơn giản là chuẩn hóa các trọng số cho mỗi mẫu huấn luyện $\mathbf{x}^{(i)}$.

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathbf{x}^{(i)} - \sum_{j=1}^{m} w_{i,j}\mathbf{x}^{(j)} \right)^2$$

$$\text{trong đó} \begin{cases} w_{i,j} = 0 & \text{nếu } \mathbf{x}^{(j)} \text{ không phải một trong } k \text{ mẫu gần } \mathbf{x}^{(i)} \text{ nhất} \\ \sum_{j=1}^{m} w_{i,j} = 1 & \text{với } i = 1, 2, \cdots, m \end{cases}$$

Phương trình 8.4. Bước 1 của thuật toán LLE: mô hình hóa tuyến tính các quan hệ cục bộ

Sau bước này, ma trận trọng số $\widehat{\mathbf{W}}$ (chứa các trọng số $\hat{w}_{i,j}$) sẽ mã hóa các quan hệ tuyến tính cục bộ giữa các mẫu huấn luyện. Trong bước thứ hai của LLE, các mẫu huấn luyện này sẽ được ánh xạ vào một không gian d-chiều (với $d < n$) trong khi cố gắng bảo toàn các quan hệ cục bộ nhiều nhất có thể. Nếu $\mathbf{z}^{(i)}$ là ảnh của $\mathbf{x}^{(i)}$ trong không gian d-chiều này, thì ta muốn bình phương khoảng cách giữa $\mathbf{z}^{(i)}$ và $\sum_{j=1}^{m} \hat{w}_{i,j}\mathbf{z}^{(j)}$ càng nhỏ càng tốt. Ý tưởng này dẫn đến bài toán tối ưu không ràng buộc được mô tả trong Phương trình 8.5. Bước này tương tự bước đầu tiên, nhưng thay vì giữ cho các mẫu cố định và tìm các trọng số tối ưu, ta làm ngược lại: giữ các

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

trọng số cố định và tìm vị trí tối ưu cho ảnh của các mẫu trong không gian ít chiều. Lưu ý rằng $\mathbf{Z}$ là ma trận chứa tất cả các $\mathbf{z}^{(i)}$.

$$\widehat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\mathbf{z}^{(i)} - \sum_{j=1}^{m} \widehat{w}_{i,j} \mathbf{z}^{(j)} \right)^2$$

Phương trình 8.5. Bước thứ hai của thuật toán LLE: giảm chiều trong khi bảo toàn các quan hệ.

Phiên bản trong Scikit-Learn của thuật toán LLE có độ phức tạp tính toán $O(m \log(m) n \log(k))$ cho bước tìm k điểm gần nhất, $O(mnk^3)$ cho bước tối ưu trọng số, và $O(dm^2)$ cho bước xây dựng các đặc trưng ít chiều. Đáng tiếc là độ phức tạp tỉ lệ với m^2 ở bước cuối cùng khiến cho thuật toán này khó triển khai trên các tập dữ liệu rất lớn.

Các Kỹ thuật Giảm chiều Khác

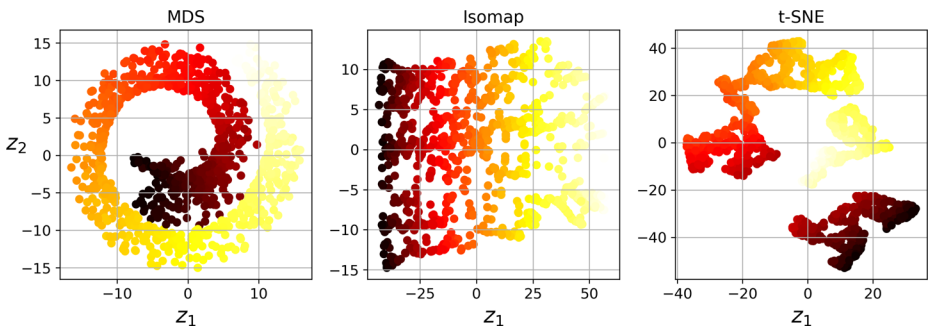
Còn rất nhiều kỹ thuật giảm chiều khác, một vài trong số đó đã được lập trình sẵn trong Scikit-Learn. Dưới đây là một vài kỹ thuật phổ biến:

- *Phép Chiếu Ngẫu nhiên*: Đúng như tên gọi, đây là phép chiếu dữ liệu sang một không gian ít chiều hơn bằng phép chiếu tuyến tính ngẫu nhiên. Điều này nghe có vẻ hơi lạ, nhưng hóa ra phép chiếu ngẫu nhiên như vậy lại rất có khả năng sẽ bảo toàn khoảng cách giữa các mẫu – điều đã được chứng minh toán học bởi William B. Johnson và Joram Lindenstrauss trong một bổ đề nổi tiếng. Tuy chất lượng phép giảm chiều phụ thuộc vào số mẫu và số chiều mục tiêu, nhưng điều đáng ngạc nhiên là kỹ thuật này lại không phụ thuộc vào số chiều ban đầu. Để biết thêm chi tiết, hãy tham khảo tài liệu về gói `sklearn.random_projection`.
- *Co giãn Đa chiều* (*Multidimensional Scaling – MDS*): Giảm chiều nhưng vẫn cố gắng bảo toàn khoảng cách giữa các mẫu.
- *Isomap*: Tạo đồ thị kết nối từng mẫu với các mẫu gần nhất của nó, sau đó giảm chiều trong khi cố gắng bảo toàn khoảng cách trắc địa (*geodesic distance*)[9] giữa các mẫu.
- *Embedding Lân cận Ngẫu nhiên theo Phân phối t* (*t-Distributed Stochastic Neighbor Embedding* – t-SNE): Giảm chiều trong khi cố gắng giữ cho các mẫu tương đồng gần nhau và các mẫu khác biệt cách xa nhau. Kỹ thuật này chủ yếu được sử dụng để trực quan hóa, đặc biệt là với các cụm trong không gian nhiều chiều (ví dụ: để trực quan hóa các ảnh MNIST trong không gian 2D).
- *Phân tích Phân biệt Tuyến tính* (*Linear Discriminant Analysis* – LDA): Là một thuật toán phân loại, nhưng trong quá trình huấn luyện, thuật toán sẽ tìm các trục cho độ tách biệt cao nhất giữa các lớp. Các trục này sau đó có thể được sử dụng để xác định một siêu phẳng để chiếu dữ liệu. Với ưu điểm là khả năng giữ các lớp xa nhau nhất có thể, LDA là một kỹ thuật tốt để giảm chiều trước khi chạy thuật toán phân loại khác, chẳng hạn như SVM.

Hình 8.13 biểu diễn kết quả thu được của một vài kỹ thuật trên.

[9] Khoảng cách trắc địa giữa hai nút trong đồ thị là số nút nằm trên đường đi ngắn nhất giữa hai nút đó.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 8.13. Sử dụng các kỹ thuật khác nhau để đưa tập dữ liệu Swiss roll xuống 2 chiều

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bài tập

1. Động lực chính để giảm chiều dữ liệu là gì? Những hạn chế chính của các kỹ thuật giảm chiều là gì?
2. Lời nguyền chiều dữ liệu là gì?
3. Khi giảm chiều của tập dữ liệu xong, liệu ta có thể đảo ngược lại phép biến đổi này không? Nếu có, hãy mô tả cách đảo ngược này? Nếu không, hãy giải thích tại sao?
4. Liệu PCA có thể được sử dụng để giảm chiều của tập dữ liệu có tính phi tuyến cao không?
5. Giả sử, ta vừa thực hiện PCA trên tập dữ liệu 1,000 chiều với tỉ lệ phương sai được giải thích bằng 95%. Tập dữ liệu thu được có bao nhiêu chiều?
6. Trong những trường hợp nào ta nên chọn kỹ thuật PCA thông thường, PCA Tăng dần, PCA Ngẫu nhiên, PCA Hạt nhân?
7. Ta thực hiện đánh giá chất lượng của một thuật toán giảm chiều trên một tập dữ liệu như thế nào?
8. Liệu việc sử dụng liên tiếp hai thuật toán giảm chiều khác nhau có ý nghĩa không?
9. Nạp tập dữ liệu MNIST (được giới thiệu trong Chương 3) và chia nó thành tập huấn luyện và tập kiểm tra (tập huấn luyện có 60,000 mẫu trong khi tập kiểm tra chứa 10,000 mẫu còn lại). Hãy huấn luyện bộ phân loại Rừng Ngẫu nhiên trên tập dữ liệu và xem thời gian huấn luyện mất bao lâu, sau đó đánh giá mô hình thu được trên tập kiểm tra. Tiếp theo, sử dụng PCA để giảm chiều của tập dữ liệu, với tỷ lệ phương sai được giải thích là 95%. Hãy huấn luyện một bộ phân loại Rừng Ngẫu nhiên mới trên tập dữ liệu đã giảm chiều và xem thời gian huấn luyện mất bao lâu. Liệu quá trình huấn luyện có nhanh hơn nhiều không? Tiếp đến, hãy đánh giá bộ phân loại thu được trên tập kiểm tra. Chất lượng của bộ phân loại này so với bộ phân loại trước đó như thế nào?
10. Hãy sử dụng t-SNE để đưa tập dữ liệu MNIST xuống không gian hai chiều và biểu diễn kết quả thu được bằng Matplotlib. Bạn có thể sử dụng biểu đồ phân tán (*scatterplot*) với 10 màu khác nhau để biểu diện cho lớp mục tiêu của mỗi ảnh. Ngoài ra, bạn có thể thay thế mỗi điểm trong biểu đồ phân tán bằng lớp của mẫu tương ứng (chữ số từ 0 đến 9), hay thậm chí vẽ các phiên bản thu nhỏ của chính các hình ảnh chữ số (nếu bạn vẽ tất cả các chữ số, biểu đồ sẽ rất lộn xộn, vì vậy bạn nên vẽ một mẫu ngẫu nhiên hoặc chỉ vẽ một mẫu nếu không có mẫu nào khác đã được vẽ gần đó). Bạn sẽ thu được một biểu đồ đẹp mắt với các cụm chữ số được phân tách rõ ràng. Hãy thử sử dụng các thuật toán giảm chiều khác như PCA, LLE, hoặc MDS, và so sánh các trực quan biểu đồ thu được.

Lời giải cho các bài tập trên được cung cấp sẵn trong Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chương 9

Các kỹ thuật Học Không giám sát

Mặc dù hầu hết các ứng dụng của Học Máy ngày nay đều dựa trên học có giám sát (vì vậy phương pháp này thu hút khá nhiều khoản đầu tư), phần lớn dữ liệu khả dụng chưa được gán nhãn: ta có đặc trưng đầu vào $\mathbf{X}$, nhưng không có nhãn tương ứng $\mathbf{y}$. Nhà khoa học máy tính Yann LeCun có một câu nói nổi tiếng "nếu trí thông minh là một chiếc bánh, học không giám sát sẽ là nhân bánh, học có giám sát là lớp kem, và học tăng cường là trái anh đào đặt trên đó". Nói cách khác, ta chỉ mới khai phá một phần rất nhỏ tiềm năng to lớn của học không giám sát mà thôi.

Giả sử ta muốn tạo một hệ thống phát hiện sản phẩm lỗi trên dây chuyền sản xuất bằng cách chụp và xử lý một vài ảnh của từng sản phẩm. Ta có thể dễ dàng tạo một hệ thống chụp ảnh tự động để thu thập hàng nghìn ảnh mỗi ngày. Như vậy, chỉ trong vòng vài tuần ta đã có thể xây dựng một tập dữ liệu khá lớn. Nhưng khoan, tập dữ liệu này không có nhãn! Nếu muốn huấn luyện một bộ phân loại nhị phân thông thường để dự đoán liệu một sản phẩm có bị lỗi hay không, ta sẽ cần gán nhãn mỗi bức ảnh là "bị lỗi" hoặc "bình thường". Để làm được điều này, ta cần các chuyên gia xem xét thủ công tất cả các ảnh. Đây là một công việc mất thời gian, tốn kém, và tẻ nhạt. Vì vậy, việc này chỉ thường được thực hiện trên một tập con nhỏ của các bức ảnh đang có. Kết quả là tập dữ liệu được gán nhãn sẽ có kích thước khá nhỏ và chất lượng phân loại sẽ không tốt. Thêm vào đó, mỗi khi sản phẩm thay đổi thì toàn bộ quá trình này sẽ phải được thực hiện lại từ đầu. Sẽ thật tuyệt vời nếu có một thuật toán có thể khai thác dữ liệu không nhãn mà không cần con người gán nhãn cho mọi bức ảnh. Học không giám sát chính là lời giải cho vấn đề này.

Trong Chương 8, ta đã xem xét tác vụ học không giám sát phổ biến nhất: giảm chiều (*dimensionality reduction*). Chương này sẽ thảo luận nhiều hơn về các tác vụ và thuật toán học không giám sát khác, bao gồm:

***Phân cụm* (*clustering*)**
Mục tiêu là nhóm các mẫu giống nhau thành các *cụm* (*cluster*). Phân cụm là một công cụ tuyệt vời cho phân tích dữ liệu, phân nhóm khách hàng, hệ thống đề xuất, công cụ tìm kiếm, phân vùng ảnh, học bán giám sát, giảm chiều, và nhiều tác vụ khác.

***Phát hiện bất thường* (*anomaly detection*)**
Mục tiêu là hiểu được dữ liệu "bình thường" trông như thế nào, sau đó sử dụng kiến thức này để phát hiện các mẫu bất thường, chẳng hạn như sản phẩm lỗi trên dây chuyền sản xuất hoặc xu hướng mới trong một chuỗi thời gian.

***Ước lượng mật độ* (*density estimation*)**
Đây là tác vụ ước lượng *hàm mật độ xác suất* (*probability density function – PDF*) của quá

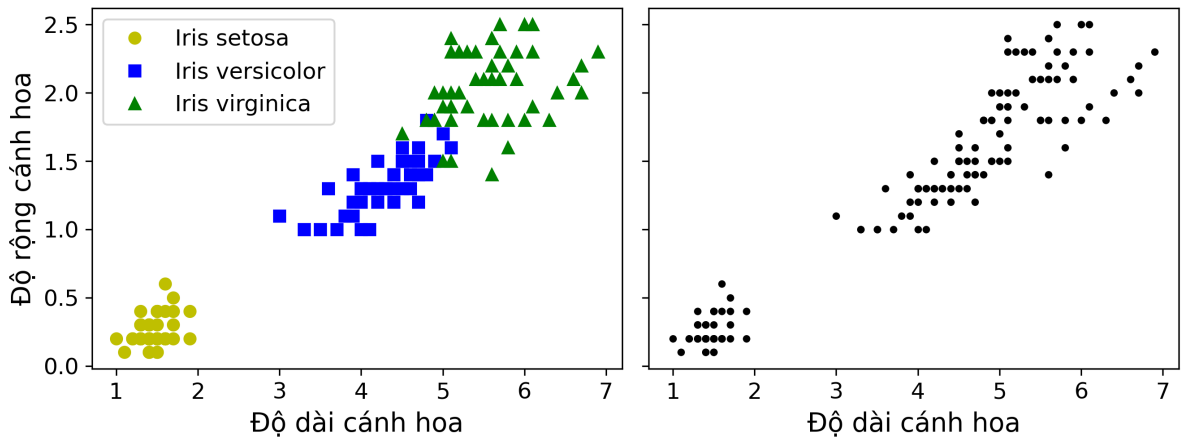
trình ngẫu nhiên đã sinh ra tập dữ liệu. Ước lượng mật độ thường được sử dụng để phát hiện bất thường: các mẫu nằm trong vùng mật độ thấp có khả năng cao là các điểm bất thường. Nó cũng rất hữu ích trong việc phân tích và trực quan hóa dữ liệu.

Giờ bạn đã sẵn sàng để khám phá nhân bánh chưa? Ta sẽ bắt đầu với phân cụm, sử dụng K-Điểm trung bình và DBSCAN. Sau đó ta sẽ thảo luận về mô hình hỗn hợp Gauss và cách mà mô hình này được sử dụng để ước lượng mật độ, phân cụm và phát hiện bất thường.

Phân cụm

Khi đang leo núi, ta tình cờ gặp một loài thực vật chưa từng thấy trước đây. Sau khi quan sát xung quanh, ta tìm thấy thêm một vài cá thể giống như vậy. Chúng không hoàn toàn giống nhau nhưng cũng đủ tương đồng để có thể phân nhóm về cùng một loài (hoặc ít nhất là cùng một chi). Ta sẽ cần một nhà thực vật học để biết cụ thể đó là loài thực vật gì, tuy nhiên ta không cần phải là chuyên gia để xác định các nhóm vật thể có điểm giống nhau. Tác vụ này được gọi là *phân cụm* (*clustering*): đây là tác vụ xác định các mẫu tương tự nhau và phân loại chúng vào các *cụm* (*cluster*), hoặc các nhóm tương đồng nhau.

Cũng giống như tác vụ phân loại, mỗi mẫu được gán cho một nhóm. Tuy nhiên, khác với phân loại, phân cụm là tác vụ không giám sát. Xét Hình 9.1: đồ thị bên trái là tập dữ liệu Iris (được giới thiệu trong Chương 4), trong đó chủng loài của mỗi mẫu (nhãn của nó) được minh họa bằng các hình khác nhau. Tập dữ liệu Iris là một tập dữ liệu được gán nhãn rất phù hợp cho các thuật toán phân loại như Hồi quy Logistic, SVM, hoặc Rừng Ngẫu nhiên. Đồ thị bên phải là cũng chính là tập dữ liệu này nhưng không chứa nhãn, vì vậy ta không thể sử dụng các thuật toán phân loại nữa. Đây là khi các thuật toán phân cụm trở nên hữu ích: rất nhiều thuật toán phân cụm có thể dễ dàng phát hiện ra cụm ở góc dưới bên trái. Ta cũng có thể dễ dàng nhìn thấy cụm này bằng mắt thường, nhưng hai cụm con thuộc cụm lớn ở góc phía trên bên phải thì không thật sự rõ ràng. Dù vậy, cũng cần lưu ý rằng có hai đặc trưng bổ sung (chiều dài và chiều rộng đài hoa) không được biểu diễn ở đây, và các thuật toán phân cụm có thể tận dụng tất cả các đặc trưng, nên thực tế, các thuật toán này xác định cả ba cụm tương đối tốt (ví dụ như khi thử nghiệm với mô hình hỗn hợp Gauss, chỉ có 5 mẫu trên 150 bị đặt sai cụm).



Hình 9.1. Tác vụ phân loại (trái) và phân cụm (phải)

Phân cụm được sử dụng rộng rãi trong nhiều ứng dụng, bao gồm:

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phân nhóm khách hàng
Ta có thể phân cụm khách hàng dựa vào giao dịch và hoạt động của họ trên website. Việc này giúp ta hiểu được khách hàng là ai và họ cần gì, từ đó vạch ra chiến lược marketing và tung ra sản phẩm phù hợp cho từng phân khúc. Ví dụ, phân nhóm khách hàng có thể hữu dụng trong *hệ thống đề xuất* để đề xuất các nội dung được ưa chuộng bởi những người dùng khác trong cụm.

Phân tích dữ liệu
Khi nghiên cứu một tập dữ liệu mới, sẽ khá hữu ích nếu ta áp dụng thuật toán phân cụm với tập dữ liệu này, sau đó tiến hành phân tích từng cụm riêng biệt.

Kỹ thuật giảm chiều
Một khi tập dữ liệu được phân cụm, ta có thể đo lường *ái lực* (*affinity*) của mỗi mẫu với từng cụm (ái lực là bất kỳ đại lượng nào cho biết mức độ phù hợp của một mẫu với một cụm). Mỗi đặc trưng vector $\mathbf{x}$ của một mẫu có thể được thay thế bằng một vector biểu diễn ái lực của nó đến các cụm. Nếu ta có k cụm, thì vector này có k chiều. Vector này thường có chiều thấp hơn nhiều so với vector đặc trưng ban đầu, nhưng vẫn bảo toàn đủ thông tin để tiếp tục xử lý.

Phát hiện bất thường (còn gọi là phát hiện ngoại lai)
Bất kỳ mẫu nào có ái lực thấp với tất cả các cụm đều có khả năng cao là một điểm bất thường. Ví dụ, nếu phân cụm những người dùng trang web dựa trên hành vi của họ, ta có thể phát hiện những người dùng với hành vi bất thường, chẳng hạn như số lượng yêu cầu được thực hiện trong mỗi giây cao bất thường. Phát hiện bất thường rất hữu dụng để phát hiện lỗi trong quá trình sản xuất, hoặc để *phát hiện gian lận* (*fraud detection*).

Học bán giám sát
Nếu có ít nhãn, ta có thể thực hiện phân cụm và lan truyền nhãn tới tất cả các mẫu trong cùng một cụm. Kỹ thuật này có thể tăng đáng kể số lượng nhãn có sẵn để sử dụng cho thuật toán học có giám sát về sau, từ đó cải thiện chất lượng mô hình.

Công cụ tìm kiếm
Một số công cụ tìm kiếm cho phép ta tìm kiếm những hình ảnh tương đồng với ảnh tham chiếu. Để xây dựng hệ thống như vậy, trước hết ta cần áp dụng thuật toán phân cụm với tất cả các hình ảnh trong tập dữ liệu, rồi các ảnh tương tự sẽ được gom về cùng một cụm. Sau đó, khi người dùng cung cấp một ảnh tham chiếu, ta chỉ cần sử dụng mô hình phân cụm đã huấn luyện để tìm cụm của hình ảnh này, rồi đơn thuần trả về tất cả các hình ảnh trong cụm đó.

Phân vùng ảnh
Bằng cách phân cụm các điểm ảnh theo màu sắc của chúng, sau đó thay thế màu của điểm ảnh bằng màu trung bình của cụm chứa điểm ảnh đó, ta có thể giảm đáng kể số lượng màu khác nhau trong ảnh. Phân vùng ảnh được sử dụng bởi nhiều hệ thống phát hiện và theo dõi vật thể nhờ tính hiệu quả trong việc phát hiện đường bao của các vật thể đó.

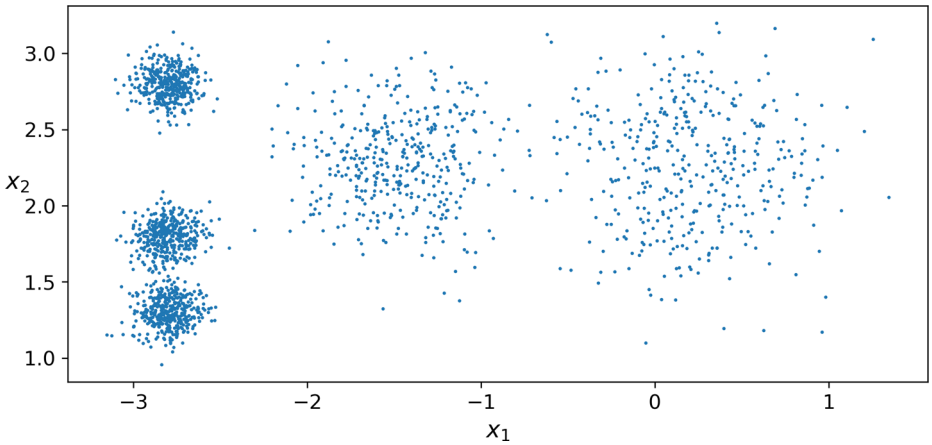
Không tồn tại định nghĩa tổng quát để giải thích như thế nào là một cụm: khái niệm này phụ thuộc vào ngữ cảnh, các thuật toán khác nhau sẽ xác định được các loại cụm khác nhau. Một số thuật toán tìm kiếm các mẫu tập trung xung quanh một điểm nhất định gọi là *tâm cụm* (*centroid*). Một số khác thì tìm kiếm các vùng liên tục chứa rất nhiều mẫu nằm cạnh nhau: các cụm này có thể có bất kỳ hình dạng nào. Một số thuật toán tiếp cận theo hướng phân cấp, tìm kiếm các cụm của cụm. Và còn rất nhiều loại cụm khác nữa.

Trong phần này, ta sẽ xem xét hai thuật toán phân cụm phổ biến là K-Điểm trung bình và DBSCAN, đồng thời khám phá một số ứng dụng của chúng, chẳng hạn như giảm chiều phi tuyến, học bán giám sát, và phát hiện bất thường.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

K-Điểm trung bình

Hãy xem xét tập dữ liệu không nhãn được trình bày trong Hình 9.2: ta có thể dễ dàng nhận ra năm khối mẫu riêng biệt. Thuật toán K-Điểm trung bình là một thuật toán đơn giản có khả năng phân cụm tập dữ liệu dạng này một cách nhanh chóng và hiệu quả, thường chỉ trong một vài vòng lặp. K-Điểm trung bình được đề xuất bởi Stuart Lloyd tại Bell Labs vào năm 1957 như một kỹ thuật để điều chế mã xung (*pulse code modulation*), nhưng phải đến năm 1982 thì nó mới được xuất bản ra ngoài.[1] Năm 1965, Edward W. Forgy đã công bố một thuật toán gần như tương tự, vì vậy K-Điểm trung bình đôi khi còn được gọi là Lloyd-Forgy.



Hình 9.2. Một tập dữ liệu không nhãn được tạo bởi năm khối mẫu

Hãy huấn luyện một bộ phân cụm K-Điểm trung bình với tập dữ liệu trên. Bộ phân cụm này sẽ cố gắng tìm trung tâm của từng khối mẫu và gán mỗi mẫu với khối gần nó nhất.

```
from sklearn.cluster import KMeans
k = 5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

Lưu ý rằng ta cần chỉ rõ số lượng cụm k mà thuật toán cần tìm. Trong ví dụ này, có thể dễ dàng nhận thấy k nên được đặt bằng 5. Tuy nhiên, không phải lúc nào cũng đơn giản như vậy. Vấn đề này sẽ được thảo luận kỹ hơn sau.

Mỗi mẫu được gán cho một trong năm cụm. Trong ngữ cảnh phân cụm, *nhãn* của một mẫu là chỉ số của cụm mà mẫu này được gán: lưu ý rằng nhãn cụm và nhãn lớp hoàn toàn khác nhau (nhớ rằng phân cụm là tác vụ học không giám sát). Thực thể `KMeans` lưu giữ một bản sao nhãn của các mẫu huấn luyện, có thể được trích xuất thông qua thuộc tính `labels_`:

```
>>> y_pred
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
>>> y_pred is kmeans.labels_
True
```

Ta cũng có thể biết năm tâm cụm tìm được:

[1] Stuart P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory* 28, no. 2 (1982): 129–137.

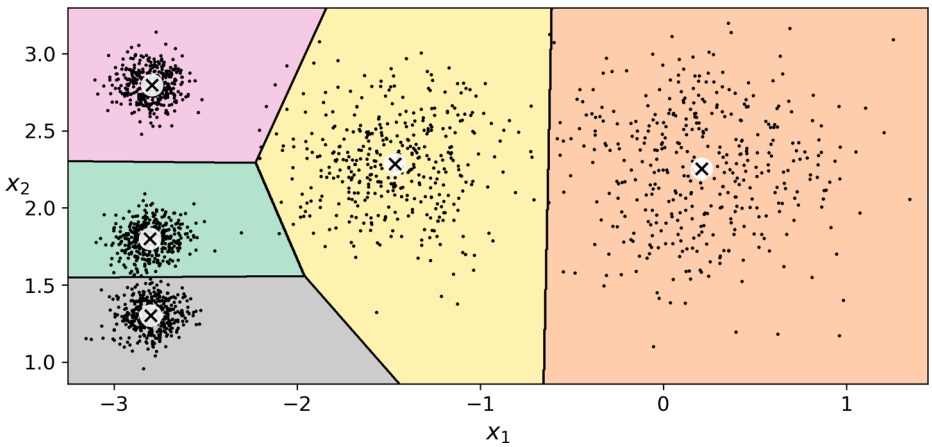
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> kmeans.cluster_centers_
array([[-2.80389616,  1.80117999],
       [ 0.20876306,  2.25551336],
       [-2.79290307,  2.79641063],
       [-1.46679593,  2.28585348],
       [-2.80037642,  1.30082566]])
```

Có thể dễ dàng gán các mẫu mới cho cụm có tâm gần nhất:

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```

Nếu vẽ ranh giới quyết định của các cụm, ta sẽ có một lưới Voronoi (*Voronoi tessellation* – xem Hình 9.3, mỗi tâm cụm được biểu diễn bởi ký tự **x**).



Hình 9.3. Ranh giới quyết định của K-Điểm trung bình (dạng lưới Voronoi)

Phần lớn các mẫu sẽ được gán cho cụm phù hợp, tuy nhiên vẫn có một vài mẫu bị gán nhãn sai (đặc biệt là ở khu vực gần ranh giới giữa cụm góc trên-trái và cụm trung tâm.) Thật vậy, thuật toán K-Điểm trung bình không hoạt động tốt khi các khối có đường kính khác nhau. Lý do là nó chỉ dựa vào khoảng cách đến tâm cụm để gán nhãn cho một mẫu.

Thay vì gán mỗi mẫu cho chỉ một cụm, còn được gọi là *phân cụm cứng*, ta có thể cung cấp điểm số tương quan tới từng cụm cho mỗi mẫu, hay còn gọi là *phân cụm mềm*. Điểm số này có thể là khoảng cách giữa mẫu và tâm cụm. Ngược lại, nó cũng có thể là điểm số tương đồng (hay ái lực), có thể được tính thông qua hàm RBF Gauss (*Gaussian Radial Basis Function* – giới thiệu trong Chương 5). Trong lớp `KMeans`, phương thức `transform()` tính khoảng cách của mỗi mẫu đến tất cả các tâm cụm:

```
>>> kmeans.transform(X_new)
array([[2.81093633, 0.32995317, 2.9042344 , 1.49439034, 2.88633901],
       [5.80730058, 2.80290755, 5.84739223, 4.4759332 , 5.84236351],
       [1.21475352, 3.29399768, 0.29040966, 1.69136631, 1.71086031],
       [0.72581411, 3.21806371, 0.36159148, 1.54808703, 1.21567622]])
```

Trong ví dụ này, mẫu đầu tiên trong `X_new` cách tâm cụm đầu tiên một khoảng 2.81, cách tâm cụm thứ hai một khoảng 0.33, cách tâm cụm thứ ba một khoảng 2.90, cách tâm cụm thứ tư

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

một khoảng 1.49, và cách tâm cụm thứ năm một khoảng 2.89. Nếu ta biến đổi một tập dữ liệu nhiều chiều theo phương pháp này, ta sẽ thu được một tập dữ liệu k chiều: đây có thể là một kỹ thuật giảm chiều phi tuyến rất hiệu quả.

Thuật toán K-Điểm trung bình

Vậy thuật toán K-Điểm trung bình hoạt động như thế nào? Giả sử ta đã biết trước các tâm cụm, vậy ta có thể dễ dàng gán nhãn từng mẫu trong tập dữ liệu vào cụm có tâm gần nhất. Ngược lại, nếu biết trước nhãn của tất cả các mẫu, ta có thể dễ dàng xác định vị trí các tâm cụm bằng cách tính trung bình các mẫu trong từng cụm. Tuy nhiên, không may là ta không biết trước cả nhãn và tâm cụm. Nhưng ta có thể bắt đầu bằng cách chọn ngẫu nhiên các tâm cụm (ví dụ như chọn ngẫu nhiên k mẫu và sử dụng vị trí của chúng làm tâm cụm). Tiếp theo, ta gán nhãn các mẫu, cập nhật các tâm cụm, tiếp tục gán nhãn các mẫu, cập nhật lại các tâm cụm, và lặp lại cho đến khi các tâm cụm ngừng thay đổi. Thuật toán này sẽ không dao động vĩnh viễn, mà chắc chắn sẽ hội tụ sau một số lượng bước nhất định (thường rất ít).[2]

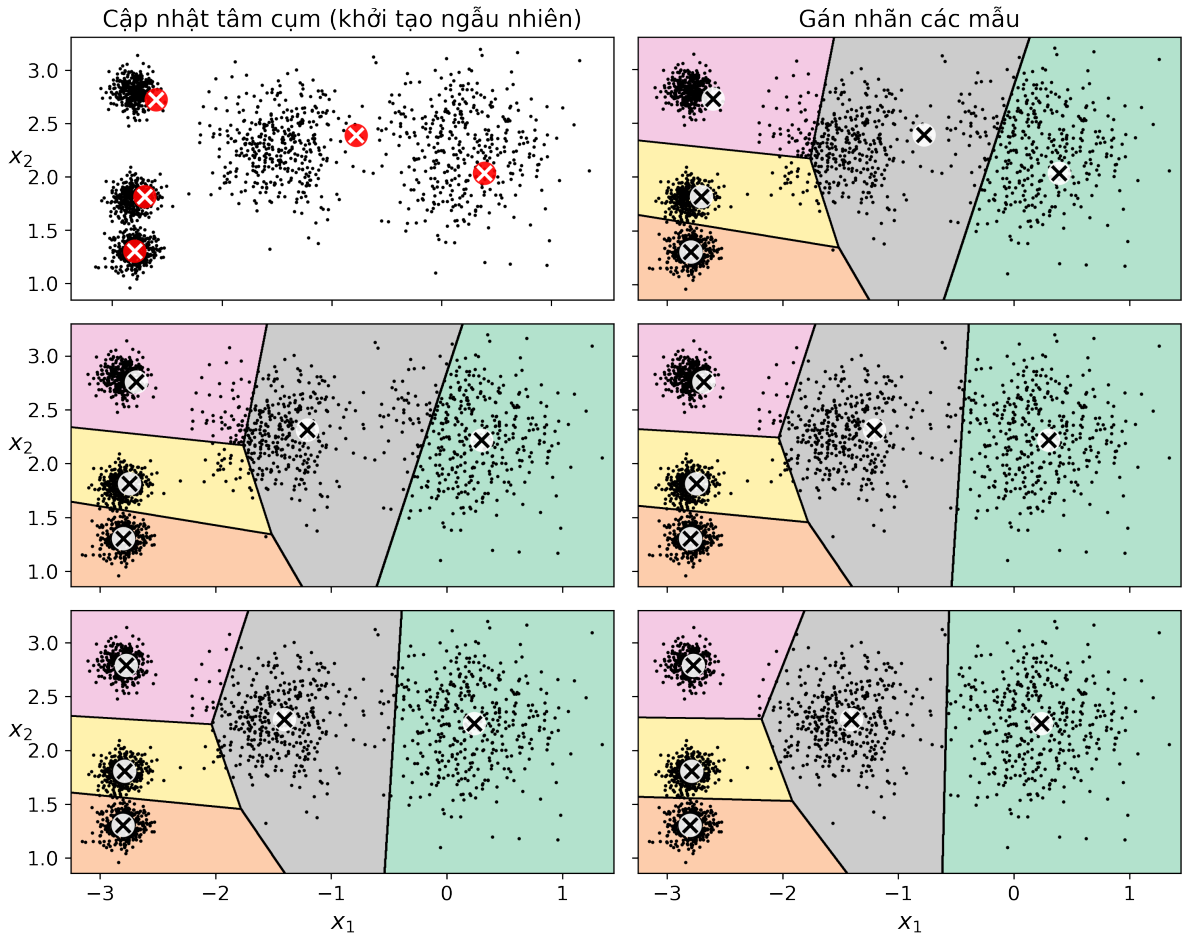
Ta có thể quan sát cách thuật toán hoạt động tại Hình 9.4: các tâm cụm được khởi tạo ngẫu nhiên (biểu đồ trên cùng bên trái), sau đó các mẫu được gán nhãn (biểu đồ trên cùng bên phải), kế tiếp các tâm cụm được cập nhật (biểu đồ giữa bên trái), các mẫu được gán nhãn lại (biểu đồ giữa bên phải), và lặp lại. Có thể thấy rằng chỉ sau ba vòng lặp, các cụm của thuật toán đã đạt tới gần điểm tối ưu.

Ghi chú

Nhìn chung thì độ phức tạp tính toán của thuật toán này tuyến tính với số lượng mẫu m, số cụm k, và số chiều n. Tuy nhiên, điều này chỉ đúng khi dữ liệu có cấu trúc dạng phân cụm. Với tập dữ liệu thuộc dạng khác, độ phức tạp tính toán có thể tăng theo cấp số mũ với số lượng mẫu trong trường hợp xấu nhất. Thực tế thì điều này hiếm khi xảy ra, và nhìn chung K-Điểm trung bình là một trong những thuật toán phân cụm nhanh nhất.

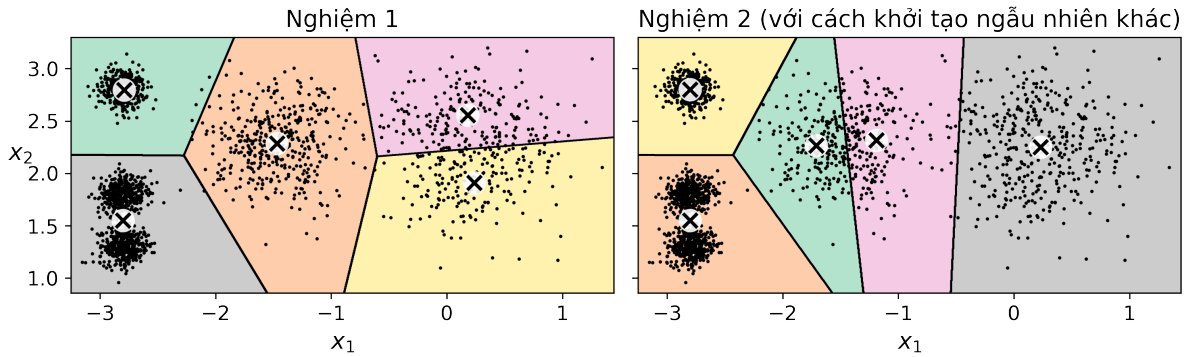
[2] Lý do là bởi trung bình bình phương khoảng cách giữa các mẫu và tâm cụm gần nhất chỉ có thể giảm đi sau mỗi bước.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 9.4. Thuật toán K-Điểm trung bình

Mặc dù thuật toán chắc chắn hội tụ, kết quả có thể sẽ không tối ưu (nó có thể hội tụ ở một điểm tối ưu cục bộ). Việc có tìm được nghiệm tốt hay không phụ thuộc vào cách khởi tạo tâm cụm. Hình 9.5 biểu diễn hai nghiệm chưa tối ưu mà thuật toán có thể hội tụ nếu ta không gặp may tại bước khởi tạo ngẫu nhiên.



Hình 9.5. Các nghiệm chưa tối ưu trong trường hợp khởi tạo tâm cụm không tốt

Hãy xem xét một số phương án có thể được sử dụng để giảm thiểu rủi ro này bằng cách cải

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

thiện việc khởi tạo tâm cụm.

Các phương pháp khởi tạo tâm cụm

Nếu ta tình cờ biết được vị trí gần đúng của các tâm cụm tối ưu (ví dụ như khi đã chạy một thuật toán phân cụm trước đó), thì ta có thể gán siêu tham số `init` là một mảng Numpy chứa danh sách các vị trí đó, và đặt `n_init` bằng 1:

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

Một giải pháp khác là khởi tạo ngẫu nhiên và chạy thuật toán nhiều lần, rồi lưu nghiệm tốt nhất. Số lần khởi tạo ngẫu nhiên được kiểm soát bởi tham số `n_init`: tham số này mặc định bằng 10, có nghĩa là toàn bộ thuật toán sẽ chạy 10 lần khi ta gọi phương thức `fit()`, và Scikit-Learn sẽ giữ lại nghiệm tốt nhất. Nhưng làm thế nào mà Scikit-Learn biết đâu là nghiệm tốt nhất? Bằng cách dùng một phép đo chất lượng! Phép đo này có tên gọi là *inertia*, được định nghĩa là trung bình bình phương khoảng cách từ mỗi mẫu đến tâm cụm gần nhất. Đại lượng này gần bằng 223.3 cho mô hình bên trái trong Hình 9.5, 237.5 cho mô hình bên phải trong Hình 9.5, và 211.6 cho mô hình tại Hình 9.3. Lớp `KMeans` chạy thuật toán này `n_init` lần và giữ lại mô hình có inertia thấp nhất. Trong ví dụ này, mô hình tại Hình 9.3 sẽ được chọn (trừ khi chúng ta không gặp may với toàn bộ `n_init` lần khởi tạo liên tiếp). Ta có thể biết inertia của một mô hình nhờ thuộc tính `inertia_`:

```
>>> kmeans.inertia_
211.59853725816856
```

Hàm `score()` trả về inertia âm. Tại sao lại âm? Bởi vì hàm `score()` của bộ dự đoán phải luôn tuân thủ quy tắc "càng lớn càng tốt" của Scikit-Learn: nếu một bộ dự đoán tốt hơn các bộ khác thì hàm `score()` của nó sẽ trả về giá trị cao hơn.

```
>>> kmeans.score(X)
-211.59853725816856
```

Một cải tiến quan trọng cho thuật toán K-Điểm trung bình, *K-Điểm trung bình++* (*K-Means++*), đã được đề xuất trong bài báo năm 2007 của David Arthur và Sergei Vassilvitskii.[3] Các tác giả đã giới thiệu một bước khởi tạo thông minh hơn nhằm chọn các tâm cụm cách xa nhau. Cải tiến này giúp thuật toán K-Điểm trung bình ít có khả năng hội tụ tại một nghiệm chưa tối ưu. Họ đã chỉ ra rằng dù việc khởi tạo thông minh cần thêm thời gian tính toán, tổng thời gian vẫn sẽ giảm vì số lần chạy thuật toán cần thiết để tìm được nghiệm tối ưu có thể giảm đi đáng kể. Thuật toán khởi tạo này hoạt động như sau:

1. Chọn ngẫu nhiên một tâm cụm $\mathbf{c}^{(1)}$ theo phân phối đều từ tập dữ liệu.
2. Lấy một tâm cụm mới $\mathbf{c}^{(i)}$ bằng cách chọn một mẫu $\mathbf{x}^{(i)}$ với xác suất: $\frac{D(\mathbf{x}^{(i)})^2}{\sum_{j=1}^{m} D(\mathbf{x}^{(j)})^2}$, trong đó $D(\mathbf{x}^{(i)})$ là khoảng cách giữa mẫu $\mathbf{x}^{(i)}$ và tâm cụm đã được chọn gần nó nhất. Phân phối xác suất này đảm bảo rằng các mẫu xa các tâm cụm đã được chọn có khả năng được chọn làm tâm cụm cao hơn.

[3] David Arthur và Sergei Vassilvitskii, "k-Means++: The Advantages of Careful Seeding," *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms* (2007): 1027–1035.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

3. Lặp lại bước trên cho tới khi chọn được hết k tâm cụm.

Lớp `KMeans` mặc định sử dụng phương pháp khởi tạo này. Nếu muốn sử dụng phương pháp gốc (tức khởi tạo k mẫu ngẫu nhiên làm tâm cụm), ta có thể đặt siêu tham số `init` là `"random"`, nhưng hiếm khi ta cần làm điều này.

K-Điểm trung bình Tăng tốc và K-Điểm trung bình mini-batch

Một cải tiến quan trọng khác cho thuật toán K-Điểm trung bình được đề xuất trong bài báo năm 2003 của Charles Elkan.[4] Cải tiến này tăng tốc đáng kể thuật toán bằng việc tránh nhiều phép tính khoảng cách không cần thiết. Elkan thực hiện điều này bằng cách tận dụng bất đẳng thức tam giác (đoạn thẳng luôn là khoảng cách ngắn nhất giữa hai điểm[5]) và đồng thời lưu lại các cận trên và dưới cho các khoảng cách giữa các mẫu và tâm cụm. Đây là thuật toán mà lớp `KMeans` sử dụng làm mặc định (ta có thể sử dụng thuật toán gốc bằng cách đặt siêu tham số `algorithm` là `"full"`, nhưng không có lý do gì để làm điều này).

Một biến thể quan trọng khác của thuật toán K-Điểm trung bình được đề xuất trong bài báo năm 2010 của David Sculley.[6] Thay vì sử dụng toàn bộ tập dữ liệu ở mỗi vòng lặp, thuật toán có thể sử dụng các mini-batch, di chuyển các tâm cụm một chút ở mỗi vòng lặp. Điều này thường tăng tốc thuật toán ba hoặc bốn lần và cho phép ta phân cụm các tập dữ liệu khổng lồ vượt quá dung lượng bộ nhớ. Thuật toán này được lập trình trong lớp `MiniBatchKMeans` của Scikit-Learn. Ta có thể sử dụng lớp này tương tự như lớp `KMeans`:

```
from sklearn.cluster import MiniBatchKMeans

minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

Nếu tập dữ liệu không nằm vừa trong bộ nhớ, lựa chọn đơn giản nhất là sử dụng lớp `memmap`, như ta đã làm với PCA gia tăng ở Chương 8. Ngoài ra, ta có thể truyền từng mini-batch một vào hàm `partial_fit()`, nhưng việc này đòi hỏi nhiều công đoạn hơn vì ta cần thực hiện nhiều lần khởi tạo và tự chọn lần khởi tạo tốt nhất (xem ví dụ ở phần K-Điểm trung bình mini-batch trong notebook).

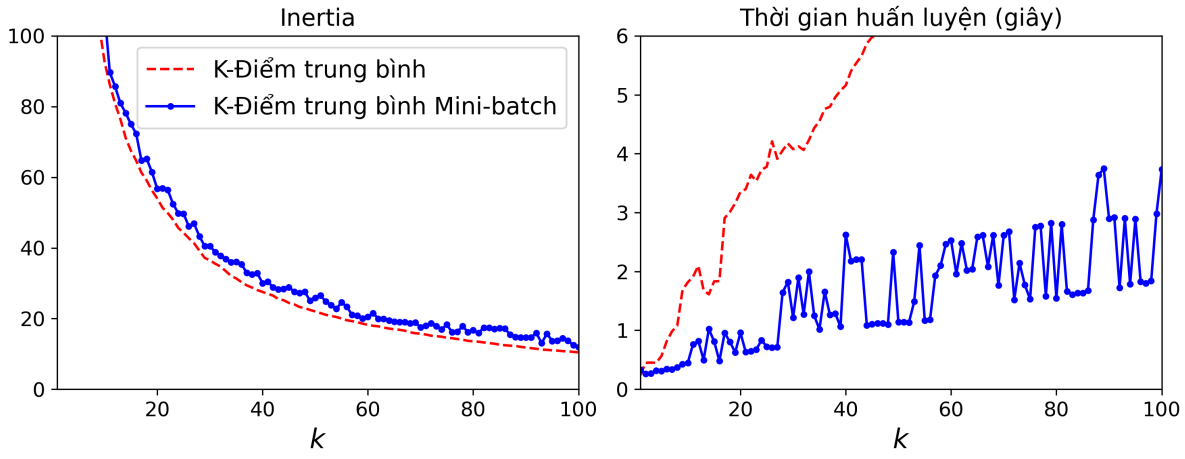
Mặc dù thuật toán K-Điểm trung bình Mini-batch nhanh hơn nhiều so với thuật toán K-Điểm trung bình thông thường, inertia của nó thường thấp hơn một chút, đặc biệt là khi số lượng cụm tăng. Ta có thể thấy điều này ở Hình 9.6: đồ thị bên trái so sánh inertia của K-Điểm trung bình Mini-batch và K-Điểm trung bình thông thường, được huấn luyện trên tập dữ liệu đã đề cập ở trên và với các số cụm k khác nhau. Khoảng cách giữa hai đường cong gần như không đổi, tuy nhiên sự khác biệt này sẽ đáng kể hơn khi k tăng lên, vì inertia sẽ nhỏ đi. Đồ thị bên phải cho thấy K-Điểm trung bình Mini-batch nhanh hơn nhiều so với K-Điểm trung bình thông thường, và sự vượt trội này tăng theo k.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

[4] Charles Elkan, "Using the Triangle Inequality to Accelerate k-Means," *Proceedings of the 20th International Conference on Machine Learning* (2003): 147–153.

[5] Bất đẳng thức tam giác là $AC \leq AB + BC$ trong đó A, B và C là ba điểm và AB, AC và BC là khoảng cách giữa các điểm này.

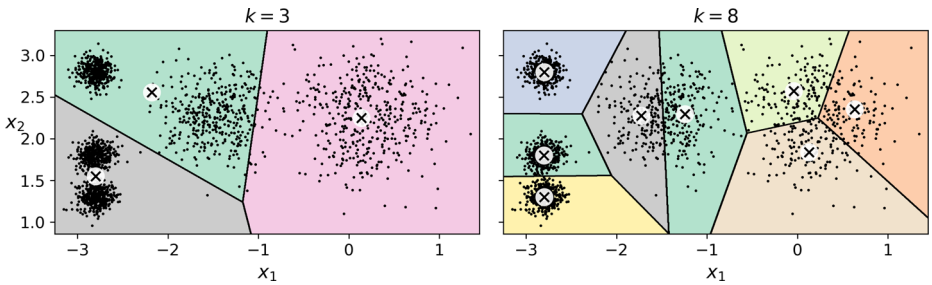
[6] David Sculley, "Web-Scale K-Means Clustering," *Proceedings of the 19th International Conference on World Wide Web* (2010): 1177–1178.



Hình 9.6. K-Điểm trung bình Mini-batch có inertia cao hơn (trái) và nhanh hơn (phải) K-Điểm trung bình thông thường, đặc biệt là khi k tăng

Tìm số cụm tối ưu

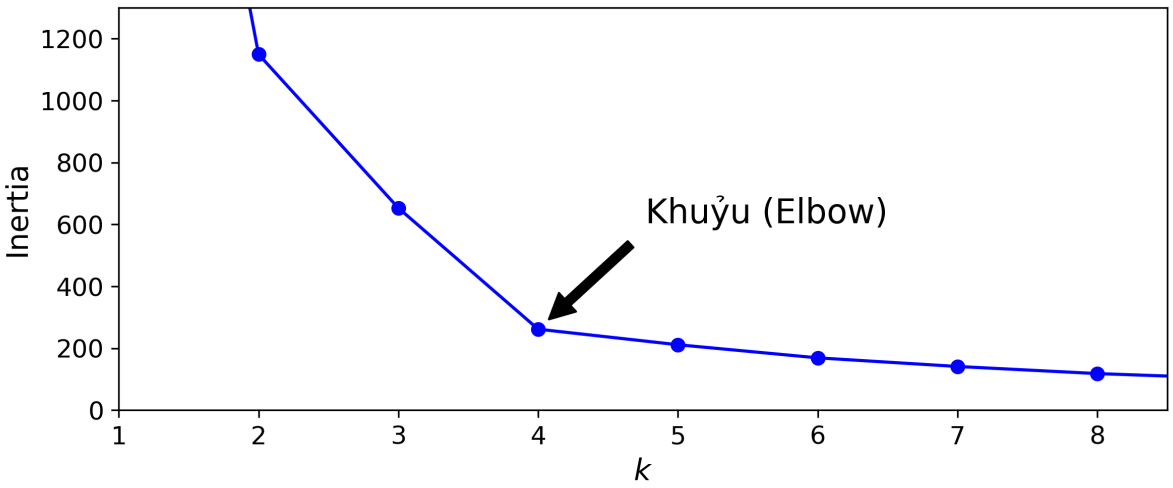
Cho đến giờ, ta mới chỉ đặt số cụm k là 5 vì nếu xem xét dữ liệu thì rõ ràng đây là con số chính xác. Nhưng thường thì việc lựa chọn giá trị chính xác cho k sẽ không dễ dàng như vậy, và kết quả có thể khá tệ nếu ta đặt sai giá trị. Có thể thấy trong Hình 9.7, việc đặt k bằng 3 hoặc 8 sẽ dẫn đến những mô hình khá tệ.



Hình 9.7. Lựa chọn số lượng cụm chưa tốt: các cụm phân tách bị gộp lại lẫn lộn khi k quá nhỏ (trái), và một số cụm bị chia thành nhiều phần khi k quá lớn (phải)

Bạn có thể đang nghĩ rằng, vậy thì chỉ việc chọn mô hình có inertia thấp nhất. Rất tiếc, việc này không đơn giản như vậy. Inertia với $k = 3$ là 653.2, cao hơn rất nhiều khi $k = 5$ (211.6). Nhưng với $k = 8$, inertia chỉ là 119.1. Inertia không phải là phép đo chất lượng tốt để chọn k vì nó sẽ giảm khi k tăng. Thật vậy, càng có nhiều cụm, mỗi mẫu sẽ càng gần với tâm gần nhất của nó, vì vậy inertia càng thấp. Hãy vẽ đồ thị inertia dưới dạng hàm số của k (tham khảo Hình 9.8).

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



*Hình 9.8. Khi biểu diễn inertia như một hàm số của số cụm k, đường cong thường chứa một điểm uốn được gọi là **khuỷu***

Có thể thấy, inertia giảm rất nhanh khi ta tăng k đến 4, nhưng sau đó nó giảm chậm hơn nhiều khi k tiếp tục tăng. Đường cong này giống như hình dạng của cánh tay, và có một "khuỷu tay" tại $k = 4$. Vì thế, nếu ta không có thêm thông tin, 4 có thể là một lựa chọn tốt: bất cứ giá trị nào thấp hơn 4 sẽ cho kết quả kém, trong khi bất cứ giá trị nào cao hơn 4 sẽ không đem lại nhiều lợi ích, vì rất có thể các cụm hoàn chỉnh sẽ bị tách làm đôi mà không có lý do chính đáng.

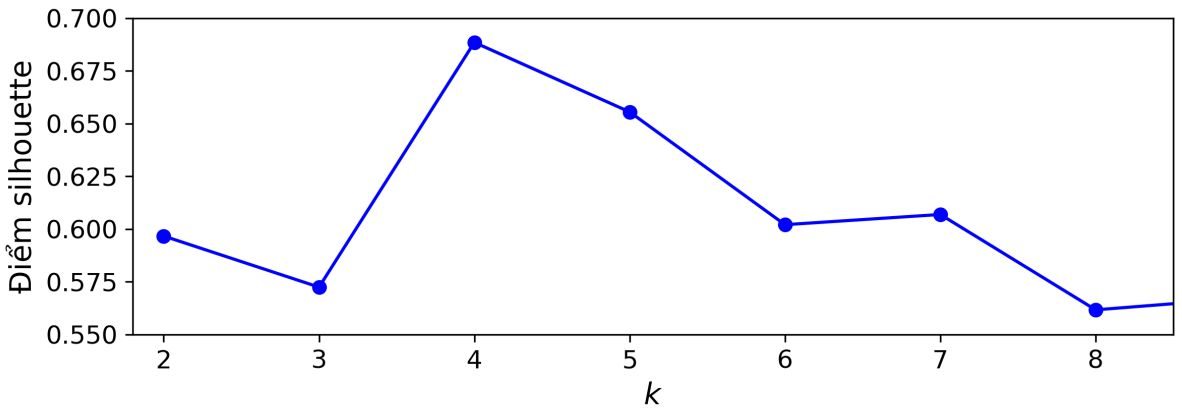
Kỹ thuật chọn số cụm tốt nhất này khá thô. Một phương pháp chính xác hơn (nhưng có chi phí tính toán cao hơn) đó là sử dụng điểm số silhouette. Điểm số này là giá trị trung bình của *hệ số silhouette* trên toàn bộ mẫu. Hệ số silhouette của một mẫu bằng $\frac{b-a}{max(a,b)}$, trong đó a là khoảng cách trung bình từ mẫu đó đến các mẫu khác trong cùng cụm (khoảng cách nội cụm trung bình) và b là khoảng cách trung bình tới cụm gần nhất (khoảng cách trung bình từ mẫu đó đến các mẫu của cụm gần thứ nhì – cụm khiến b đạt giá trị nhỏ nhất mà không phải là cụm chứa mẫu đó). Hệ số silhouette có thể thay đổi trong khoảng -1 đến +1. Một hệ số gần +1 nghĩa là mẫu đó nằm trong cụm riêng của nó và xa các cụm khác. Hệ số gần 0 nghĩa là mẫu đó gần với ranh giới cụm. Cuối cùng, hệ số gần -1 nghĩa là mẫu đó có thể bị gán sai cụm.

Để tính điểm số silhouette, ta có thể sử dụng hàm `silhouette_score()` trong Scikit-Learn, truyền vào tất cả các mẫu trong tập dữ liệu và nhãn cụm của chúng:

```
>>> from sklearn.metrics import silhouette_score
>>> silhouette_score(X, kmeans.labels_)
0.655517642572828
```

Hãy so sánh điểm số silhouette với các số lượng cụm khác nhau (tham khảo Hình 9.9)

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

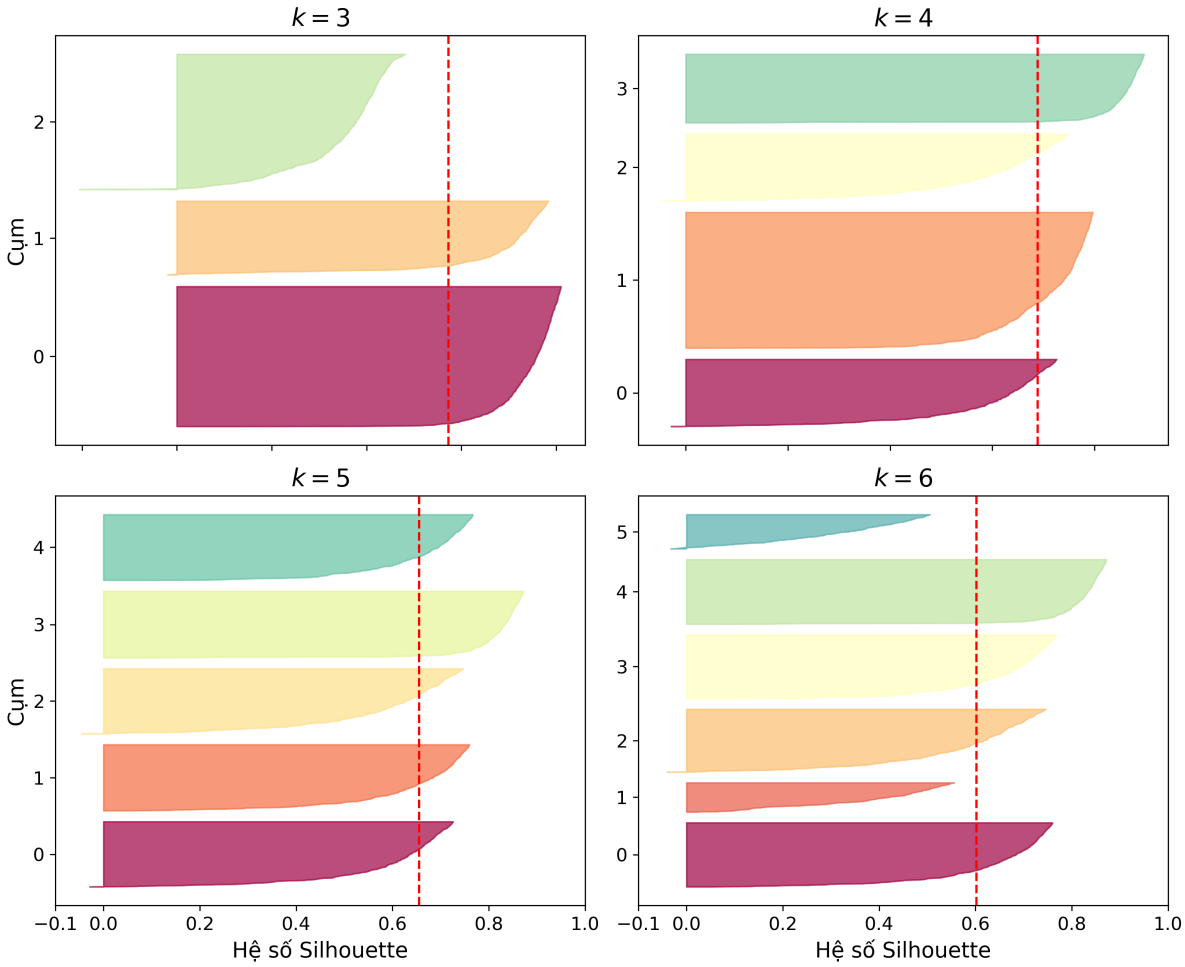


Hình 9.9. Lựa chọn số cụm k dựa trên điểm số silhouette

Có thể thấy, đồ thị này giàu thông tin hơn nhiều so với đồ thị trước: ngoài việc xác nhận rằng $k = 4$ là một lựa chọn rất tốt, đồ thị này cho thấy $k = 5$ cũng cho kết quả tốt, tốt hơn nhiều so với $k = 6$ và 7. Thông tin này không được thể hiện khi so sánh giá trị inertia.

Ta có thể thu được một biểu diễn trực quan giàu thông tin hơn nữa nếu ta biểu diễn hệ số silhouette của từng mẫu, sắp xếp theo cụm mà chúng được gán và giá trị của hệ số đó. Đồ thị này được gọi là *đồ thị silhouette* (tham khảo Hình 9.10) Mỗi đồ thị có hình dạng lưỡi dao tương ứng với một cụm. Chiều cao của lưỡi dao biểu diễn số mẫu trong cụm, trong khi chiều rộng biểu diễn hệ số silhouette của các mẫu (càng rộng càng tốt). Đường nét đứt biểu diễn hệ số silhouette trung bình.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



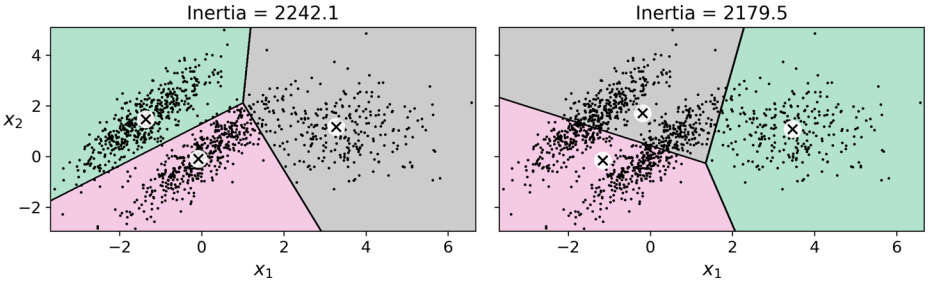
Hình 9.10. Phân tích đồ thị silhouette với các giá trị k *khác nhau*

Các đường nét đứt thẳng đứng biểu diễn điểm số silhouette tương ứng với từng số lượng cụm. Khi hầu hết các mẫu trong một cụm có hệ số thấp hơn điểm số này (tức phần lớn các mẫu cách xa đường nét đứt về phía trái), thì cụm đó khá tệ vì các mẫu của nó nằm quá gần với các cụm khác. Có thể thấy với $k = 3$ và $k = 6$, ta thu được các cụm không tốt. Ngược lại, với $k = 4$ hoặc $k = 5$, các cụm trông khá tốt: hầu hết các mẫu kéo dài xuyên qua đường nét đứt về phía bên phải và gần với 1.0. Khi $k = 4$, cụm số 1 (thứ ba từ trên xuống) khá lớn. Khi $k = 5$, tất cả các cụm có kích thước tương tự nhau. Do đó, mặc dù $k = 4$ có điểm silhouette tổng thể cao hơn $k = 5$, ta vẫn nên sử dụng $k = 5$ để thu được các cụm với kích thước giống nhau.

Hạn chế của K-Điểm trung bình

Mặc dù có nhiều ưu điểm về tốc độ và khả năng mở rộng quy mô, thuật toán K-Điểm trung bình không thật sự hoàn hảo. Như đã đề cập trước đó, chúng ta cần chạy thuật toán nhiều lần để tránh các nghiệm không tối ưu. Ta cần chỉ định trước số lượng cụm, và điều này có thể khá phức tạp. Thêm vào đó, K-Điểm trung bình không hoạt động tốt nếu các cụm có kích thước khác nhau, có mật độ khác nhau, hoặc có dạng phi cầu. Ví dụ, Hình 9.11 biểu diễn cách thuật toán này phân cụm một tập dữ liệu với ba cụm elip có kích thước, mật độ, và hướng khác nhau.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 9.11. K-Điểm trung bình không thể phân cụm ba khối elip một cách phù hợp

Có thể thấy rằng cả hai nghiệm trên đều không tốt. Nghiệm bên trái tốt hơn, tuy vậy nó vẫn gán nhầm 25% cụm ở giữa cho cụm bên phải. Nghiệm ở đồ thị bên phải thì quá tệ, mặc dù điểm inertia của nó thấp hơn. Do đó, tùy thuộc vào dữ liệu, các thuật toán phân cụm khác có thể hoạt động tốt hơn. Đối với loại cụm elip thì mô hình hỗn hợp Gauss hoạt động khá hiệu quả.

Mẹo

Việc co giãn các đặc trưng đầu vào trước khi chạy thuật toán K-Điểm trung bình rất quan trọng, nếu không các cụm sẽ bị giãn quá mức và thuật toán sẽ hoạt động rất kém. Việc co giãn các đặc trưng không đảm bảo rằng tất cả các cụm sẽ có dạng hình cầu, nhưng điều này sẽ giúp cải thiện kết quả thu được.

Giờ hãy xem xét một vài lợi ích của việc phân cụm. Chúng ta sẽ sử dụng K-Điểm trung bình, nhưng các thuật toán phân cụm khác cũng có thể được sử dụng.

Sử dụng Phân cụm để Phân vùng Ảnh

Phân vùng ảnh (*image segmentation*) là tác vụ phân chia một ảnh thành nhiều vùng. Trong *phân vùng theo nhóm* (*semantic segmentation*), tất cả điểm ảnh nằm trên cùng một loại vật thể được gán cùng một phân vùng. Ví dụ, trong hệ thống thị giác của xe tự lái, tất cả các điểm ảnh là một phần của người đi bộ sẽ được gán cho phân vùng "người đi bộ" (sẽ có một vùng chứa tất cả người đi bộ). Trong *phân vùng thực thể* (*instance segmentation*), tất cả điểm ảnh nằm trên cùng một vật thể được gán vào cùng một vùng. Trong trường hợp này, mỗi người đi bộ sẽ có một vùng khác nhau. Phương pháp tân tiến nhất trong phân vùng theo nhóm hoặc phân vùng thực thể ngày nay là sử dụng các kiến trúc phức tạp dựa trên mạng nơ-ron tích chập (sẽ được đề cập trong Tập 2). Ở đây, ta sẽ thực hiện một tác vụ đơn giản hơn nhiều: *phân vùng màu* (*color segmentation*). Ta chỉ đơn thuần gán các điểm ảnh vào cùng một vùng nếu chúng có màu sắc giống nhau. Việc này có thể là đủ trong một vài ứng dụng. Ví dụ, nếu ta muốn phân tích ảnh vệ tinh để tính tổng diện tích rừng trong một vùng nào đó, phân vùng màu có thể sẽ phù hợp.

Đầu tiên, hãy sử dụng hàm `imread()` của Matplotlib để đọc ảnh (xem ảnh ở góc trên-trái trong Hình 9.12):

```
>>> from matplotlib.image import imread  # or `from imageio import imread`
>>> image = imread(os.path.join("images","unsupervised_learning","ladybug.png"))
>>> image.shape
(533, 800, 3)
```

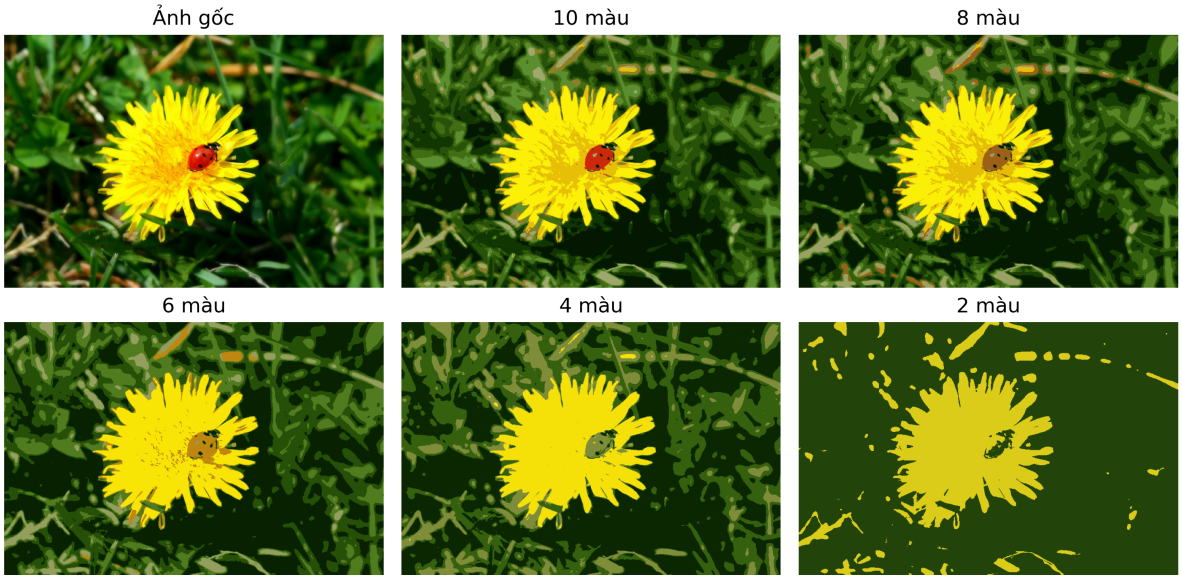
Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ảnh này được biểu diễn dưới dạng mảng ba chiều. Kích thước chiều đầu tiên là chiều cao, chiều thứ hai là chiều rộng, và chiều thứ ba là số kênh màu, trong trường hợp này là đỏ, xanh lá và xanh dương (RGB). Nói cách khác, mỗi điểm ảnh ứng với một vector ba chiều chứa cường độ của màu đỏ, màu xanh lá và màu xanh dương, có giá trị nằm trong khoảng 0.0 và 1.0 (hoặc giữa 0 và 255 nếu sử dụng `imageio.imread()`). Một vài ảnh có thể chứa ít kênh màu hơn, như ảnh thang độ xám (một kênh). Một số ảnh có thể có nhiều kênh màu hơn như ảnh màu có thêm một *kênh alpha* (*alpha channel*) để biểu diễn độ trong suốt, hoặc ảnh vệ tinh thường có nhiều kênh cho nhiều tần số ánh sáng (ví dụ như hồng ngoại). Đoạn mã sau thay đổi lại kích thước mảng để thu được một danh sách màu RGB, sau đó phân cụm những màu này sử dụng K-Điểm trung bình:

```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=8).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```

Ví dụ, phương pháp này có thể xác định một cụm màu cho tất cả các sắc thái của màu xanh lá. Tiếp đó, với mỗi màu cụ thể (xanh lá đậm chẳng hạn), nó sẽ tìm màu trung bình của cụm ứng với màu cụ thể đó. Tất cả sắc thái xanh lá có thể được thay thế bởi một màu xanh lá nhạt (giả sử màu trung bình của cụm xanh lá là xanh lá nhạt). Cuối cùng, danh sách màu này được thay đổi về kích thước ban đầu. Vậy là tất cả các bước đã hoàn thành!

Đầu ra của đoạn mã là ảnh ở góc trên-phải của Hình 9.12. Ta có thể thử nghiệm với các số lượng cụm khác nhau như trong hình. Khi sử dụng ít hơn tám cụm, hãy để ý rằng thuật toán không tách màu đỏ loè loẹt của bọ rùa thành một cụm riêng: nó bị lẫn với màu sắc của môi trường. Điều này là do K-Điểm trung bình có xu hướng tạo ra các cụm có kích thước tương tự nhau. Con bọ rùa có kích thước nhỏ – nhỏ hơn nhiều so với phần còn lại của bức ảnh – nên mặc dù màu sắc của nó rất sặc sỡ, K-Điểm trung bình vẫn không thể dành một cụm riêng cho nó.



Hình 9.12. Phân vùng ảnh sử dụng K-Điểm trung bình với các số cụm khác nhau

Điều này không quá khó khăn, phải không? Bây giờ hãy xem một ứng dụng khác của việc phân cụm: tiền xử lý.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Sử dụng Phân cụm để Tiền xử lý

Phân cụm có thể là một phương pháp hiệu quả để giảm chiều, cụ thể như một bước tiền xử lý trước khi sử dụng một thuật toán học có giám sát. Ta sẽ sử dụng tập dữ liệu digits để minh họa việc sử dụng phân cụm để giảm chiều. Đây là tập dữ liệu tương tự như MNIST, chứa 1.797 ảnh xám với kích thước 8×8 chứa các chữ số từ 0 đến 9. Đầu tiên, ta nạp tập dữ liệu như sau:

```
from sklearn.datasets import load_digits

X_digits, y_digits = load_digits(return_X_y=True)
```

Giờ hãy chia dữ liệu thành tập huấn luyện và tập kiểm tra:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_digits, y_digits)
```

Tiếp theo, ta khớp một mô hình Hồi quy Logistic:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

Ta đánh giá độ chính xác trên tập kiểm tra như sau:

```
>>> log_reg.score(X_test, y_test)
0.9688888888888889
```

Vậy độ chính xác của mô hình cơ sở là 96.9%. Ta sẽ sử dụng K-Điểm trung bình như một bước tiền xử lý để xem liệu có thể cải thiện độ chính xác hay không. Ta sẽ tạo một pipeline để phân cụm tập huấn luyện thành 50 cụm và thay từng ảnh bằng khoảng cách của ảnh tới 50 cụm này, sau đó áp dụng mô hình Hồi quy Logistic:

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("kmeans", KMeans(n_clusters=50)),
    ("log_reg", LogisticRegression()),
])
pipeline.fit(X_train, y_train)
```

Lưu ý

Vì có 10 chữ số khác nhau, có thể bạn sẽ muốn chọn số cụm là 10. Tuy nhiên, mỗi chữ số có thể được viết theo vài cách khác nhau, do đó bạn nên chọn giá trị lớn hơn cho số cụm, ví dụ như 50.

Giờ hãy đánh giá pipeline phân loại này:

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> pipeline.score(X_test, y_test)
0.9777777777777777
```

Ta đã giảm gần 30% tỉ lệ lỗi (từ khoảng 3.1% xuống khoảng 2.2%)!

Nhưng ta đã chọn số cụm k một cách tùy ý, và tất nhiên ta có thể làm tốt hơn. Vì K-Điểm trung bình chỉ là một bước tiền xử lý trong pipeline phân loại, việc tìm giá trị tốt của k đơn giản hơn hẳn so với lúc trước. Ta không cần phải thực hiện phân tích silhouette hay cực tiểu hóa inertia, mà giá trị tốt nhất của k chỉ đơn giản là giá trị đem lại chất lượng phân loại tốt nhất sau khi kiểm định chéo. Ta có thể dùng `GridSearchCV` để tìm số cụm tối ưu:

```
from sklearn.model_selection import GridSearchCV

param_grid = dict(kmeans__n_clusters=range(2, 100))
grid_clf = GridSearchCV(pipeline, param_grid, cv=3, verbose=2)
grid_clf.fit(X_train, y_train)
```

Hãy xem giá trị tốt nhất của k và chất lượng của pipeline thu được:

```
>>> grid_clf.best_params_
{'kmeans__n_clusters': 99}
>>> grid_clf.score(X_test, y_test)
0.9822222222222222
```

Với $k = 99$ cụm, độ chính xác sẽ được cải thiện đáng kể, lên đến 98.22% trên tập kiểm tra. Ta vẫn có thể thử các giá trị k cao hơn, vì ở đây 99 là giá trị lớn nhất trong khoảng tìm kiếm.

Ứng dụng Phân Cụm cho Học Bán Giám sát

Một ứng dụng khác của phân cụm liên quan tới học bán giám sát, khi ta có khá nhiều mẫu không có nhãn và chỉ có rất ít mẫu được gán nhãn. Hãy huấn luyện một mô hình Hồi quy Logistic trên tập con gồm 50 mẫu có nhãn lấy từ tập dữ liệu digits:

```
n_labeled = 50
log_reg = LogisticRegression()
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])
```

Chất lượng của mô hình này trên tập kiểm tra sẽ ra sao?

```
>>> log_reg.score(X_test, y_test)
0.8333333333333334
```

Độ chính xác chỉ đạt 83.3%. Không có gì bất ngờ khi con số này giảm mạnh so với lúc trước, khi ta huấn luyện mô hình trên toàn bộ tập huấn luyện. Hãy xem liệu ta có thể làm tốt hơn không. Đầu tiên, hãy phân tập huấn luyện thành 50 cụm. Tiếp theo, với mỗi cụm, ta tìm ảnh gần tâm cụm nhất. Ta sẽ gọi các ảnh này là *ảnh đại diện* (*representative image*):

```
k = 50
kmeans = KMeans(n_clusters=k)
X_digits_dist = kmeans.fit_transform(X_train)
representative_digit_idx = np.argmin(X_digits_dist, axis=0)
X_representative_digits = X_train[representative_digit_idx]
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Hình 9.13 minh họa 50 bức ảnh đại diện.

Hình 9.13. Năm mươi ảnh chữ số đại diện (mỗi cụm một ảnh)

Hãy cùng nhìn qua từng bức ảnh và gán nhãn thủ công cho chúng:

```
y_representative_digits = np.array([4, 8, 0, 6, 8, 3, ..., 7, 6, 2, 3, 1, 1])
```

Giờ ta có tập dữ liệu với chỉ 50 mẫu có nhãn, nhưng các mẫu này không hề ngẫu nhiên mà là các ảnh đại diện của các cụm tương ứng. Hãy xem chất lượng phân loại có cải thiện hơn không:

```
>>> log_reg = LogisticRegression()
>>> log_reg.fit(X_representative_digits, y_representative_digits)
>>> log_reg.score(X_test, y_test)
0.9222222222222223
```

Độ chính xác đã tăng từ 83.3% lên đến 92.2%, mặc dù ta vẫn đang chỉ huấn luyện mô hình trên 50 mẫu. Việc gán nhãn thường tẻ nhạt và tốn kém, đặc biệt là khi cần các chuyên gia phải gán nhãn thủ công. Do đó, sẽ tốt hơn nếu ta gán nhãn các mẫu đại diện thay vì các mẫu ngẫu nhiên.

Nhưng có lẽ ta có thể tiến thêm một bước: nếu ta gán nhãn của mẫu đại diện cho tất cả các mẫu còn lại trong cụm thì sao? Cách này được gọi là *lan truyền nhãn* (*label propagation*):

```
y_train_propagated = np.empty(len(X_train), dtype=np.int32)
for i in range(k):
    y_train_propagated[kmeans.labels_==i] = y_representative_digits[i]
```

Giờ hãy huấn luyện lại mô hình và kiểm tra chất lượng đạt được:

```
>>> log_reg = LogisticRegression()
>>> log_reg.fit(X_train, y_train_propagated)
>>> log_reg.score(X_test, y_test)
0.9333333333333333
```

Độ chính xác đã tăng, nhưng không quá đáng kể. Vấn đề là do ta đã lan truyền nhãn của mẫu đại diện tới tất cả các mẫu trong cùng một cụm, bao gồm cả các mẫu nằm gần ranh giới giữa các cụm và dễ bị gán nhãn sai. Hãy thử lan truyền nhãn tới chỉ 20% số mẫu nằm gần tâm cụm nhất:

```
percentile_closest = 20
X_cluster_dist = X_digits_dist[np.arange(len(X_train)), kmeans.labels_]
for i in range(k):
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
    in_cluster = (kmeans.labels_ == i)
    cluster_dist = X_cluster_dist[in_cluster]
    cutoff_distance = np.percentile(cluster_dist, percentile_closest)
    above_cutoff = (X_cluster_dist > cutoff_distance)
    X_cluster_dist[in_cluster & above_cutoff] = -1

partially_propagated = (X_cluster_dist != -1)
X_train_partially_propagated = X_train[partially_propagated]
y_train_partially_propagated = y_train_propagated[partially_propagated]
```

Giờ hãy huấn luyện lại mô hình trên tập dữ liệu được lan truyền một phần:

```
>>> log_reg = LogisticRegression()
>>> log_reg.fit(X_train_partially_propagated, y_train_partially_propagated)
>>> log_reg.score(X_test, y_test)
0.94
```

Rất tốt! Chỉ với 50 mẫu có nhãn (trung bình 5 mẫu một lớp), ta có được độ chính xác 94.0%, khá gần với chất lượng của Hồi quy Logistic trên tập dữ liệu chữ số được gán nhãn hoàn toàn (có độ chính xác là 96.9%). Ta thu được chất lượng mô hình tốt như vậy là do kết quả của việc lan truyền nhãn khá tốt. Độ chính xác của nhãn lên tới gần 99%, thể hiện qua đoạn mã dưới đây:

```
>>> np.mean(y_train_partially_propagated == y_train[partially_propagated])
0.9896907216494846
```

Học Chủ động

Để tiếp tục cải thiện chất lượng của mô hình và tập huấn luyện, trong bước kế tiếp ta có thể thực hiện *học chủ động* (*active learning*) một vài lần, trong đó một chuyên gia gán nhãn tương tác với thuật toán và cung cấp nhãn cho các mẫu nhất định khi được yêu cầu. Có rất nhiều chiến lược khác nhau cho việc học chủ động, nhưng một trong những chiến lược phổ biến nhất đó là *lấy mẫu bất định* (*uncertainty sampling*). Chiến lược này hoạt động như sau:

1. Mô hình được huấn luyện trên các mẫu huấn luyện đã được gán nhãn cho đến giờ, và được dùng để đưa ra dự đoán cho tất cả các mẫu không có nhãn.
2. Các mẫu mà mô hình không chắc chắn nhất (tức xác suất ước lượng nhỏ nhất) được đưa cho chuyên gia để gán nhãn.
3. Lặp lại quá trình này cho tới khi mức cải thiện chất lượng không đáng công sức bỏ ra để gán nhãn.

Các chiến lược khác bao gồm gán nhãn các mẫu khiến cho mô hình thay đổi nhiều nhất hoặc lỗi kiểm định giảm mạnh nhất, hay các mẫu khiến cho các mô hình khác nhau bất đồng khi đưa ra dự đoán (ví dụ như SVM hay Rừng Ngẫu nhiên).

Trước khi chuyển sang mô hình hỗn hợp Gauss, hãy cùng xem qua DBSCAN, một thuật toán phân cụm phổ biến với hướng tiếp cận rất khác, dựa trên việc ước lượng mật độ cục bộ. Phương pháp này cho phép thuật toán nhận biết các cụm có hình dạng bất kỳ.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

DBSCAN

Thuật toán này định nghĩa cụm là các vùng liên tục có mật độ cao. DBSCAN hoạt động như sau:

- Với mỗi mẫu, thuật toán đếm xem có bao nhiêu mẫu nằm cách mẫu này một khoảng ε (epsilon) nhỏ. Vùng này được gọi là *vùng lân cận* ε (*ε-neighborhood*) của mẫu đó.
- Nếu một mẫu có ít nhất `min_samples` mẫu trong vùng lân cận ε (bao gồm cả chính nó), mẫu này sẽ được xem là một *mẫu chủ chốt* (*core instance*). Nói cách khác, đây là các mẫu nằm trong vùng dày đặc.
- Mọi mẫu nằm trong vùng lân cận của một mẫu chủ chốt sẽ thuộc cùng một cụm. Vùng lân cận này có thể chứa các mẫu chủ chốt khác, do đó một chuỗi các mẫu chủ chốt lân cận sẽ tạo thành một cụm duy nhất.
- Các mẫu không phải là mẫu chủ chốt và cũng không có mẫu chủ chốt nằm trong vùng lân cận sẽ được xem là một điểm bất thường.

Thuật toán này sẽ hoạt động tốt khi tất cả các cụm đều có mật độ đủ cao và được phân tách rõ ràng bởi các vùng có mật độ thấp. Vẫn như kỳ vọng, lớp `DBSCAN` trong Scikit-Learn khá dễ sử dụng. Hãy thử dùng lớp này trên tập dữ liệu moons được giới thiệu trong Chương 5:

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=1000, noise=0.05)
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
```

Nhãn của tất cả các mẫu nằm trong thuộc tính `labels_`:

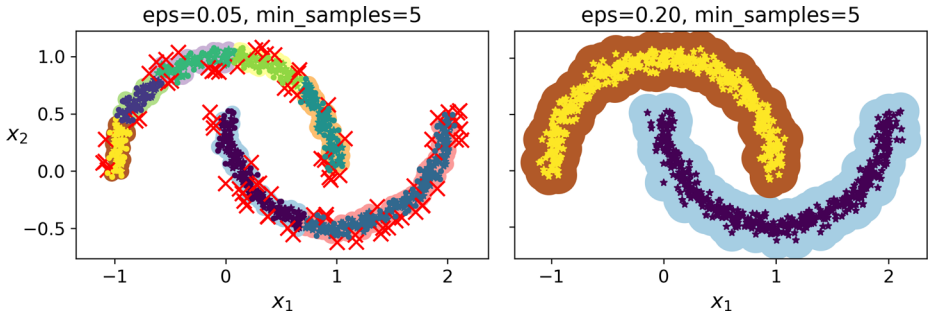
```
>>> dbscan.labels_
array([ 0,  2, -1, -1,  1,  0,  0,  0, ...,  3,  2,  3,  3,  4,  2,  6,  3])
```

Lưu ý rằng một số mẫu có chỉ số cụm bằng -1, và điều này có nghĩa là thuật toán cho rằng đó là các điểm bất thường. Chỉ số của các mẫu chủ chốt được lưu trong thuộc tính `core_sample_indices_` và các mẫu chủ chốt được lưu trong thuộc tính `components_`:

```
>>> len(dbscan.core_sample_indices_)
808
>>> dbscan.core_sample_indices_
array([ 0,  4,  5,  6,  7,  8, 10, 11, ..., 992, 993, 995, 997, 998, 999])
>>> dbscan.components_
array([[-0.02137124,  0.40618608],
       [-0.84192557,  0.53058695],
                  ...
       [-0.94355873,  0.3278936 ],
       [ 0.79419406,  0.60777171]])
```

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Các cụm được biểu diễn trong biểu đồ bên trái của Hình 9.14. Như ta có thể thấy, thuật toán phát hiện khá nhiều điểm bất thường bên cạnh bảy cụm chính. Một kết quả thật đáng thất vọng! May mắn thay, nếu ta mở rộng vùng lân cận của mỗi mẫu bằng cách tăng `eps` lên 0.2, ta sẽ có được kết quả hoàn hảo ở biểu đồ bên phải. Hãy tiếp tục tìm hiểu thêm về mô hình này.



Hình 9.14. Phân cụm DBSCAN với hai bán kính lân cận khác nhau

Bất ngờ là lớp DBSCAN không có phương thức `predict()`, mặc dù nó có phương thức `fit_predict()`. Nói cách khác, lớp DBSCAN không thể dự đoán cụm của một mẫu mới. Quyết định về mặt lập trình này được thực hiện vì các thuật toán phân loại khác nhau sẽ phù hợp cho các tác vụ khác nhau, nên tác giả của thư viện đã quyết định giao quyền lựa chọn thuật toán phân loại cho người dùng. Hơn nữa, việc lựa chọn này cũng không hề khó. Ví dụ, hãy thử huấn luyện một `KNeighborsClassifier`:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
```

Bây giờ, với mỗi mẫu mới, ta có thể dự đoán cụm mà nó có khả năng thuộc về nhất và thậm chí ước lượng xác suất mà mẫu thuộc về mỗi cụm:

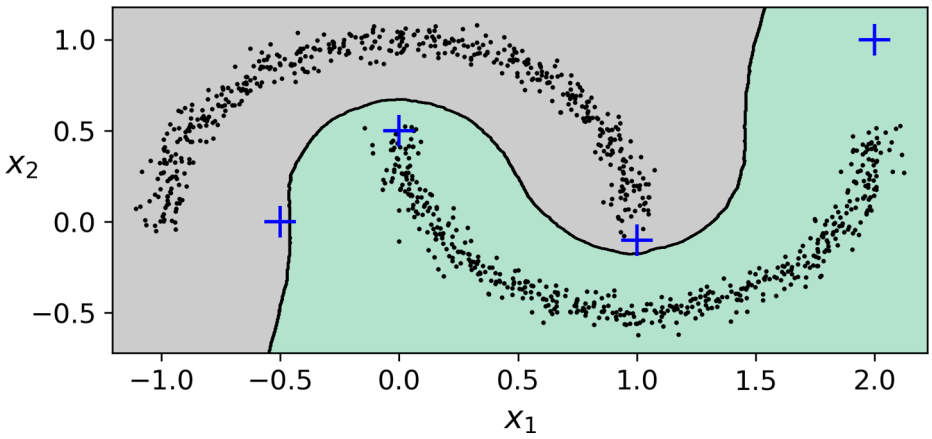
```
>>> X_new = np.array([[-0.5, 0], [0, 0.5], [1, -0.1], [2, 1]])
>>> knn.predict(X_new)
array([1, 0, 1, 0])
>>> knn.predict_proba(X_new)
array([[0.18, 0.82],
       [1.  , 0.  ],
       [0.12, 0.88],
       [1.  , 0.  ]])
```

Lưu ý rằng ta chỉ huấn luyện bộ phân loại dựa trên các mẫu chủ chốt, nhưng ta cũng có thể huấn luyện trên toàn bộ tập hoặc toàn bộ trừ các mẫu bất thường. Lựa chọn này phụ thuộc vào tác vụ cuối cùng.

Ranh giới quyết định được minh họa trong Hình 9.15 (các hình chữ thập biểu diễn bốn mẫu trong `X_new`). Lưu ý rằng vì trong tập huấn luyện không có điểm bất thường, bộ phân loại sẽ luôn chọn một cụm, kể cả khi cụm đó thực chất ở rất xa. Giải pháp dễ thấy nhất cho vấn đề này là sử dụng một mức giới hạn khoảng cách. Khi đó, hai mẫu cách xa cả hai cụm sẽ được phân loại là các điểm bất thường. Để làm điều này, ta có thể sử dụng phương thức `kneighbors()` của `KNeighborsClassifier`. Với một tập các mẫu, phương thức này trả về khoảng cách và chỉ số của k điểm lân cận trong tập huấn luyện (hai ma trận, mỗi ma trận có k cột):

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

```
>>> y_dist, y_pred_idx = knn.kneighbors(X_new, n_neighbors=1)
>>> y_pred = dbscan.labels_[dbscan.core_sample_indices_][y_pred_idx]
>>> y_pred[y_dist > 0.2] = -1
>>> y_pred.ravel()
array([-1,  0,  1, -1])
```



Hình 9.15. Ranh giới quyết định giữa hai cụm

Như vậy, DBSCAN là một thuật toán đơn giản nhưng mạnh mẽ, và có khả năng phát hiện các cụm với bất kỳ hình dạng nào. Thuật toán hoạt động tốt ngay cả khi xuất hiện các điểm ngoại lai, và chỉ có hai siêu tham số (`eps` và `min_samples`). Tuy nhiên, nếu mật độ thay đổi rõ rệt giữa các cụm, việc xác định đúng tất cả các cụm có thể sẽ trở nên bất khả thi. Độ phức tạp tính toán của thuật toán này là $O(m \log m)$, giúp thời gian chạy gần như tuyến tính so với kích thước tập dữ liệu, nhưng phiên bản của Scikit-Learn có thể cần tới $O(m^2)$ bộ nhớ trong trường hợp `eps` lớn.

Mẹo

Bạn có thể cũng muốn thử *DBSCAN Phân cấp* (*Hierarchical DBSCAN* - HDBSCAN) có trong dự án scikit-learn-contrib.

Các Thuật toán Phân cụm khác

Scikit-Learn còn có một vài thuật toán phân cụm nữa mà ta có thể tìm hiểu. Chúng tôi không thể đi vào chi tiết cho tất cả các thuật toán, nhưng dưới đây là một danh sách sơ lược:

Tích cụm (agglomerative clustering)
Một hệ thống các cụm có cấp bậc được xây dựng từ dưới lên. Hãy hình dung nhiều bong bóng nhỏ trên mặt nước dần dần kết hợp lại với nhau đến khi hình thành một nhóm lớn. Tương tự, tại mỗi vòng lặp, thuật toán tích cụm nối các cặp cụm gần nhất lại với nhau (bắt đầu từ các mẫu riêng lẻ). Nếu ta vẽ một cái cây với mỗi nhánh biểu diễn một cặp các cụm được nối lại, ta sẽ có được một cây nhị phân của các cụm, trong đó mỗi lá là một mẫu. Phương pháp này mở rộng rất tốt với số lượng mẫu hoặc cụm lớn. Thuật toán có thể xác định được các cụm với hình dạng bất kỳ, tạo ra một cây cụm rất linh hoạt và chứa nhiều thông tin thay vì ép ta phải chọn một số lượng cụm nhất định, và có thể được dùng với bất kỳ phép đo khoảng cách nào. Nó cũng mở rộng tốt trong trường hợp số lượng mẫu lớn nếu ta cung cấp ma trận

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

kết nối – một ma trận thưa $m \times m$ để chỉ định các cặp mẫu lân cận (giống như ma trận mà `sklearn.neighbors.kneighbors_graph()` trả về). Nếu không có ma trận này, thuật toán sẽ mở rộng rất kém với các tập dữ liệu lớn.

BIRCH

Thuật toán BIRCH (*Balanced Iterative Reducing and Clustering using Hierarchies*) được thiết kế dành riêng cho các tập dữ liệu rất lớn. Thuật toán này có thể còn nhanh hơn K-Điểm trung bình theo batch nhưng vẫn đưa ra kết quả tương tự, miễn là số lượng các đặc trưng không quá lớn (<20). Trong quá trình huấn luyện, thuật toán sẽ xây dựng một cấu trúc cây chứa vừa đủ thông tin để nhanh chóng gán cụm cho các mẫu mà không phải lưu trữ toàn bộ dữ liệu trong cây. Phương pháp này cho phép thuật toán sử dụng một lượng bộ nhớ hạn chế khi xử lý một tập dữ liệu khổng lồ.

Dịch-Trung bình (Mean-Shift)

Thuật toán này khởi đầu bằng cách đặt một vòng tròn xung quanh các mẫu, với mỗi vòng tròn tính giá trị trung bình của các mẫu bên trong nó, sau đó dịch chuyển vòng tròn sao cho tâm nằm tại giá trị trung bình. Tiếp đến, nó lặp lại bước dịch chuyển này cho đến khi không còn vòng tròn nào di chuyển nữa (tức là cho đến khi tâm các vòng tròn đều nằm tại điểm trung bình của các mẫu bên trong). Thuật toán này dịch chuyển các vòng tròn về hướng có mật độ mẫu cao, cho đến khi tất cả các vòng tròn đều tìm được một điểm mà mật độ cục bộ đạt tối đa. Cuối cùng, các mẫu có vòng tròn nằm tại cùng một vị trí (hoặc đủ gần) được gán chung một cụm. Dịch-Trung bình có một vài điểm giống với thuật toán DBSCAN, như việc có thể tìm số lượng cụm bất kỳ và với hình dạng bất kỳ, có rất ít siêu tham số (chỉ một siêu tham số là bán kính vòng tròn, được gọi là *băng thông*) và cùng dựa trên việc ước lượng mật độ cục bộ. Nhưng khác với DBSCAN, Dịch-Trung bình có xu hướng chia nhỏ các cụm nếu mật độ bên trong cụm biến thiên. Thật không may, thuật toán này có độ phức tạp tính toán là $O(m^2)$, vậy nên nó không phù hợp cho các tập dữ liệu lớn.

Lan truyền Ái lực (affinity propagation)

Thuật toán này hoạt động dựa trên cơ chế biểu quyết, trong đó mỗi mẫu sẽ bầu ra các mẫu giống nó làm các đại diện. Một khi thuật toán hội tụ, mỗi đại diện và các mẫu bầu cho nó sẽ hình thành một cụm. Thuật toán này có thể phát hiện số lượng cụm bất kỳ với các kích thước khác nhau. Tuy nhiên, thuật toán này có độ phức tạp tính toán $O(m^2)$, nên nó cũng không phù hợp cho các tập dữ liệu lớn.

Phân cụm phổ (spectral clustering)

Thuật toán này nhận một ma trận độ tương tự giữa các mẫu và từ đó tạo một embedding thấp chiều hơn (tức thực hiện giảm chiều cho ma trận tương tự), rồi sử dụng một thuật toán phân cụm khác trong không gian đã giảm chiều này (Scikit-Learn mặc định sử dụng K-Điểm trung bình). Phân cụm Phổ có thể xác định được các cấu trúc cụm phức tạp và có thể được sử dụng để chia nhỏ đồ thị (ví dụ như phát hiện các cụm bạn bè trong một mạng xã hội). Thuật toán này không mở rộng quá tốt với số lượng mẫu lớn và cũng không hoạt động tốt khi các cụm có kích thước khác nhau.

Giờ thì ta hãy cùng tìm hiểu về mô hình hỗn hợp Gauss. Mô hình này thường được dùng cho các tác vụ ước lượng mật độ, phân cụm và phát hiện bất thường.

Hỗn hợp Gauss

Một *mô hình hỗn hợp Gauss* (*Gaussian mixture model* - GMM) là một mô hình xác suất với giả định là các mẫu được sinh ra từ hỗn hợp các phân phối Gauss với các tham số chưa xác định.

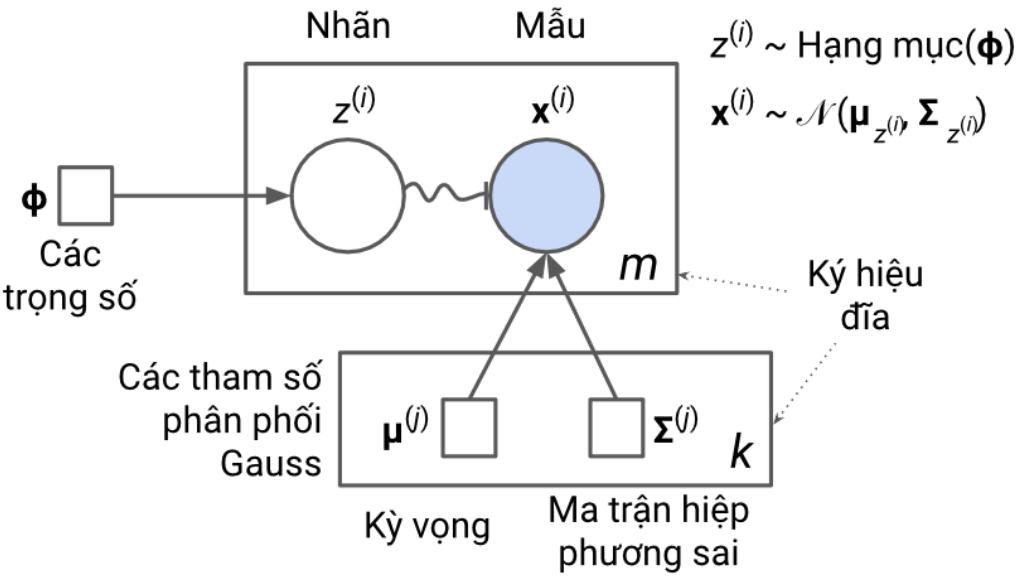
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tất cả các mẫu được sinh từ cùng một phân phối Gauss thường tạo thành một cụm có hình dạng giống một ellipsoid (hình trái xoan). Mỗi cụm có thể là một ellipsoid với hình dạng, kích thước, mật độ và góc khác nhau, như trong Hình 9.11. Khi quan sát một mẫu, ta biết rằng nó được sinh từ một trong những phân phối Gauss, nhưng không biết cụ thể đó là phân phối nào cũng như các tham số của phân phối đó.

GMM có nhiều biến thể khác nhau. Trong biến thể đơn giản nhất được lập trình trong lớp `GaussianMixture`, ta cần biết trước số lượng phân phối Gauss k. Giả định rằng tập dữ liệu $\mathbf{X}$ được sinh thông qua quá trình xác suất sau:

- Với từng mẫu, ta chọn ngẫu nhiên một cụm từ k cụm. Xác suất chọn cụm thứ j được định nghĩa bằng trọng số $\phi^{(j)}$[7] của cụm. Chỉ số cụm được chọn cho mẫu thứ i có ký hiệu là $z^{(i)}$.
- Nếu $z^{(i)} = j$, có nghĩa là mẫu thứ i được gán cho cụm thứ j, vị trí $\mathbf{x}^{(i)}$ của mẫu được lấy mẫu ngẫu nhiên từ phân phối Gauss với trung bình $\boldsymbol{\mu}^{(j)}$ và ma trận hiệp phương sai $\boldsymbol{\Sigma}^{(j)}$, ký hiệu là $\mathbf{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)})$.

Quá trình sinh trên có thể được biểu diễn bằng mô hình đồ thị. Hình 9.16 biểu diễn cấu trúc phụ thuộc có điều kiện giữa các biến ngẫu nhiên.



Hình 9.16. Biểu diễn đồ thị của mô hình hỗn hợp Gauss, gồm có các tham số (hình vuông), các biến ngẫu nhiên (hình tròn), và các phụ thuộc có điều kiện (các mũi tên nét liền)

Ta diễn giải biểu đồ như sau:[8]

- Các hình tròn biểu diễn các biến ngẫu nhiên.
- Các hình vuông biểu diễn các giá trị cố định (tức các tham số của mô hình).

[7] Phi (ϕ hay φ) là ký tự thứ 21 trong bảng chữ cái Hy Lạp.
[8] Hầu hết các ký hiệu là các ký hiệu chuẩn, nhưng một vài ký hiệu bổ sung được lấy từ bài viết trên Wikipedia về ký hiệu đĩa.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- Các hình chữ nhật lớn được gọi là các *đĩa* (*plate*), cho biết các thành phần bên trong chúng được lặp lại vài lần.
- Các số ở góc dưới phải của mỗi đĩa biểu diễn số lần lặp lại của các thành phần bên trong. Vậy nên có m biến ngẫu nhiên $z^{(i)}$ (từ $z^{(1)}$ đến $z^{(m)}$) và m biến ngẫu nhiên $\mathbf{x}^{(i)}$. Ta cũng có k trung bình $\boldsymbol{\mu}^{(j)}$ và k ma trận hiệp phương sai $\boldsymbol{\Sigma}^{(j)}$. Cuối cùng, chỉ có một vector trọng số $\boldsymbol{\phi}$ (chứa tất cả các trọng số từ $\phi^{(1)}$ tới $\phi^{(k)}$.
- Mỗi biến $z^{(i)}$ được lấy ngẫu nhiên từ *phân phối hạng mục* với trọng số $\boldsymbol{\phi}$. Mỗi biến $\mathbf{x}^{(i)}$ được lấy từ phân phối chuẩn có trung bình và ma trận hiệp phương sai được định nghĩa bởi cụm $z^{(i)}$ của biến đó.
- Các mũi tên nét liền biểu diễn các phụ thuộc có điều kiện. Ví dụ, phân phối xác suất của biến ngẫu nhiên $z^{(i)}$ phụ thuộc vào vector trọng số $\boldsymbol{\phi}$. Lưu ý rằng việc mũi tên giao với biên của đĩa có nghĩa là điều kiện được áp dụng cho tất các lần lặp của đĩa đó. Ví dụ, vector trọng số $\boldsymbol{\phi}$ là điều kiện của các phân phối xác suất của tất cả các biến ngẫu nhiên từ $\mathbf{x}^{(1)}$ tới $\mathbf{x}^{(m)}$.
- Các mũi tên xoắn từ $z^{(i)}$ tới $\mathbf{x}^{(i)}$ biểu diễn sự thay đổi: tuỳ vào giá trị của $z^{(i)}$, mẫu $\mathbf{x}^{(i)}$ sẽ được lấy từ một phân phối Gauss khác. Ví dụ, nếu $z^{(i)} = j$, thì $\mathbf{x}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}^{(j)}, \boldsymbol{\Sigma}^{(j)})$.
- Các nút được tô màu biểu diễn các giá trị đã biết. Vì thế, trong trường hợp này, chỉ có giá trị của biến ngẫu nhiên $\mathbf{x}^{(i)}$ là đã biết. Chúng được gọi là các *biến đã quan sát* (*observed variable*). Các biến ngẫu nhiên $z^{(i)}$ chưa biết được gọi là các *biến tiềm ẩn* (*latent variable*).

Vậy ta có thể làm gì với mô hình này? Với tập dữ liệu $\mathbf{X}$, ta thường bắt đầu bằng việc ước lượng trọng số $\boldsymbol{\phi}$ và tất cả các tham số phân phối từ $\boldsymbol{\mu}^{(1)}$ tới $\boldsymbol{\mu}^{(k)}$ và từ $\boldsymbol{\Sigma}^{(1)}$ tới $\boldsymbol{\Sigma}^{(k)}$. Ta có thể thực hiện việc này một cách cực kỳ đơn giản với lớp `GaussianMixture` của Scikit-Learn:

```
from sklearn.mixture import GaussianMixture

gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
```

Hãy cùng xem các tham số mà thuật toán đã ước lượng:

```
>>> gm.weights_
array([0.20965228, 0.4000662 , 0.39028152])
>>> gm.means_
array([[ 3.39909717,  1.05933727],
       [-1.40763984,  1.42710194],
       [ 0.05135313,  0.07524095]])
>>> gm.covariances_
array([[[ 1.14807234, -0.03270354],
        [-0.03270354,  0.95496237]],

       [[ 0.63478101,  0.72969804],
        [ 0.72969804,  1.1609872 ]],

       [[ 0.68809572,  0.79608475],
        [ 0.79608475,  1.21234145]]])
```

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tuyệt vời, thuật toán hoạt động khá tốt! Thật vậy, các trọng số đã được sử dụng để sinh tập dữ liệu là 0.2, 0.4, và 0.4. Tương tự, trung bình và ma trận hiệp phương sai cũng rất gần với kết quả tìm được từ thuật toán. Nhưng bằng cách nào mà thuật toán làm được như vậy? Lớp này hoạt động dựa trên thuật toán *Kỳ vọng - Cực đại hóa* (*Expectation-Maximization* – EM). Thuật toán này có nhiều điểm tương đồng với thuật toán K-Điểm trung bình: nó cũng khởi tạo các tham số cụm một cách ngẫu nhiên, sau đó lặp lại hai bước cho đến khi hội tụ. Bước đầu tiên là gán các mẫu cho các cụm (gọi là *bước kỳ vọng*), sau đó cập nhật các cụm (gọi là *bước cực đại hóa*). Nghe rất quen thuộc phải không? Trong bối cảnh phân cụm, ta có thể ví EM như dạng tổng quát của K-Điểm trung bình, nhưng nó không chỉ tìm kiếm các tâm cụm (từ $\boldsymbol{\mu}^{(1)}$ tới $\boldsymbol{\mu}^{(k)}$) mà còn tìm cả kích thước, hình dạng, hướng (từ $\boldsymbol{\Sigma}^{(1)}$ tới $\boldsymbol{\Sigma}^{(k)}$), và cả trọng số tương đối của chúng (từ $\phi^{(1)}$ tới $\phi^{(k)}$). Điểm khác biệt với K-Điểm trung bình là EM sử dụng các phép gán cụm mềm thay vì gán cứng. Tại bước kỳ vọng, với mỗi mẫu, thuật toán ước lượng xác suất mẫu đó thuộc về từng cụm (dựa trên tham số hiện tại của các cụm). Sau đó tại bước cực đại hóa, mỗi cụm được cập nhật bằng cách sử dụng *tất cả* các mẫu trong tập dữ liệu, với trọng số mỗi mẫu là xác suất mà mẫu thuộc về cụm đó. Các xác suất này được gọi là *độ trách nhiệm* (*responsibility*) của cụm đối với mẫu. Trong bước cực đại hóa, kết quả cập nhật của từng cụm chịu ảnh hưởng chủ yếu bởi các mẫu mà nó chịu trách nhiệm nhiều nhất.

Lưu ý

Thật không may, cũng giống như K-Điểm trung bình, EM có thể hội tụ về các nghiệm không tốt, vì vậy ta cần chạy thuật toán nhiều lần và chỉ giữ lại nghiệm tốt nhất. Đấy là lý do tại sao ta gán `n_init` bằng 10 (theo mặc định `n_init` được gán bằng 1).

Ta có thể kiểm tra xem thuật toán này có hội tụ hay không và số vòng lặp đã được thực hiện:

```
>>> gm.converged_
True
>>> gm.n_iter_
3
```

Giờ ta đã có ước lượng về vị trí, kích thước, hình dạng, hướng, cũng như trọng số tương đối của từng cụm. Mô hình giờ có thể dễ dàng gán từng mẫu cho cụm hợp lý nhất (phân cụm cứng) hoặc ước lượng xác suất mà nó thuộc về một cụm nhất định (phân cụm mềm). Ta chỉ cần sử dụng phương thức `predict()` để phân cụm cứng, hoặc `predict_proba()` để phân cụm mềm:

```
>>> gm.predict(X)
array([2, 2, 1, ..., 0, 0, 0])
>>> gm.predict_proba(X)
array([[2.32389467e-02, 6.77397850e-07, 9.76760376e-01],
       [1.64685609e-02, 6.75361303e-04, 9.82856078e-01],
       [2.01535333e-06, 9.99923053e-01, 7.49319577e-05],
       ...,
       [9.99999571e-01, 2.13946075e-26, 4.28788333e-07],
       [1.00000000e+00, 1.46454409e-41, 5.12459171e-16],
       [1.00000000e+00, 8.02006365e-41, 2.27626238e-15]])
```

Mô hình hỗn hợp Gauss là một *mô hình sinh* (*generative model*), nghĩa là ta có thể lấy mẫu mới từ mô hình (chú ý rằng chúng được sắp xếp theo chỉ số cụm):

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

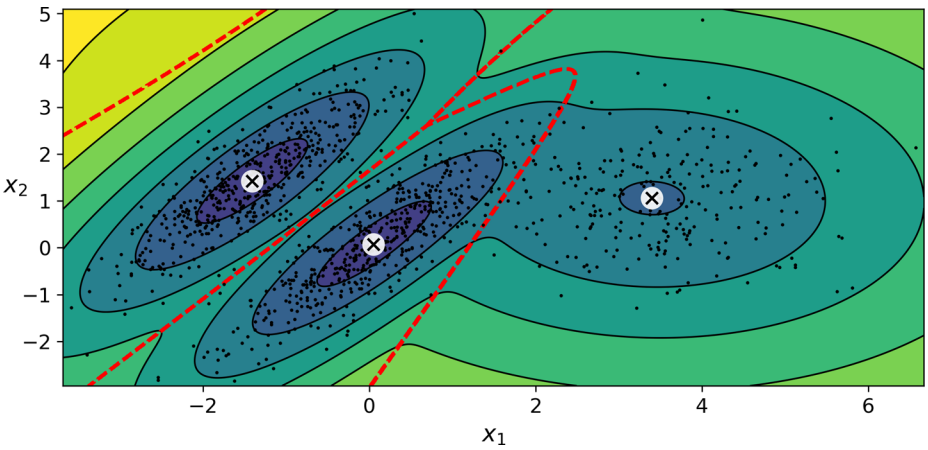
```
>>> X_new, y_new = gm.sample(6)
>>> X_new
array([[ 2.95400315,  2.63680992],
       [-1.16654575,  1.62792705],
       [-1.39477712, -1.48511338],
       [ 0.27221525,  0.690366  ],
       [ 0.54095936,  0.48591934],
       [ 0.38064009, -0.56240465]])

>>> y_new
array([0, 1, 2, 2, 2, 2])
```

Ta cũng có thể ước lượng mật độ của mô hình tại một điểm bất kỳ. Điều này được thực hiện bằng phương thức `score_samples()`: với mỗi mẫu cho trước, phương thức này ước lượng logarit của *hàm mật độ xác suất* (*probability density function* – PDF) tại điểm đó. Điểm số càng cao thì mật độ càng lớn.

```
>>> gm.score_samples(X)
array([-2.60782346, -3.57106041, -3.33003479, ..., -3.51352783,
       -4.39802535, -3.80743859])
```

Nếu tính lũy thừa của những điểm số này, ta sẽ nhận được giá trị PDF tại vị trí của các mẫu đã cho. Các giá trị này không phải là xác suất mà là *mật độ* xác suất: chúng có thể nhận bất kỳ giá trị dương nào, không chỉ trong khoảng giữa 0 và 1. Để ước lượng xác suất một mẫu nằm trong một vùng cụ thể, ta cần lấy tích phân của PDF trên vùng đó (nếu lấy tích phân trên toàn bộ không gian chứa tất cả vị trí mẫu khả dĩ, kết quả sẽ bằng 1).

Hình 9.17 biểu diễn các trung bình của cụm, ranh giới quyết định (nét đứt), và đường bao mật độ của mô hình này.



Hình 9.17. Trung bình cụm, ranh giới quyết định, và đường bao mật độ của một mô hình hỗn hợp Gauss đã được huấn luyện

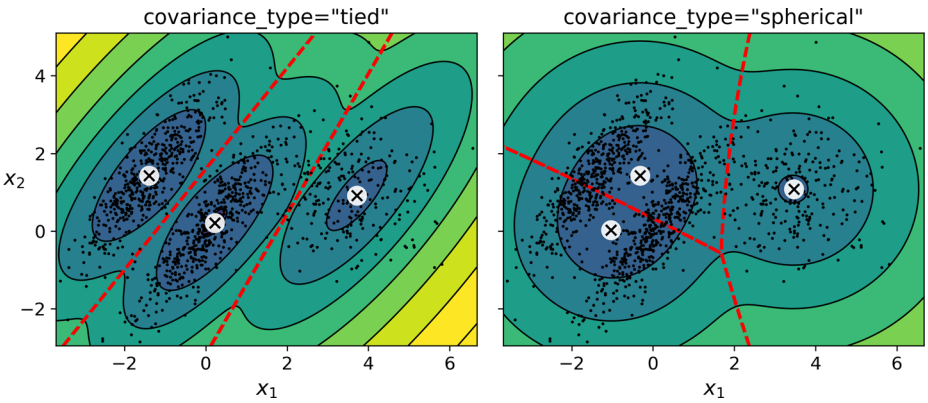
Tuyệt! Rõ ràng thuật toán này đã tìm thấy một nghiệm rất tốt. Tất nhiên, ta đã khiến tác vụ này trở nên dễ dàng khi khởi tạo dữ liệu bằng cách sử dụng các phân phối Gauss hai chiều (không may là dữ liệu thực tế không phải lúc nào cũng có ít chiều và tuân theo phân phối Gauss). Ta cũng đã cung cấp số lượng cụm chính xác cho thuật toán. Khi có nhiều chiều, hoặc nhiều cụm, hoặc ít mẫu, EM có thể gặp khó khăn trong việc hội tụ đến nghiệm tối ưu. Ta có

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

thể cần giảm độ khó của tác vụ bằng cách giới hạn số lượng tham số mà thuật toán phải học. Một cách để làm điều này là giới hạn hình dạng và hướng của các cụm. Điều này có thể đạt được bằng cách áp đặt các ràng buộc trên ma trận hiệp phương sai. Để thực hiện điều đó, hãy đặt siêu tham số `covariance_type` bằng một trong các giá trị sau:

- `"spherical"`: Tất cả các cụm phải có dạng hình cầu, tuy nhiên chúng có thể có đường kính khác nhau (tức phương sai khác nhau).
- `"diag"`: Các cụm có thể có hình dạng elip với bất kỳ kích thước nào, nhưng trục của elip phải song song với trục tọa độ (tức ma trận hiệp phương sai phải là ma trận chéo).
- `"tied"`: Tất cả các cụm phải có cùng hình dạng elip, cùng kích thước và cùng hướng (tức các cụm có cùng ma trận hiệp phương sai).

Theo mặc định, `covariance_type` được đặt là `"full"`, có nghĩa là mỗi cụm có thể có hình dạng, kích thước, và hướng bất kỳ (mỗi cụm có riêng một ma trận phương sai không bị giới hạn). Hình 9.18 biểu diễn các nghiệm tìm được bởi thuật toán EM khi `covariance_type` được gán bằng `"tied"` hoặc `"spherical"`.



Hình 9.18. Hỗn hợp Gauss với cụm `tied` (bên trái) và cụm `spherical` (bên phải)

Ghi chú

Chú ý rằng độ phức tạp tính toán để huấn luyện một mô hình `GaussianMixture` phụ thuộc vào số mẫu m, số chiều n, số cụm k, và số điều kiện ràng buộc của ma trận hiệp phương sai. Nếu `covariance_type` là `"spherical"` hoặc `"diag"`, độ phức tạp tính toán sẽ là $O(kmn)$ giả sử rằng dữ liệu có dạng cụm. Nếu `covariance_type` là `"tied"` hoặc `"full"`, độ phức tạp tính toán sẽ là $O(kmn^2) + O(kn^3)$, nghĩa là nó sẽ không mở rộng tốt với số lượng đặc trưng lớn.

Mô hình hỗn hợp Gauss cũng có thể được sử dụng để phát hiện bất thường. Hãy cùng xem cách thực hiện điều này.

Phát hiện Bất thường Sử dụng Hỗn hợp Gauss

Phát hiện bất thường – anomaly detection (còn được gọi là *phát hiện ngoại lai – outlier detection*) là tác vụ phát hiện các mẫu khác hẳn so với các mẫu thông thường. Những mẫu này được gọi là *bất thường* (*anomaly*) hoặc *ngoại lai* (*outlier*), trong khi các mẫu bình thường được gọi là

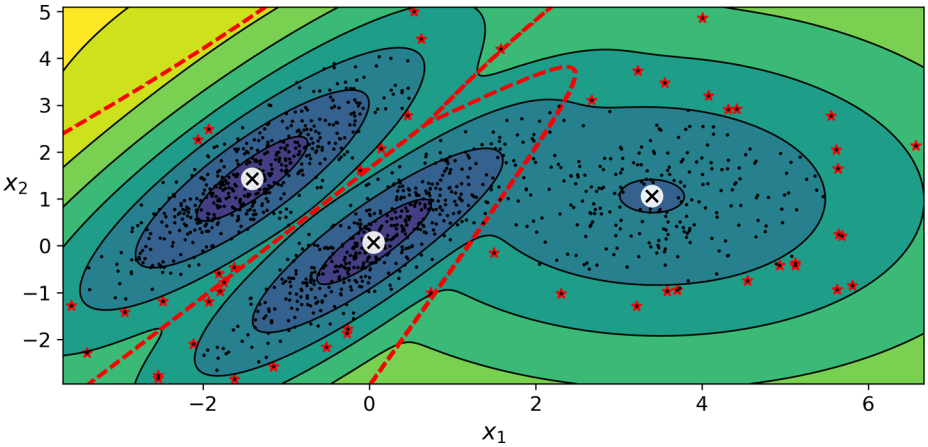
Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

inlier. Phát hiện bất thường rất hữu ích trong nhiều ứng dụng, chẳng hạn như phát hiện gian lận, phát hiện sản phẩm lỗi trong sản xuất, hoặc loại bỏ bất thường khỏi tập dữ liệu trước khi huấn luyện một mô hình khác (có thể cải thiện đáng kể chất lượng của mô hình thu được).

Việc sử dụng mô hình hỗn hợp Gauss để phát hiện bất thường khá đơn giản: bất cứ mẫu nào nằm trong vùng mật độ thấp có thể được xem là một bất thường. Ta cần định nghĩa ngưỡng mật độ mà ta muốn sử dụng. Ví dụ, với một công ty sản xuất muốn phát hiện sản phẩm lỗi thì công ty thường sẽ biết trước tỉ lệ sản phẩm lỗi. Giả sử giá trị này là 4%, ta có thể đặt ngưỡng mật độ sao cho 4% số mẫu nằm trong vùng có mật độ thấp hơn ngưỡng đó. Nếu có quá nhiều dương tính giả (sản phẩm tốt nhưng lại được gán nhãn bị lỗi), ta có thể giảm ngưỡng xuống. Ngược lại, nếu có quá nhiều âm tính giả (sản phẩm lỗi nhưng không bị phát hiện), ta có thể tăng ngưỡng lên. Đây chính là sự đánh đổi giữa precision và recall (tham khảo Chương 3). Dưới đây là cách xác định bất thường với mức ngưỡng là bách phân vị thứ tư của mật độ thấp nhất (xấp xỉ 4% mẫu sẽ được dự đoán là bất thường):

```
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 4)
anomalies = X[densities < density_threshold]
```

Hình 9.19 biểu diễn các bất thường dưới dạng hình ngôi sao.



Hình 9.19. Phát hiện bất thường sử dụng mô hình hỗn hợp Gauss

Một tác vụ liên quan mật thiết khác là *phát hiện tính mới* (*novelty detection*): tác vụ này khác với phát hiện bất thường ở giả định rằng thuật toán đã được huấn luyện trên một tập dữ liệu "sạch" và không chứa mẫu bất thường, trong khi phát hiện bất thường không hề có giả định này. Thật vậy, thường thì phát hiện bất thường được sử dụng để làm sạch dữ liệu.

Mẹo

Các mô hình hỗn hợp Gauss sẽ cố gắng khớp tất cả các điểm dữ liệu bao gồm cả bất thường. Vì vậy nếu ta có quá nhiều bất thường, việc này sẽ làm sai lệch quan điểm của mô hình về "tính bình thường", khiến một vài bất thường có thể được cho là bình thường. Nếu việc này xảy ra, ta có thể khớp trước dữ liệu một lần, sử dụng nó để phát hiện và loại bỏ những mẫu bất thường nhất, sau đó khớp mô hình lại một lần nữa trên tập dữ liệu đã được làm sạch. Một hướng tiếp cận khác là sử dụng các phương pháp ước lượng hiệp phương sai tăng cường (tham khảo lớp `EllipticEnvelope`).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tương tự như K-điểm trung bình, thuật toán `GaussianMixture` đòi hỏi ta phải cung cấp trước số cụm. Vậy thì làm cách nào để tìm được nó?

Chọn Số Cụm

Với K-điểm trung bình, ta có thể sử dụng inertia hoặc silhouette để chọn số cụm phù hợp. Nhưng với mô hình hỗn hợp Gauss, ta không thể sử dụng những phép đo này vì chúng không đáng tin cậy khi cụm không có hình cầu hoặc các cụm có kích thước khác nhau. Thay vào đó, ta có thể tìm mô hình tối thiểu hóa một *tiêu chí thông tin lý thuyết* (*theoretical information criterion*), như *tiêu chí thông tin Bayes* (*Bayesian information criterion* – BIC) hoặc *tiêu chí thông tin Akaike* (*Akaike information criterion* – AIC), được định nghĩa trong Phương trình 9.1.

$$\begin{aligned} BIC &= \log(m)p - 2\log(\widehat{L}) \\ AIC &= 2p - 2\log(\widehat{L}) \end{aligned}$$

Phương trình 9.1. Tiêu chí thông tin Bayesian (BIC) và tiêu chí thông tin Akaike (AIC)

Trong đó:

- m vẫn là số mẫu như thường lệ.
- p là số các tham số được học bởi mô hình.
- $\widehat{L}$ là giá trị cực đại của *hàm hợp lý* (*likelihood function*) của mô hình.

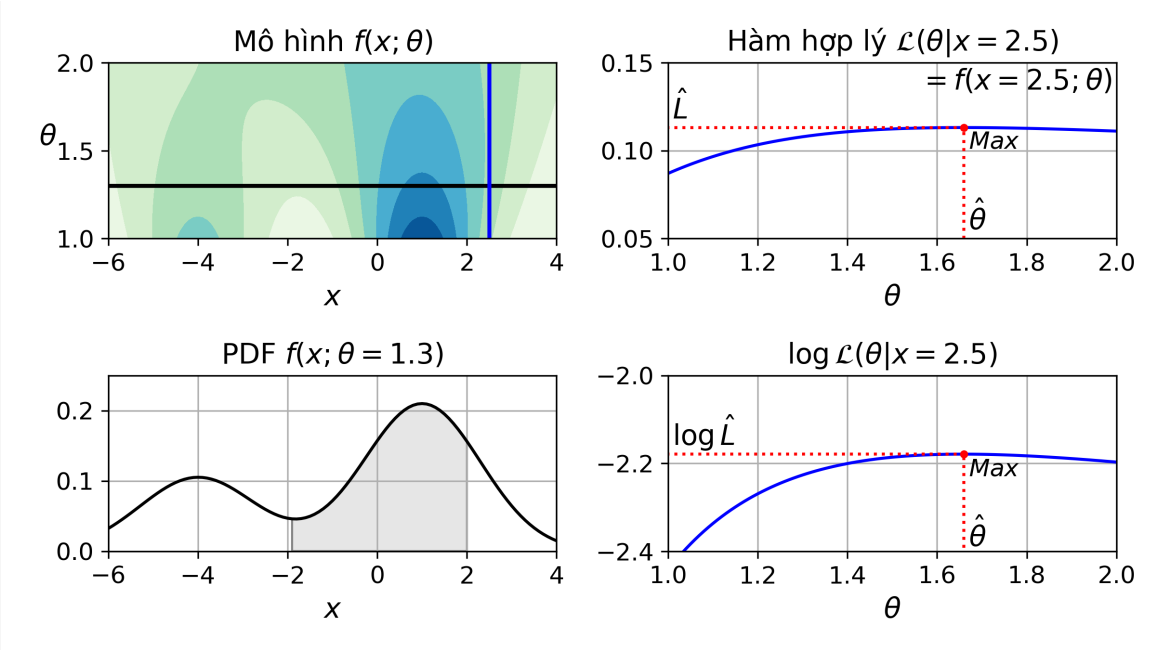
Cả BIC và AIC đều phạt các mô hình có nhiều tham số cần học hơn (ví dụ như nhiều cụm hơn) và thưởng cho các mô hình khớp tốt trên dữ liệu. Nhìn chung, chúng thường chọn ra cùng một mô hình. Nếu các mô hình được lựa chọn khác nhau, mô hình được chọn bởi BIC thường đơn giản hơn (ít tham số hơn) mô hình được chọn bởi AIC, nhưng thường khớp dữ liệu không tốt bằng (điều này đặc biệt đúng với các tập dữ liệu lớn).

Hàm Hợp lý

Thuật ngữ "probability" và "likelihood" hay được dùng thay cho nhau trong tiếng Anh, nhưng chúng có ý nghĩa rất khác nhau trong thống kê. Cho trước một mô hình thống kê với các tham số $\boldsymbol{\theta}$, thuật ngữ "xác suất" (*probability*) được sử dụng để mô tả khả năng xảy ra của kết quả $\mathbf{x}$ (khi biết các giá trị tham số $\boldsymbol{\theta}$), trong khi thuật ngữ "hợp lý" (*likelihood*) được sử dụng để mô tả tính hợp lý của một bộ giá trị các tham số $\boldsymbol{\theta}$, sau khi đã biết kết quả $\mathbf{x}$.

Xét một mô hình hỗn hợp một chiều của hai phân phối Gauss với tâm tại -4 và +1. Để đơn giản hóa vấn đề, mô hình thử nghiệm này chỉ có một tham số θ kiểm soát độ lệch chuẩn của cả hai phân phối. Đồ thị đường đồng mức góc trên bên trái tại Hình 9.20 biểu diễn toàn bộ mô hình $f(x;\theta)$ dưới dạng hàm số của x và θ. Để ước lượng phân phối xác xuất của một kết quả x, ta cần xác định trước giá trị tham số mô hình θ. Ví dụ, nếu chỉ định θ bằng 1.3 (đường ngang), ta có hàm mật độ xác suất $f(x;\theta = 1.3)$ biểu diễn tại đồ thị góc dưới bên trái. Giả sử ta cần ước lượng xác suất mà x nằm trong khoảng -2 đến +2, ta sẽ phải tính tích phân của hàm mật độ xác suất trong khoảng này (vùng được tô đậm). Tuy nhiên, nếu ta không biết θ mà thay vào đó quan sát được một mẫu $x = 2.5$ (đường thẳng đứng ở đồ thị góc trên bên trái) thì sao? Trong trường hợp này, ta có hàm hợp lý $\mathcal{L}(\theta|x = 2.5) = f(x = 2.5;\theta)$ được biểu diễn ở đồ thị góc trên bên phải.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 9.20. Hàm tham số của mô hình (góc trên bên trái) và các hàm được trích xuất từ nó: hàm mật độ xác suất (góc dưới bên trái), hàm hợp lý (góc trên bên phải), và hàm log-hợp lý (góc dưới bên phải)

Tóm lại, giá trị hàm mật độ xác suất là một hàm của x (với θ cố định). Một điều quan trọng cần biết đó là hàm hợp lý *không* phải là một phân phối xác suất: nếu ta tính tích phân một phân phối xác suất qua tất cả các giá trị khả dĩ của x, ta luôn nhận được kết quả bằng 1; nhưng nếu ta tính tích phân hàm hợp lý của tất cả các giá trị khả dĩ của θ, kết quả có thể là bất kỳ giá trị dương nào.

Cho trước tập dữ liệu $\mathbf{X}$, một tác vụ phổ biến đó là cố gắng ước lượng giá trị hợp lý nhất cho các tham số của mô hình. Để làm được điều này, ta cần tìm được giá trị mà tại đó hàm hợp lý đạt cực đại, với $\mathbf{X}$ cho trước. Trong ví dụ trên, nếu ta quan sát được một mẫu $x = 2.5$, *ước lượng hợp lý cực đại* (*maximum likelihood estimate – MLE*) của θ là $\hat{\theta} = 1.5$. Nếu tồn tại phân phối xác suất tiên nghiệm g của θ, ta có thể tận dụng nó bằng cách cực đại hóa $\mathcal{L}(\theta|x)g(\theta)$ thay vì chỉ cực đại hóa $\mathcal{L}(\theta|x)$. Đây còn được gọi là ước lượng *hậu nghiệm cực đại* (*maximum a-posteriori – MAP*). Vì MAP ràng buộc các giá trị tham số, ta có thể coi nó như là một phiên bản được điều chuẩn của MLE.

Lưu ý rằng việc cực đại hóa hàm hợp lý cũng tương tự như việc cực đại hóa hàm logarit của nó (được biểu diễn bởi đồ thị góc dưới bên phải tại Hình 9.20). Thật vậy, hàm logarit là một hàm tăng nghiêm ngặt, vì vậy việc θ cực đại hóa hàm log hợp lý cũng đồng nghĩa với việc nó cực đại hóa hàm hợp lý. Thực chất, việc cực đại hóa hàm log hợp lý thường dễ hơn so với hàm gốc. Ví dụ, nếu ta quan sát được một số mẫu độc lập $x^{(1)}$ tới $x^{(m)}$, ta sẽ cần tìm giá trị của θ mà tại đó tích của các hàm hợp lý riêng lẻ là cực đại. Tuy nhiên ta có thể làm điều tương tự nhưng đơn giản hơn bằng cách cực đại hóa hàm tổng (không phải tích) của các hàm log hợp lý, nhờ vào tính chất chuyển tích thành tổng của hàm logarit: $\log(ab) = \log(a) + \log(b)$.

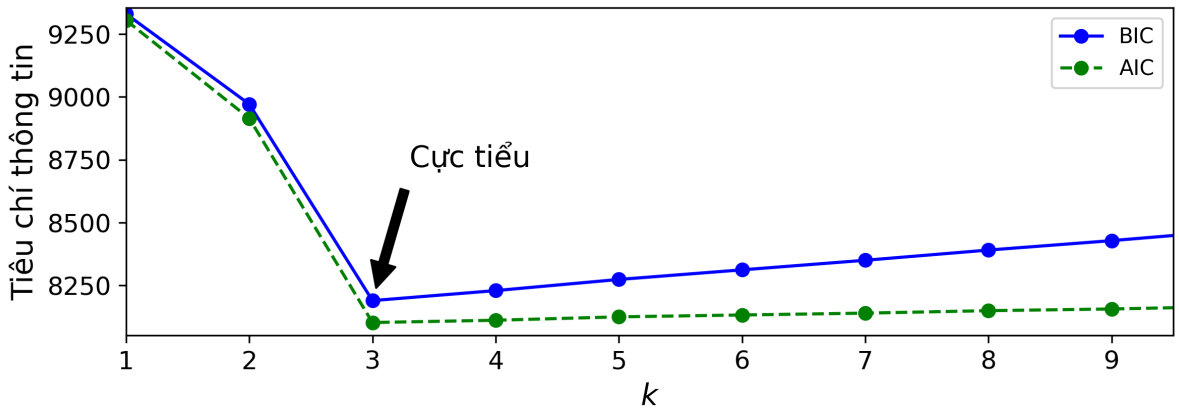
Một khi đã ước lượng được $\hat{\theta}$, giá trị của θ mà tại đó hàm hợp lý đạt cực đại, thì ta đã sẵn sàng để tính toán $\widehat{L}(\hat{\theta}, \mathbf{X})$, giá trị được sử dụng để tính toán AIC và BIC. Ta có thể xem giá trị này như một phép đo để đánh giá độ khớp của mô hình với dữ liệu.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Để tính BIC và AIC, ta có thể sử dụng phương thức `bic()` và `aic()`:

```
>>> gm.bic(X)
8189.74345832983
>>> gm.aic(X)
8102.518178214792
```

Hình 9.21 biểu diễn BIC với các số cụm k khác nhau. Cả BIC và AIC đều thấp nhất khi $k = 3$, nên đây rất có thể là lựa chọn tốt nhất. Chú ý rằng ta cũng có thể tìm giá trị tốt nhất cho siêu tham số `covariance_type`. Ví dụ, nếu tham số này là `"spherical"` thay vì `"full"`, mô hình có ít tham số để học hơn nhiều, nhưng lại không khớp với dữ liệu tốt bằng.



Hình 9.21. AIC và BIC cho các số cụm k *khác nhau*

Mô hình Hỗn hợp Bayes Gauss

Thay vì tìm kiếm số cụm tối ưu một cách thủ công, ta có thể sử dụng lớp `BayesianGaussianMixture`. Lớp này có thể gán trọng số bằng (hoặc gần bằng) không cho những cụm không cần thiết. Ta chỉ cần đặt số cụm `n_components` là một giá trị mà lớn hơn số cụm tối ưu (cần có căn cứ và thường đòi hỏi một chút kiến thức tối thiểu về bài toán), và thuật toán sẽ tự động loại trừ những cụm không cần thiết. Ví dụ, hãy thử đặt số cụm là 10 và xem điều gì sẽ xảy ra:

```
>>> from sklearn.mixture import BayesianGaussianMixture
>>> bgm = BayesianGaussianMixture(n_components=10, n_init=10)
>>> bgm.fit(X)
>>> np.round(bgm.weights_, 2)
array([0.4 , 0.21, 0.4 , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ])
```

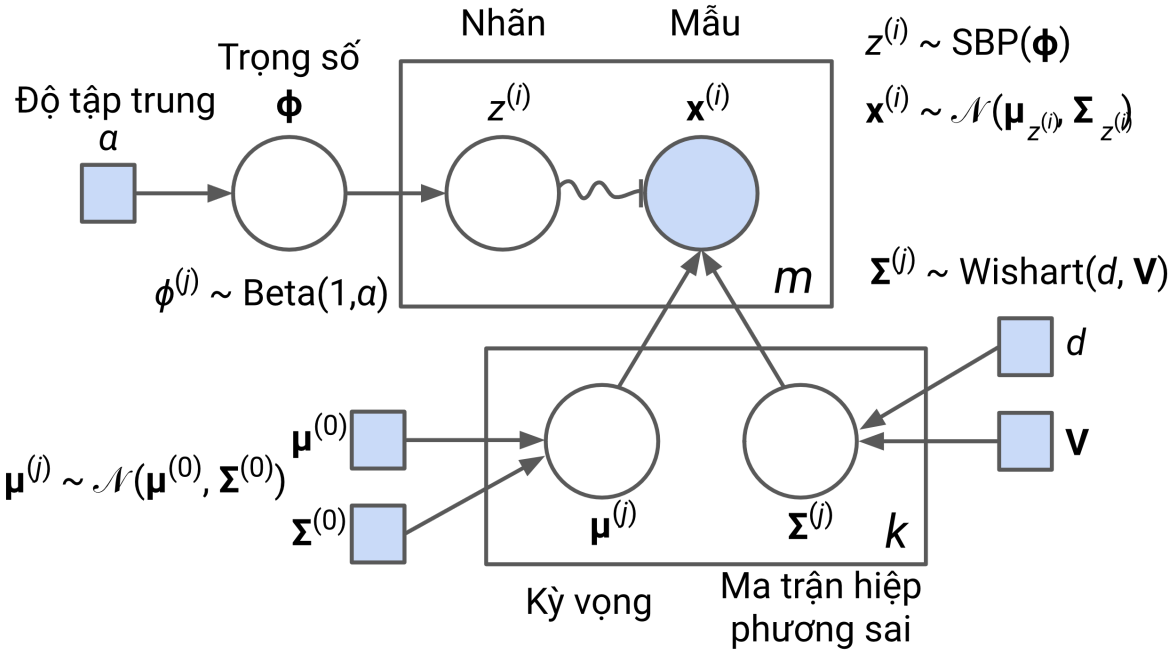
Hoàn hảo, thuật toán tự động phát hiện chỉ có ba cụm thật sự cần thiết và các cụm thu được gần giống hệt các cụm trong Hình 9.17.

Trong mô hình này, các tham số cụm (bao gồm các trọng số, trung bình và ma trận hiệp phương sai) không còn được xem là các tham số mô hình cố định mà là các biến ngẫu nhiên tiềm ẩn, giống như việc gán cụm (xem Hình 9.22). Vì vậy, $\mathbf{z}$ bây giờ bao gồm cả các tham số cụm và kết quả gán cụm.

Phân phối Beta thường được dùng để mô hình hóa các biến ngẫu nhiên có giá trị nằm trong một khoảng cố định. Trong trường hợp này, khoảng giá trị kéo dài từ 0 đến 1. Quy trình Chia

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

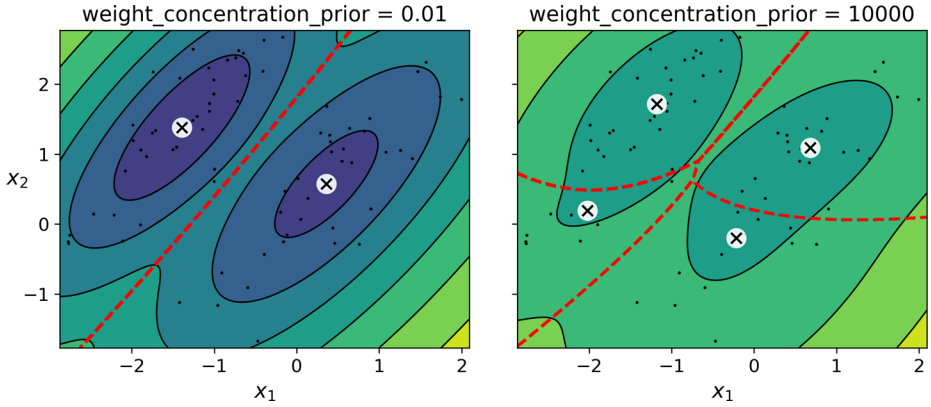
Đoạn (*Stick-Breaking Process – SBP*) có thể được giải thích rõ ràng thông qua ví dụ sau. Giả sử $\Phi = [0.3, 0.6, 0.5, \ldots]$, trong đó 30% mẫu sẽ được gán cho cụm 0, 60% mẫu tiếp theo được gắn cho cụm 1, 50% mẫu còn lại được gán cho cụm 2, và cứ thế tiếp tục. Quy trình này là một mô hình tốt cho các tập dữ liệu trong đó các mẫu mới có khuynh hướng tham gia vào các cụm lớn thay vì các cụm nhỏ (như việc người dân có khuynh hướng di chuyển đến các thành phố lớn hơn). Nếu độ tập trung α cao, thì các giá trị Φ sẽ gần với 0, và SBP sẽ sinh ra nhiều cụm. Ngược lại, nếu độ tập trung thấp thì các giá trị Φ sẽ gần với 1, và SBP sẽ có ít cụm. Cuối cùng, phân phối Wishart được sử dụng để lấy mẫu ma trận hiệp phương sai: các tham số d và $\mathbf{V}$ kiểm soát phân phối hình dạng của cụm.



Hình 9.22. Mô hình hỗn hợp Bayes Gauss

Kiến thức có trước về các biến tiềm ẩn $\mathbf{z}$ có thể được mã hóa thành phân phối xác suất $p(\mathbf{z})$ được gọi là *tiên nghiệm* (*prior*). Ví dụ, trước đó ta có thể tin rằng các cụm có ít mẫu (độ tập trung thấp) hay ngược lại các cụm chứa nhiều mẫu (độ tập trung cao). Niềm tin có sẵn về số cụm có thể được thay đổi bằng cách điều chỉnh siêu tham số `weight_concentration_prior`. Việc đặt giá trị này bằng 0.01 hoặc 10,000 sẽ dẫn đến các cụm rất khác nhau (xem Hình 9.23). Tuy nhiên, nếu có càng nhiều dữ liệu thì tiên nghiệm càng có ít ảnh hưởng. Thực chất, để vẽ được đồ thị với sự khác biệt lớn như vậy, ta phải sử dụng các tiên nghiệm rất mạnh và ít dữ liệu.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.



Hình 9.23. Việc sử dụng các mức độ tập trung tiên nghiệm khác nhau trên cùng tập dữ liệu dẫn đến số cụm khác nhau

Định lý Bayes (Phương trình 9.2) cho ta cách cập nhật phân phối xác suất theo biến tiềm ẩn sau khi quan sát được một phần dữ liệu $\mathbf{X}$. Công thức này tính phân phối *hậu nghiệm* (*posterior*) $p(\mathbf{z}|\mathbf{X})$, tức xác suất có điều kiện của $\mathbf{z}$ khi biết $\mathbf{X}$.

$$p(\mathbf{z}|\mathbf{X}) = \text{Hậu Nghiệm} = \frac{\text{Hợp Lý} \times \text{Tiên Nghiệm}}{\text{Bằng Chứng}} = \frac{p(\mathbf{X}|\mathbf{z}) \times p(\mathbf{z})}{p(\mathbf{X})}$$

Phương trình 9.2. Định lý Bayes

Đáng tiếc, với mô hình hỗn hợp Gauss (và nhiều bài toán khác), mẫu số $p(\mathbf{X})$ là bất khả tính vì ta cần phải lấy tích phân trên mọi giá trị khả dĩ của $\mathbf{z}$ (Phương trình 9.3). Điều này đòi hỏi ta phải xem xét tất cả các tổ hợp của tham số cụm và cách gán cụm khả thi.

$$p(\mathbf{X}) = \int p(\mathbf{X}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Phương trình 9.3. Bằng chứng $p(\mathbf{X})$ thường bất khả tính

Sự bất khả tính này là một trong những bài toán chính trong thống kê Bayes, và ta có một vài phương pháp để giải quyết bài toán này. Một trong những phương pháp này là *suy luận biến phân* (*variational inference*), trong đó ta chọn một nhóm phân phối $q(\mathbf{z};\boldsymbol{\lambda})$ với *tham số biến phân* (*variational parameters*) $\boldsymbol{\lambda}$ (lambda), rồi tối ưu tham số đó để $q(\mathbf{z})$ trở thành một xấp xỉ tốt của $p(\mathbf{z}|\mathbf{X})$. Ta đạt được điều này bằng cách tìm giá trị của $\boldsymbol{\lambda}$ để cực tiểu hóa phân kỳ KL từ $q(\mathbf{z})$ tới $p(\mathbf{z}|\mathbf{X})$, ký hiệu là $D_{KL}(q||p)$. Công thức tính độ phân kỳ KL nằm ở Phương trình 9.4, và nó có thể được viết thành logarit của bằng chứng ($\log p(\mathbf{X})$) trừ *cận dưới bằng chứng* (*evidence lower bound* – ELBO). Vì logarit của bằng chứng không phụ thuộc vào q, số hạng này là một hằng số, do đó việc cực tiểu hóa độ phân kỳ KL chỉ đơn thuần là cực đại hóa ELBO.

Trong thực tiễn, ta có vài kỹ thuật khác nhau để cực đại hóa ELBO. Trong *suy luận biến phân trường trung bình* (*mean field variational inference*), ta cần phải chọn nhóm phân phối $q(\mathbf{z};\boldsymbol{\lambda})$ và tiên nghiệm $p(z)$ một cách rất cẩn thận để đảm bảo rằng công thức tính ELBO có thể được tối giản về dạng mà ta có thể tính được. Đáng tiếc là không tồn tại phương pháp tổng quát để thực hiện việc này. Việc lựa chọn đúng nhóm phân phối và tiên nghiệm sẽ tùy thuộc vào từng tác vụ và đòi hỏi một chút kỹ năng toán học. Ví dụ, các phân phối và phương trình cận dưới trong lớp `BayesianGaussianMixture` của Scikit-Learn được trình bày trong tài liệu. Từ các phương trình này ta có thể suy ra phương trình cập nhật cho tham số cụm và biến gán cụm, rồi sử dụng chúng tương tự như trong thuật toán Kỳ vọng-Cực đại hóa. Thực chất, độ phức tạp

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

$$
\begin{aligned}
D_{KL} &= \mathbb{E}_q \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{X})} \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{z}|\mathbf{X}) \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}) - \log \frac{p(\mathbf{z}, \mathbf{X})}{p(\mathbf{X})} \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}) - \log p(\mathbf{z}, \mathbf{X}) + \log p(\mathbf{X}) \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}) \right] - \mathbb{E}_q \left[\log p(\mathbf{z}, \mathbf{X}) \right] + \mathbb{E}_q \left[\log p(\mathbf{X}) \right] \\
&= \mathbb{E}_q \left[\log p(\mathbf{X}) \right] - \left(\mathbb{E}_q \left[\log p(\mathbf{z}, \mathbf{X}) \right] - \mathbb{E}_q \left[\log q(\mathbf{z}) \right] \right) \\
&= \log p(\mathbf{X}) - \text{ELBO} \\
&\text{trong đó ELBO} = \mathbb{E}_q \left[\log p(\mathbf{z}, \mathbf{X}) \right] - \mathbb{E}_q \left[\log q(\mathbf{z}) \right]
\end{aligned}
$$

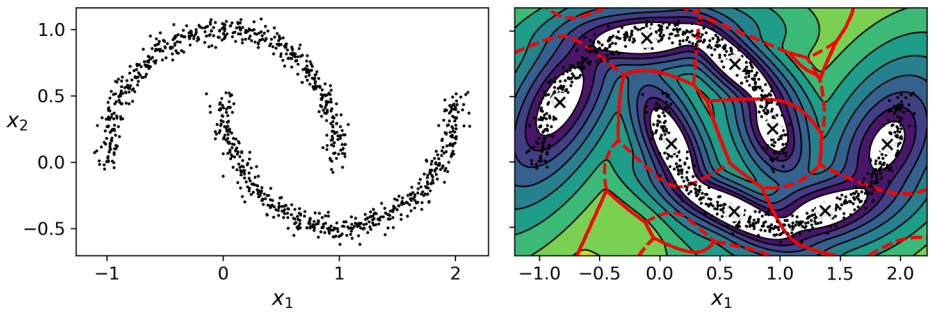
Phương trình 9.4. Độ phân kỳ KL từ $q(\mathbf{z})$ tới $p(\mathbf{z}|\mathbf{X})$

tính toán của lớp `BayesianGaussianMixture` gần giống với độ phức tạp của lớp `GaussianMixture` (nhưng thường sẽ chậm hơn đáng kể). Một cách đơn giản hơn để cực đại hóa ELBO được gọi là *suy luận biến phân hộp đen ngẫu nhiên* (*black box stochastic variational inference* – BBSVI). Trong phương pháp này, tại mỗi vòng lặp ta sẽ lấy một vài mẫu từ q, và chúng sẽ được dùng để ước lượng gradient của ELBO theo tham số biến phân $\boldsymbol{\lambda}$, rồi gradient này được dùng trong bước nâng gradient. Phương pháp này cho phép ta sử dụng suy luận Bayes với bất cứ loại mô hình nào (miễn là nó khả vi), bao gồm cả mạng nơ-ron sâu. Việc sử dụng suy luận Bayes với mạng nơ-ron sâu được gọi là Học Sâu Bayes.

Mẹo

Nếu bạn muốn đi sâu hơn vào thống kê Bayes, hãy tham khảo cuốn sách *Bayesian Data Analysis* của Andrew Gelman và cộng sự (Chapman & Hall).

Mô hình hỗn hợp Gauss chỉ hoạt động tốt với các cụm có dạng ellipsoid, nên nếu ta thử khớp một tập dữ liệu với nhiều hình dạng khác nhau, có thể ta sẽ không đạt được kết quả mong muốn. Để lấy ví dụ, hãy thử sử dụng mô hình hỗn hợp Bayes Gauss để phân cụm tập dữ liệu moons (xem Hình 9.24).



Hình 9.24. Khớp một mô hình hỗn hợp Gauss cho các cụm không có dạng Ellipsoid

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thuật toán vẫn tiếp tục tìm kiếm các cụm có dạng ellipsoid, vì thế kết quả trả về là tám cụm khác nhau thay vì hai cụm. Kết quả ước lượng mật độ cũng không quá tệ, nên mô hình này có thể được dùng cho tác vụ phát hiện bất thường, nhưng nó đã thất bại trong việc phát hiện ra hai cụm hình bán nguyệt. Vậy nên, ta hãy cùng điểm qua ngay một vài thuật toán phân cụm có thể hoạt động với các cụm có hình dạng bất kỳ.

Các Thuật toán Phát hiện Bất thường và Tính mới

Scikit-Learn cung cấp nhiều thuật toán dành riêng cho tác vụ phát hiện bất thường hoặc tính mới:

- PCA (cũng như các kỹ thuật giảm chiều khác có phương thức `inverse_transform()`): Nếu so với lỗi khôi phục của một mẫu bình thường, lỗi khôi phục của một mẫu bất thường thường sẽ lớn hơn đáng kể. Đây là một phương pháp đơn giản và hiệu quả để phát hiện bất thường (tham khảo phần bài tập của chương này để biết một cách ứng dụng của phương pháp này).

- Fast-MCD (*minimum covariance determinant* – tạm dịch *định thức hiệp phương sai nhỏ nhất*): Được lập trình trong lớp `EllipticEnvelope`, và đây là thuật toán hữu ích cho việc phát hiện ngoại lai, cụ thể là cho việc làm sạch dữ liệu. Nó giả định rằng các mẫu bình thường (*inlier*) được sinh từ một phân phối Gauss đơn lẻ (không phải hỗn hợp các phân phối). Nó cũng giả định rằng các ngoại lai trong tập dữ liệu không được sinh bởi phân phối Gauss này. Khi thuật toán ước lượng tham số của phân phối Gauss (tức hình dạng của ellipsoid bao xung quanh các mẫu bình thường), các mẫu có khả năng cao là ngoại lai sẽ được thuật toán cẩn thận bỏ qua. Phương pháp này đưa ra một ước lượng tốt hơn của đường bao ellipsoid, nhờ đó chính xác hơn trong việc phát hiện ngoại lai.

- Rừng Cô lập (*Isolation Forest*): Đây là một thuật toán hiệu quả cho tác vụ phát hiện ngoại lai, đặc biệt với dữ liệu nhiều chiều. Thuật toán xây dựng một Rừng Ngẫu nhiên mà mỗi Cây Quyết định được dựng một cách ngẫu nhiên: tại mỗi nút, đặc trưng và ngưỡng sẽ được chọn một cách ngẫu nhiên (giữa giá trị lớn nhất và nhỏ nhất) để chia tập dữ liệu ra làm hai. Tập dữ liệu tiếp tục được chia nhỏ bằng cách này cho đến khi các mẫu hoàn toàn biệt lập với nhau. Các điểm bất thường cách xa so với các mẫu còn lại, vậy nên nhìn chung (trên toàn bộ các Cây Quyết định) chúng sẽ bị cô lập nhanh hơn so với những mẫu bình thường.

- Nhân tố Ngoại lai Cục bộ (*Local Outlier Factor – LOF*): Thuật toán này cũng hoạt động tốt cho tác vụ phát hiện ngoại lai. Thuật toán so sánh mật độ xung quanh một mẫu cho trước với mật độ của các điểm gần nhất. Một điểm ngoại lai thường sẽ tách biệt hơn so với k điểm gần nhất của nó.

- SVM Một lớp (*One-class SVM*): Thuật toán này thích hợp với tác vụ phát hiện tính mới. Nhắc lại rằng một bộ phân loại SVM sử dụng hạt nhân sẽ phân tách hai lớp bằng cách (ngầm) ánh xạ tất cả các mẫu lên một không gian nhiều chiều, sau đó tách chúng ra bằng một bộ phân loại SVM tuyến tính trong không gian này (tham khảo Chương 5). Bởi vì ta chỉ có một lớp duy nhất, thuật toán SVM một lớp sẽ cố gắng phân tách các mẫu khỏi gốc tọa độ trong không gian nhiều chiều. Trong không gian gốc, điều này đồng nghĩa với việc tìm kiếm một vùng nhỏ có thể bao bọc tất cả các mẫu. Nếu một mẫu mới không nằm trong vùng này, nó là một bất thường. Thuật toán có một vài siêu tham số mà ta có thể điều chỉnh: các siêu tham số thông thường trong SVM hạt nhân và một siêu tham số biên tương ứng với xác suất mẫu mới bị xét nhầm thành điểm mới trong khi nó hoàn toàn bình thường. Thuật toán hoạt động rất ổn, đặc biệt là khi làm việc với tập dữ liệu nhiều chiều, nhưng cũng giống với SVM, nó không mở rộng tốt với các tập dữ liệu lớn.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Bài tập

1. Bạn định nghĩa phân cụm là gì? Bạn có thể kể tên một số thuật toán phân cụm không?
2. Bạn có thể liệt kê một vài ứng dụng chính của các thuật toán phân cụm không?
3. Mô tả hai kỹ thuật chọn số cụm phù hợp khi sử dụng K-điểm trung bình.
4. Lan truyền nhãn là gì? Tại sao ta cần lập trình nó và làm điều đó như thế nào?
5. Bạn có thể kể tên hai thuật toán phân cụm có khả năng mở rộng với tập dữ liệu lớn không? Liệu bạn có thể tìm hai thuật toán nữa có khả năng tìm các vùng có mật độ cao không?
6. Bạn có thể tìm một trường hợp mà học chủ động sẽ hữu dụng không? Bạn sẽ cài đặt nó như thế nào?
7. Điểm khác biệt giữa phát hiện bất thường và phát hiện tính mới là gì?
8. Mô hình hỗn hợp Gauss là gì? Bạn có thể sử dụng nó trong tác vụ nào?
9. Bạn có thể kể tên hai kỹ thuật tìm kiếm số cụm phù hợp khi sử dụng mô hình hỗn hợp Gauss không?
10. Tập dữ liệu khuôn mặt Olivetti cổ điển chứa 400 ảnh xám với kích thước 64×64 của các khuôn mặt. Mỗi ảnh đã được biến đổi về thành một vector một chiều với kích thước 4,096. Ảnh chụp là của 40 người khác nhau (mỗi người 10 tấm), và ta cần huấn luyện một mô hình có thể dự đoán người được chụp trong bức ảnh. Ta nạp tập dữ liệu thông qua hàm `sklearn.datasets.fetch_olivetti_faces()`, rồi tách ra thành các tập huấn luyện, kiểm định và kiểm tra (chú ý rằng tập dữ liệu đã được co về khoảng 0 đến 1). Bởi vì tập dữ liệu khá nhỏ, ta có thể sẽ cần sử dụng lấy mẫu stratified để đảm bảo rằng ta có cùng số lượng ảnh của mỗi người trong mỗi tập. Tiếp đến, hãy phân cụm các ảnh bằng K-điểm trung bình, và đảm bảo rằng bạn đã chọn một số lượng cụm hợp lý (sử dụng một trong những phương pháp đã thảo luận trong chương này). Trực quan hóa các cụm: bạn có thấy các khuôn mặt giống nhau trong mỗi cụm không?
11. Tiếp tục với tập dữ liệu khuôn mặt Olivetti, hãy huấn luyện một bộ phân loại để dự đoán người trong bức ảnh và thực hiện đánh giá trên tập kiểm định. Tiếp theo, hãy sử dụng K-điểm trung bình để giảm chiều và huấn luyện bộ phân loại trên tập dữ liệu mới này. Hãy tìm kiếm số lượng cụm sao cho bộ phân loại đạt chất lượng tốt nhất: Bạn đạt được con số bao nhiêu? Còn nếu bạn thêm những đặc trưng của tập dữ liệu đã giảm chiều này vào các đặc trưng ban đầu thì sao (một lần nữa, hãy tìm số lượng cụm tốt nhất)?
12. Hãy huấn luyện một mô hình hỗn hợp Gauss trên tập dữ liệu khuôn mặt Olivetti. Để tăng tốc thuật toán, ta nên giảm chiều của tập dữ liệu (ví dụ như sử dụng PCA, bảo toàn 99% phương sai). Hãy sử dụng mô hình để sinh ra một vài khuôn mặt mới (sử dụng phương thức `sample()`) và biểu diễn chúng (nếu sử dụng PCA, bạn sẽ cần đến phương thức `inverse_transform()`). Hãy thử biến đổi các tấm ảnh một chút (ví dụ như xoay, lật, làm tối) và xem liệu mô hình có thể nhận biết các bất thường không (tức so sánh kết quả của phương thức `score_samples()` giữa ảnh bình thường và bất thường).
13. Một vài kỹ thuật giảm chiều cũng có thể được sử dụng cho phát hiện bất thường. Ví dụ, hãy thực hiện giảm chiều trên tập dữ liệu khuôn mặt Olivetti với PCA, sao cho 99% phương sai được bảo toàn. Sau đó hãy tính toán lỗi khôi phục của từng ảnh. Tiếp đến, hãy lấy một vài ảnh đã qua chỉnh sửa trong bài tập trước và xem lỗi khôi phục của chúng: chú ý việc lỗi khôi phục lớn hơn như thế nào. Nếu ta hiển thị ảnh đã được khôi phục, ta sẽ thấy lý do: nó cố gắng tái hiện lại một khuôn mặt bình thường.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lời giải của các bài tập trên có thể tìm được trong Phụ lục A.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phụ lục A

Lời giải các Bài tập

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Ghi chú

Lời giải cho các bài tập lập trình có thể được xem trực tuyến trong các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 1: Toàn cảnh Học Máy

1. Học Máy là việc xây dựng các hệ thống có thể học từ dữ liệu. Học có nghĩa là thực hiện một tác vụ nào đó tốt hơn, theo một thang đo chất lượng xác định.
2. Học Máy rất hiệu quả cho các vấn đề phức tạp mà không có thuật toán nào giải được, để thay thế các chuỗi dài quy luật được thiết kế thủ công, để xây dựng các hệ thống có thể thích ứng với môi trường biến động, và cuối cùng là để giúp con người học (ví dụ như khai phá dữ liệu).
3. Một tập huấn luyện có gán nhãn là một tập huấn luyện có chứa đáp án mong muốn (gọi là nhãn) cho mỗi mẫu.
4. Hai tác vụ có giám sát phổ biến nhất là hồi quy và phân loại.
5. Tác vụ không giám sát phổ biến gồm có phân cụm, biễu diễn, giảm chiều dữ liệu, và học luật kết hợp.
6. Học Tăng cường nhiều khả năng sẽ hoạt động tốt nhất nếu ta muốn robot có thể học cách đi trong nhiều địa hình chưa biết vì đây là dạng vấn đề mà Học Tăng cường thường đối phó. Ta có thể biểu diễn bài toán dưới dạng tác vụ giám sát hoặc bán giám sát, nhưng như vậy sẽ không tự nhiên bằng.
7. Nếu bạn không biết trước các nhóm cần tách biệt thì có thể sử dụng thuật toán phân cụm (học không giám sát) để chia phân khúc thành các cụm khách hàng tương tự nhau. Tuy nhiên, nếu đã biết những nhóm cần chia thì ta có thể đưa các mẫu của mỗi nhóm vào một thuật toán phân loại (học có giám sát) và nó sẽ phân loại khách hàng vào các nhóm đấy.
8. Phân loại thư rác là một bài toán học có giám sát điển hình: thuật toán được cho xem nhiều mẫu thư cùng với nhãn của chúng (là thư rác hay không).
9. Một hệ thống học trực tuyến có thể học từ dòng dữ liệu gia tăng, khác với hệ thống học theo batch. Điều này khiến nó có thể thích ứng nhanh chóng với thay đổi trong cả dữ liệu hay các hệ thống tự động, và có khả năng huấn luyện trên các lượng dữ liệu cực lớn.

10. Thuật toán "ngoài bộ nhớ chính" có thể xử lý lượng dữ liệu khổng lồ mà không thể chứa trong bộ nhớ chính của máy tính. Một thuật toán "ngoài bộ nhớ chính" chia dữ liệu thành các mini-batch và sử dụng các kỹ thuật học trực tuyến để học từ các mini-batch này.

11. Một hệ thống học theo mẫu sẽ học thuộc lòng các mẫu huấn luyện; rồi khi được cho một mẫu mới nó sẽ sử dụng một thang đo độ tương tự để tìm mẫu giống nhất trong tập huấn luyện và dùng nó để đưa ra dự đoán.

12. Mô hình có một hoặc nhiều tham số mô hình dùng để xác định giá trị dự đoán khi được cho một mẫu mới (ví dụ như là tham số độ dốc trong mô hình tuyến tính). Một thuật toán học sẽ cố gắng tìm các giá trị tối ưu cho các tham số đó, sao cho mô hình sẽ khái quát hóa tốt cho các mẫu mới. Một siêu tham số là một tham số của chính bản thân thuật toán học, không phải của mô hình (ví dụ như lượng điều chuẩn sẽ áp dụng).

13. Thuật toán học dựa trên mô hình sẽ tìm giá trị tối ưu cho các tham số mô hình sao cho khái quát hóa tốt trên các mẫu mới. Ta thường huấn luyện các hệ thống như vậy bằng cách tối thiểu hóa một hàm chi phí đo mức độ sai lệch của các dự đoán của mô hình trên tập huấn luyện, cộng với một lượng phạt cho mức độ phức tạp của mô hình nếu mô hình được điều chuẩn. Để đưa ra dự đoán, ta đưa các đặc trưng của một mẫu mới vào hàm dự đoán của mô hình, sử dụng giá trị tham số tìm được từ thuật toán học.

14. Một vài thách thức chính trong Học máy là thiếu dữ liệu, chất lượng dữ liệu kém, dữ liệu không mang tính đại diện, đặc trưng thiếu thông tin, mô hình quá đơn giản luôn dưới khớp dữ liệu huấn luyện, và các mô hình quá phức tạp tới mức quá khớp dữ liệu.

15. Nếu một mô hình hoạt động tốt trên tập huấn luyện nhưng khái quát kém trên dữ liệu mới, mô hình đó nhiều khả năng là đã quá khớp dữ liệu huấn luyện (hoặc ta cực kỳ may mắn trên tập huấn luyện). Các giải pháp khả dĩ cho việc quá khớp gồm thu thập thêm dữ liệu, đơn giản hóa mô hình (chọn một thuật toán đơn giản hơn, giảm số lượng tham số hoặc đặc trưng, hoặc điều chuẩn mô hình), hoặc giảm mức độ nhiễu trong dữ liệu huấn luyện.

16. Một tập kiểm tra được dùng để ước lượng mức độ lỗi khái quát mà một mô hình sẽ có trên các mẫu dữ liệu mới, trước khi mô hình được chạy trong thực tế.

17. Một tập kiểm định được dùng để so sánh mô hình. Nó cho phép ta lựa chọn mô hình tốt nhất và tinh chỉnh các siêu tham số.

18. Tập huấn luyện–phát triển thì được dùng khi có nguy cơ dữ liệu huấn luyện, kiểm định và kiểm tra không khớp với nhau (điều mà ta nên giảm thiểu càng ít càng tốt và làm nó càng gần với dữ liệu sẽ gặp trong thực tế càng tốt). Tập huấn luyện–phát triển là một phần của tập huấn luyện được ta giữ lại (mô hình không huấn luyện trên đó). Mô hình sau khi được huấn luyện trên phần còn lại của tập huấn luyện sẽ được đánh giá trên cả tập huấn luyện–phát triển và tập kiểm định. Nếu mô hình hoạt động tốt trên tập huấn luyện nhưng không tốt trên tập huấn luyện–phát triển, khi đó mô hình có khả năng là đã quá khớp dữ liệu huấn luyện. Nếu nó hoạt động tốt trên cả tập huấn luyện và huấn luyện–phát triển, nhưng không tốt trên tập kiểm định, khi đó có khả năng là dữ liệu huấn luyện quá khác biệt so với dữ liệu kiểm định và kiểm tra, và ta nên cố gắng cải thiện dữ liệu huấn luyện để giống với dữ liệu kiểm định và kiểm tra hơn.

19. Nếu bạn tinh chỉnh siêu tham số trên tập kiểm tra, tập kiểm tra sẽ có khả năng bị quá khớp, và lỗi khái quát đo được sẽ làm bạn quá lạc quan (bạn có thể sẽ triển khai một mô hình hoạt động tệ hơn so với dự kiến).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Lời giải Chương 2: Dự án Học Máy từ Đầu tới Cuối

Vui lòng xem qua các Jupyter Notebook có sẵn tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 3: Bài toán Phân loại

Vui lòng xem qua các Jupyter Notebook có sẵn tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 4: Huấn luyện Mô hình

1. Nếu ta có một tập huấn luyện với hàng triệu đặc trưng, ta có thể sử dụng Hạ Gradient Ngẫu nhiên (Stochastic Gradient Descent), Hạ Gradient theo Mini-batch (Mini-batch Gradient Descent), hoặc thậm chí Hạ Gradient theo Batch (Batch Gradient Descent) nếu có đủ bộ nhớ cho tập huấn luyện. Tuy nhiên ta không thể sử dụng Phương Trình Pháp Tuyến (Normal Equation) hoặc SVD bởi độ phức tạp tính toán tăng rất nhanh (lớn hơn cấp số mũ bậc hai) theo số lượng các đặc trưng.

2. Nếu các đặc trưng trong tập huấn luyện có tỷ lệ rất khác nhau, hàm mất mát sẽ có dạng một cái tô dài, và vì thế các thuật toán Hạ Gradient sẽ mất khá nhiều thời gian để hội tụ. Để giải quyết vấn đề này, ta nên co giãn tập dữ liệu trước khi huấn luyện mô hình. Lưu ý rằng Phương Trình Pháp Tuyến hoặc SVD vẫn sẽ hoạt động tốt mà không cần co giãn tập huấn luyện. Thêm vào đó, các mô hình điều chuẩn có thể hội tụ tại một giải pháp không tối ưu nếu các đặc trưng không được chia tỷ lệ: việc điều chuẩn phạt các trọng số lớn, do đó các đặc trưng có giá trị nhỏ hơn thường có xu hướng bị bỏ qua so với các đặc trưng có giá trị lớn hơn.

3. Hạ Gradient không thể bị kẹt ở một điểm tối ưu cục bộ khi huấn luyện mô hình Hồi quy Logistic (Logistic Regression) bởi hàm mất mát có tính lồi.[1]

4. Nếu bài toán tối ưu hóa có tính lồi (giống như Hồi quy Tuyến Tính và Hồi quy Logistic), và giả sử tốc độ học không quá cao, thì tất cả các thuật toán Gradient Descent sẽ tiến đến điểm tối ưu toàn cục. Cuối cùng tạo ra các mô hình tương đối giống nhau. Tuy nhiên trừ khi ta giảm dần tốc độ học, Hạ Gradient Ngẫu Nhiên và Hạ Gradient theo Mini-batch sẽ không bao giờ thực sự hội tụ; thay vào đó, chúng sẽ dao động xung quanh điểm tối ưu toàn cục. Điều này có nghĩa là ngay cả khi ta để chúng chạy trong một thời gian dài, các thuật toán Hạ Gradient trên sẽ tạo ra các mô hình hơi khác nhau một chút.

5. Nếu lỗi kiểm định tăng liên tục sau mỗi epoch, thì khả năng cao là tốc độ học quá cao và thuật toán đang phân kỳ. Nếu lỗi huấn luyện cũng tăng lên, thì chắc chắn là ta cần giảm tốc độ học. Tuy nhiên nếu lỗi huấn luyện không tăng lên, thì mô hình đã quá khớp (overfitting) với tập huấn luyện và ta nên dừng quá trình huấn luyện lại.

6. Do tính chất ngẫu nhiên, cả Hạ Gradient Ngẫu Nhiên và Hạ Gradient theo Mini-batch đều không đảm bảo rằng chúng sẽ đạt được tiến bộ ở mỗi vòng lặp huấn luyện. Vì vậy nếu ta ngừng huấn luyện ngay lập tức khi lỗi kiểm định tăng lên, ta có thể đã dừng quá sớm trước khi thuật toán đạt đến điểm tối ưu. Một lựa chọn tốt hơn đó là lưu mô hình theo các khoảng thời gian đều đặn; sau đó khi mô hình không được cải thiện trong một thời gian dài (có nghĩa là mô hình có thể sẽ không bao giờ vượt qua mức tối ưu gần nhất), bạn có thể hoàn tác về mô hình đã lưu tốt nhất.

[1] Nếu ta vẽ một đường thẳng giữa hai điểm bất kỳ trên một đường cong, đường thẳng đó không bao giờ cắt đường cong.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

7. Hạ Gradient Ngẫu nhiên có tốc độ vòng lặp huấn luyện nhanh nhất nếu ta chỉ xét trên một mẫu huấn luyện, vì vậy nó thường tiếp cận vùng lân cận của điểm tối ưu toàn cục đầu tiên (hoặc là Hạ Gradient theo Mini-batch với kích thước mini-batch rất nhỏ). Tuy nhiên, chỉ có Hạ Gradient theo Batch mới thực sự hội tụ khi có đủ thời gian huấn luyện. Như đã nói ở trên, Hạ Gradient Ngẫu nhiên và Hạ Gradient theo Batch sẽ dao động xung quanh điểm tối ưu, trừ khi ta giảm dần tốc độ học.

8. Nếu lỗi kiểm định cao hơn nhiều so với lỗi huấn luyện, khả năng cao là mô hình của chúng ta đã quá khớp (overfitting) tập huấn luyện. Một cách để khắc phục điều này đó là giảm bậc đa thức: một mô hình có ít bậc tự do hơn sẽ ít rủi ro bị quá khớp hơn. Một hướng tiếp cận khác ta có thể áp dụng đó là điều chuẩn mô hình—ví dụ, bằng cách thêm vào giá trị phạt ℓ_2 (Ridge) hoặc ℓ_1 (Lasso) vào hàm mất mát. Điều nay cũng giúp giảm bậc tự do của mô hình. Cuối cùng, ta có thể thử tăng kích thước tập huấn luyện.

9. Nếu cả lỗi huấn luyện và lỗi kiểm định khá lớn và xấp xỉ nhau, khả năng cao là mô hình của chúng ta dưới khớp (underfitting) tập huấn luyện, có nghĩa là mô hình có độ chệch cao. Trường hợp này ta nên thử giảm siêu tham số điều chuẩn α.

10. Xét các trường hợp sau:

    - Một mô hình được điều chuẩn thường hoạt động tốt hơn mô hình không được điều chuẩn, vì vậy ta nên sử dụng mô hình Hồi quy Ridge thay vì mô hình Hồi quy Tuyến tính đơn giản.
    - Hồi quy Lasso sử dụng giá trị phạt ℓ_1 có xu hướng ép trọng số xuống bằng 0. Điều này tạo ra các mô hình thưa, trong đó hầu hết các trọng số đều bằng 0 ngoại trừ những trọng số quan trọng nhất. Đây là một phương án để tiến hành lựa chọn đặc trưng một cách tự động, sẽ rất hiệu quả nếu ta nghi ngờ rằng chỉ có một vài đặc trưng thực sự quan trọng. Trường hợp không chắc chắn, ta nên chọn Hồi quy Ridge.
    - Elastic Net thường được ưa chuộng hơn Lasso vì Lasso có thể trở nên bất định trong một số trường hợp (khi một số đặc trưng có mối tương quan cao hoặc khi có nhiều đặc trưng hơn mẫu huấn luyện). Tuy nhiên, ta sẽ có thêm một siêu tham số cần phải hiệu chỉnh. Để giảm tính bất định của Lasso, ta có thể sử dụng Elastic Net với `l1_ratio` gần bằng 1.

11. Nếu ta muốn phân loại các ảnh ngoài trời/trong nhà và ban ngày/ban đêm, vì chúng không phải là các lớp riêng lẻ (tức là có thể kết hợp với nhau), ta nên huấn luyện hai bộ phân loại Hồi quy Logistic.

12. Vui lòng xem qua các Jupyter Notebook có sẵn tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 5: Máy Vector Hỗ trợ

1. Ý tưởng căn bản phía sau Máy vector Hỗ trợ (*Support Vector Machine – SVM*) là làm sao để tìm "con đường" rộng nhất ngăn cách các lớp. Nói cách khác, mục tiêu của thuật toán là tối đa hóa độ rộng biên giữa đường ranh giới quyết định phân chia hai lớp và các mẫu huấn luyện. Khi thực hiện phân loại biên mềm, SVM dung hoà việc phân loại hai lớp một cách hoàn hảo với việc có được "con đường" rộng nhất (tức là một số mẫu có thể nằm trong con đường này.) Một ý tưởng chính yếu khác đó là sử dụng các hạt nhân khi huấn luyện những tập dữ liệu phi tuyến.

2. Sau khi huấn luyện một SVM, một *vector hỗ trợ* là bất kỳ mẫu nào nằm trong "con đường" (xem câu trả lời trước), bao gồm cả đường biên của nó. Đường ranh giới quyết định hoàn toàn được xác định bởi các vector hỗ trợ. Bất kỳ mẫu nào nếu *không* phải là một vector hỗ

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

trợ (tức là không nằm trong khổ đường) đều không ảnh hưởng đến mô hình; ta có thể loại bỏ, di chuyển chúng, hoặc thêm vào các mẫu khác. Miễn là các mẫu này nằm ngoài con đường, ranh giới quyết định sẽ không thay đổi. Các dự đoán được tính toán dựa trên các vector hỗ trợ, thay vì toàn bộ tập huấn luyện.

3. Các bộ SVM cố gắng khớp với "con đường" lớn nhất có thể có giữa hai lớp (tham khảo câu trả lời đầu tiên), do đó nếu tập huấn luyện không được co dãn đặc trưng, SVM sẽ có xu hướng bỏ qua các đặc trưng nhỏ (tham khảo Hình 5.2).

4. Bộ phân loại SVM có thể tính toán khoảng cách giữa các mẫu kiểm tra và ranh giới quyết định, ta có thể sử dụng khoảng cách đó như một điểm tin cậy. Tuy nhiên, giá trị này không thể chuyển đổi trực tiếp thành xác suất ước lượng của lớp. Nếu ta đặt `probability=True` khi khởi tạo SVM trong Scikit–Learn, thì sau quá trình huấn luyện SVM sẽ hiệu chỉnh các xác suất bằng cách sử dụng Hồi quy Logistic (được huấn luyện bằng kiểm định chéo 5–fold trên dữ liệu huấn luyện) trên điểm số của SVM. Thao tác này sẽ thêm vào các phương thức `predict_proba()` và `predict_log_proba()` vào bộ SVM.

5. Câu hỏi này chỉ áp dụng cho SVM tuyến tính bởi SVM hạt nhân chỉ có thể sử dụng dạng đối ngẫu. Độ phức tạp tính toán của bài toán SVM gốc tỷ lệ với số mẫu huấn luyện m, trong khi độ phức tạp tính toán của bài toán đối ngẫu tỷ lệ với một giá trị trong khoảng từ m^2 đến m^3. Do đó, nếu có hàng triệu mẫu, ta nên sử dụng bài toán gốc bởi bài toán đối ngẫu sẽ rất chậm.

6. Nếu một bộ phân loại SVM được huấn luyện với hạt nhân RBF bị dưới khớp trên tập huấn luyện, khả năng cao là có quá nhiều điều chuẩn. Để giảm thiểu điều này, ta có thể tăng `gamma` hoặc `C` (hoặc cả hai).

7. Xét các tham số QP của bài toán biên cứng $\mathbf{H}'$, $\mathbf{f}'$, $\mathbf{A}'$, và $\mathbf{b}'$ (tham khảo Mục 5). Các tham số QP của bài toán biên mềm có m tham số bổ sung ($n_p = n + 1 + m$) và m ràng buộc bổ sung ($n_c = 2m$). Chúng có thể được định nghĩa như sau:

   - $\mathbf{H}$ bằng với $\mathbf{H}'$, thêm vào m cột giá trị 0 bên phải và m dòng giá trị 0 bên dưới:

$$\mathbf{H} = \begin{pmatrix} \mathbf{H}' & 0 & \cdots \\ 0 & 0 & \\ \vdots & & \ddots \end{pmatrix}$$

   - $\mathbf{f}$ bằng với $\mathbf{f}'$ thêm vào m phần tử, tất cả đều bằng giá trị của siêu tham số C.
   - $\mathbf{b}$ bằng với $\mathbf{b}'$ thêm vào m phần tử, tất cả đều bằng 0.
   - $\mathbf{A}$ bằng với $\mathbf{A}'$, thêm ma trận đơn vị $\mathbf{I}_m$ ($m \times m$) vào bên phải và $-\mathbf{I}_m$ vào bên dưới, các phần tử còn lại đều bằng 0: $\mathbf{A} = \begin{pmatrix} \mathbf{A}' & \mathbf{I}_m \\ \mathbf{0} & -\mathbf{I}_m \end{pmatrix}$

Lời giải cho các bài tập 8, 9, và 10 có thể được tìm thấy trong các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 6: Cây Quyết định

1. Một cây nhị phân cân bằng chứa m nút lá sẽ có chiều sâu là $\log_2(m)$,[2] được làm tròn lên. Một Cây Quyết định nhị phân (cây chỉ dự đoán đúng hoặc sai, cũng là trường hợp của mọi

[2] $\log_2$ là logarit cơ số 2; $\log_2(m) = \log(m)/\log(2)$.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

cây trong Scikit–Learn) sẽ đạt đến một sự cân bằng nhất định khi huấn luyện kết thúc, với mỗi lá là một mẫu huấn luyện nếu không có bất cứ ràng buộc nào. Vậy nên, nếu tập huấn luyện có một triệu mẫu, Cây Quyết định sẽ có độ sâu $\log_2(10^6) \approx 20$ (thực tế sẽ hơn một chút vì cây không phải lúc nào cũng cân bằng hoàn toàn)

2. Độ pha tạp Gini của một nút nhìn chung sẽ thấp hơn so với nút cha của nó. Đó là do thuật toán huấn luyện CART với hàm chi phí có xu hướng chia tách mỗi nút sao cho độ pha tạp theo trọng số của các nút con đạt tối tiểu. Tuy nhiên, vẫn có khả năng một nút có độ pha tạp Gini lớn hơn cha của nó, miễn rằng sự tăng này bù đắp nhiều hơn cho sự giảm về độ pha tạp của nút con còn lại. Để ví dụ, xét một nút chứa bốn mẫu của lớp A và một mẫu của lớp B. Độ pha tạp Gini của nó là $1 - (1/5)^2 - (4/5)^2 = 0.32$. Giờ giả sử tập dữ liệu là một chiều và các mẫu có thứ tự là: A, B, A, A, A. Bạn có thể xác minh rằng thuật toán sẽ phân tách nút này tại vị trí sau mẫu thứ hai, tạo thành một nút con với hai mẫu A, B và nút còn lại với A, A, A. Độ pha tạp Gini của nút đầu tiên là $1 - (1/2)^2 - (1/2)^2 = 0.5$; cao hơn so với nút cha. Nhưng điều này được bù đắp lại bằng nút nguyên chất còn lại, nên độ pha tạp Gini theo trọng số là $2/5 \times 0.5 + 3/5 \times 0 = 0.2$; nhỏ hơn so với nút cha.

3. Nếu một Cây Quyết định bị quá khớp với tập huấn luyện, thử giảm `max_depth` có thể là một ý tưởng tốt vì nó sẽ giới hạn mô hình lại, dẫn đến mô hình được điều chuẩn.

4. Cây Quyết định không quan tâm đến việc tập dữ liệu đã được co giãn hay cân bằng chưa. Đó là một trong những điểm tốt của chúng. Vậy nên nếu Cây Quyết định đang bị dưới khớp, co giãn lại các đặc trưng đầu vào chỉ tổ phí thời gian.

5. Độ phức tạp tính toán khi huấn luyện một Cây Quyết định là $O(n \times m \log(m))$. Vậy nên nếu gấp 10 tập huấn luyện, thời gian huấn luyện sẽ được nhân thêm

$$K = \frac{n \times 10m \times \log(10m)}{n \times m \times \log(m)} = 10 \times \frac{\log(10m)}{\log(m)}.$$

   Với $m = 10^6$, $K \approx 11.7$, vậy sẽ mất khoảng 11.7 tiếng để thực hiện huấn luyện.

6. Việc sắp xếp dữ liệu trước khi huấn luyện sẽ tăng thời gian chạy chỉ khi tập dữ liệu nhỏ hơn vài nghìn mẫu. Nếu tập huấn luyện có 100,000 mẫu, đặt `presort=True` sẽ làm chậm thời gian huấn luyện đi đáng kể.

Đối với đáp án của bài tập 7 và 8, hãy tham khảo các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 7: Học Ensemble và Rừng Ngẫu nhiên

1. Nếu bạn đã huấn luyện năm mô hình và tất cả đều có 95% precision, bạn có thể thử kết hợp chúng vào một ensemble biểu quyết; thường sẽ cho bạn kết quả tốt hơn. Phương pháp này sẽ hoạt động tốt nếu các mô hình đều khác biệt nhau (ví dụ, một bộ phân loại SVM, một bộ phân loại Cây Quyết định, một bộ phân loại Hồi quy Logistic,...). Thậm chí sẽ còn tốt hơn nếu chúng được huấn luyện trên các mẫu huấn luyện khác nhau (đây là ý tưởng đằng sau các phương pháp bagging và pasting ensemble), còn không phương pháp này vẫn tỏ ra hiệu quả miễn là các mô hình đều rất khác nhau.

2. Một bộ phân loại biểu quyết cứng sẽ chỉ đếm kết quả của các bộ phân loại và lấy ra lớp nhiều nhất. Một bộ phân loại biểu quyết mềm sẽ tính xác suất ước tính trung bình của các lớp và chọn ra lớp có xác suất cao nhất. Phương pháp này cho các dự đoán với độ tin cậy cao có nhiều sức nặng hơn và thường sẽ hoạt động tốt hơn, nhưng nó chỉ có tác dụng khi bạn có

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

thể ước lượng được xác suất các lớp (ví dụ, bạn sẽ phải đặt `probability=True` với SVM trong Scikit–Learn).

3. Ta hoàn toàn có thể tăng tốc huấn luyện bagging ensemble bằng cách phân bố trên nhiều máy chủ khác nhau, vì mỗi bộ dự đoán trong ensemble là độc lập với số còn lại. Điều tương tự cũng đúng đối với pasting ensemble và Rừng Ngẫu nhiên, với cùng một lý do. Tuy nhiên, một bộ dự đoán trong boosting ensemble được xây dựng dựa trên bộ dự đoán trước nó, vậy nên huấn luyện về căn bản là tuần tự và bạn sẽ không đạt được gì khi phân bố chúng trên các máy chủ khác nhau. Với stacking ensemble, tất cả các bộ dự đoán trong một tầng đều độc lập với nhau, nên chúng có thể được huấn luyện song song trên nhiều máy chủ. Tuy nhiên, việc huấn luyện các bộ dự đoán trong một tầng sẽ cần tầng trước huấn luyện xong trước khi bắt đầu.

4. Với đánh giá out-of-bag, mỗi bộ dự đoán trong bagging ensemble sẽ được đánh giá bằng các mẫu ngoài tập huấn luyện (các mẫu được giữ lại). Điều này cho phép ta có một đánh giá tương đối công bằng mà không cần thêm tập kiểm định. Vậy nên, bạn càng có nhiều mẫu để huấn luyện, ensemble của bạn càng tốt.

5. Khi xây dựng một cây trong Rừng Ngẫu nhiên, chỉ một tập con ngẫu nhiên các đặc trưng được sử dụng để chia tách tại các nút. Điều này cũng đúng với Cây Siêu Ngẫu nhiên, nhưng có thêm một bước: thay vì tìm kiếm mức ngưỡng tốt nhất, như Cây Quyết định thường làm, Cây Siêu Ngẫu nhiên sử dụng mức ngưỡng ngẫu nhiên cho mỗi đặc trưng. Tính ngẫu nhiên cộng thêm này có tác dụng như một dạng điều chuẩn: nếu Rừng Ngẫu nhiên quá khớp tập huấn luyện, Cây Siêu Ngẫu nhiên sẽ hoạt động tốt hơn. Hơn nữa, vì Cây Siêu Ngẫu nhiên không tìm mức ngưỡng tốt nhất, huấn luyện chúng sẽ nhanh hơn so với Rừng Ngẫu nhiên. Tuy nhiên, không có sự khác biệt nào về thời gian giữa Cây Siêu Ngẫu nhiên và Rừng Ngẫu nhiên khi thực hiện dự đoán.

6. Nếu AdaBoost ensemble dưới khớp dữ liệu huấn luyện, bạn có thể thử tăng số lượng bộ dự đoán hoặc giảm điều chuẩn của các bộ dữ đoán cơ sở. Ngoài ra, bạn cũng có thể thử tăng nhẹ tốc độ học.

7. Nếu Gradient Boosting ensemble quá khớp tập huấn luyện, bạn có thể thử giảm tốc độ học. Bạn cũng có thể thử áp dụng dừng sớm để tìm số lượng bộ dự đoán phù hợp (vì có thể bạn đang có quá nhiều chúng).

Đối với đáp án của bài tập 8 và 9, hãy tham khảo các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 8: Giảm Chiều

1. Động lực chính của các phép giảm chiều là:

   - Tăng tốc thuật toán huấn luyện (trong một số trường hợp nó còn có thể loại bỏ nhiễu và các đặc trưng dư thừa, khiến thuật toán huấn luyện thậm chí còn hoạt động tốt hơn nữa)
   - Trực quan hóa dữ liệu và nhận những thông tin hữu ích về các đặc trưng quan trọng nhất
   - Tiết kiệm không gian lưu trữ (nén)

   Các hạn chế chính là:

   - Một phần thông tin sẽ bị mất, có khả năng khiến thuật toán huấn luyện giảm chất lượng.
   - Có thể sẽ cần nhiều tài nguyên tính toán.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- Khiến hệ thống pipeline Học Máy trở nên phức tạp hơn một chút.
- Các đặc trưng sau biến đổi có thể khá khó để giải nghĩa.

2. Lời nguyền chiều đề cập về những vấn đề tuy không xuất hiện ở không gian ít chiều nhưng sẽ hiện diện khi ở không gian nhiều chiều. Trong Học Máy, các vector được lấy mẫu ngẫu nhiên trong không gian nhiều chiều thường sẽ rất thưa, tăng nguy cơ quá khớp dữ liệu đồng thời khiến việc nhận biết các khuôn mẫu trở nên cực kỳ khó nếu không có thật nhiều dữ liệu huấn luyện.

3. Một khi tập dữ liệu đã được giảm chiều bằng một trong những kỹ thuật giảm chiều đã thảo luận, việc khôi phục lại dữ liệu một cách toàn diện là gần như không thể bởi vị một vài thông tin đã bị mất trong quá trình giảm chiều. Hơn nữa, trong khi một vài thuật toán (như PCA) có phương thức biến đổi ngược rất đơn giản có khả năng khôi phục tập dữ liệu gần giống với ban đầu, những thuật toán khác (như là t-SNE) thì không.

4. PCA có thể được sử dụng để giảm đáng kể số chiều của hầu hết các tập dữ liệu, kể cả dữ liệu cực kỳ phi tuyến tính, bởi vì ít nhất nó cũng có thể loại bỏ các chiều vô dụng. Tuy nhiên, nếu không thực sự có chiều thừa thãi nào – như trong tập dữ liệu Swiss roll – thực hiện giảm chiều với PCA sẽ dẫn đến việc mất quá nhiều thông tin. Điều ta muốn làm là trải Swiss roll ra, chứ không phải đập bẹp nó.

5. Vấn đề này phụ thuộc vào tập dữ liệu. Hãy thử xét trên hai ví dụ trái ngược nhau. Ví dụ đầu tiên, giả sử tập dữ liệu bao gồm các điểm gần như thẳng hàng. Trong trường hợp này, PCA có thể giảm xuống chỉ một chiều dữ liệu trong khi 95% phương sai vẫn được bảo toàn. Bây giờ hãy hình dung một tập dữ liệu với các điểm hoàn toàn ngẫu nhiên, trải khắp 1,000 chiều. Trong trường hợp này, cần ít nhất 950 chiều để bảo toàn 95% phương sai. Vậy nên câu trả lời là, điều đó phụ thuộc vào tập dữ liệu, và đáp án có thể là bất cứ số nguyên nào nằm trong khoảng từ 1 đến 950. Vẽ phương sai được giải thích như một hàm của số chiều là một cách để biết về số chiều thực chất trong tập dữ liệu.

6. PCA thông thường là lựa chọn mặc định, nhưng nó sẽ chỉ hoạt động nếu tập dữ liệu đủ nhỏ để nạp vào bộ nhớ. PCA Gia tăng phù hợp cho các tập huấn luyện lớn không thể chứa vừa trong bộ nhớ, nhưng nó sẽ chậm hơn so với PCA, vậy nên đối với những tập dữ liệu kích thước vừa với bộ nhớ, bạn nên ưu tiên PCA thông thường. PCA Gia tăng cũng hữu ích khi xử lý các tác vụ trực tuyến khi ta cần áp dụng PCA mỗi khi có được mẫu mới. PCA Ngẫu nhiên sẽ hữu ích khi bạn muốn giảm đáng kể số chiều và khi tập dữ liệu đủ chứa trong bộ nhớ; lúc này, PCA Ngẫu nhiên sẽ nhanh hơn nhiều so với PCA thông thường. Cuối cùng, PCA Hạt nhân sẽ phù hợp cho các tập dữ liệu phi tuyến.

7. Một cách dễ hiểu, một thuật toán giảm chiều tốt sẽ loại bỏ được một số lượng lớn chiều nhưng làm mất đi rất ít thông tin của tập dữ liệu. Một cách đo lường chất lượng là thực hiện phép biến đổi ngược và tính lỗi khôi phục. Tuy nhiên, không phải tất cả các thuật toán giảm chiều đều tồn tại phép biến đổi ngược. Tuy nhiên, nếu ta sử dụng một phép giảm chiều trước khi áp dụng một thuật toán Học Máy khác (chẳng hạn như một bộ phân loại Rừng Ngẫu nhiên), ta có thể đơn giản đo chất lượng của mô hình thứ hai; nếu kỹ thuật giảm chiều không làm mất quá nhiều thông tin, thì thuật toán Học Máy sẽ hoạt động với chất lượng gần giống với khi học trên tập dữ liệu ban đầu.

8. Sử dụng hai thuật toán giảm chiều khác nhau liên tiếp nhau là một việc hoàn toàn hợp lý. Một ví dụ tiêu biểu là sử dụng PCA để nhanh chóng loại bỏ đi những đặc trưng dư thừa, sau đó sử dụng một kỹ thuật giảm chiều chậm hơn, như LLE. Quá trình hai bước này khả năng cao sẽ có kết quả gần giống với việc chỉ áp dụng LLE đơn thuần, nhưng trong một khoảng thời gian ngắn hơn nhiều.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Đối với đáp án của bài tập 9 và 10, hãy tham khảo các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Lời giải Chương 9: Các kỹ thuật Học Không giám sát

1. Trong Học Máy, phân cụm là một tác vụ không giám sát gom nhóm các mẫu tương đồng lại với nhau. Cách tính độ tương đồng phụ thuộc vào vấn đề ta muốn giải quyết: ví dụ, trong một vài trường hợp, hai mẫu có khoảng cách gần nhau được cho là giống nhau, trong khi một vài trường hợp khác cho rằng dù chúng đang ở cách xa nhau, miễn chúng thuộc về một nhóm mẫu mật độ cao, thì vẫn được coi là giống nhau. Các thuật toán gom cụm phổ biến gồm K–điểm trung bình, DBSCAN, Tích cụm, BIRCH, Dịch–Trung bình, Lan truyền Ái lực và Phân cụm Phổ.

2. Các ứng dụng chính của phân cụm gồm phân tích dữ liệu, phân khúc khách hàng, hệ thống đề xuất, hệ thống tìm kiếm, phân vùng ảnh, học bán giám sát, giảm chiều, phát hiện bất thường và phát hiện tính mới.

3. Luật khuỷu tay là một kỹ thuật đơn giản để chọn số cụm khi sử dụng K–điểm trung bình: chỉ cần vẽ đồ thị inertia (khoảng cách trung bình bình phương của mỗi mẫu đến tâm cụm gần nhất) theo số cụm và tìm điểm mà tại đó inertia giảm chậm lại (phần "khuỷu tay"). Con số này thông thường sẽ gần với số cụm tối ưu. Một các tiếp cận khác là vẽ điểm silhouette như một hàm của số cụm. Thường sẽ có một đỉnh và số cụm tối ưu sẽ nằm quanh đó. Điểm silhouette là tham số silhouette trung bình của tất cả các mẫu. Tham số này biến thiên từ +1 cho các mẫu hoàn toàn nằm trong cụm và xa các cụm khác, đến –1 cho các mẫu rất gần với một cụm không phải nó thuộc về. Bạn cũng có thể vẽ các biểu đồ silhouette và thực hiện phân tích cặn kẽ hơn.

4. Việc gán nhãn một tập dữ liệu tốn rất nhiều chi phí và thời gian. Vì thế, thường ta sẽ có rất nhiều mẫu không nhãn và một số ít mẫu có nhãn. Lan truyền nhãn là một kỹ thuật sao chép một vài (hoặc tất cả) các nhãn của những mẫu đã được gán nhãn sang những mẫu không nhãn gần nó. Phương pháp này có thể tăng thêm đáng kể số lượng mẫu có nhãn, khiến thuật toán học có giám sát hoạt động tốt hơn (đây cũng có thể coi là một dạng của học bán giám sát). Một cách làm của phương pháp này là sử dụng thuật toán phân cụm, như K–điểm trung bình, trên toàn bộ tập dữ liệu, rồi với mỗi cụm tìm ra nhãn xuất hiện nhiều nhất hoặc nhãn của mẫu có tính đặc trưng của cụm (tức mẫu gần tâm cụm nhất) và lan truyền chúng cho những mẫu không nhãn trong cùm một cụm.

5. K–điểm trung bình và BIRCH mở rộng tốt cho các tập dữ liệu lớn. DBSCAN và Dịch–Trung bình tìm các vùng có mật độ cao.

6. Học chủ động hữu dụng khi ta có một lượng lớn các mẫu không nhãn vì việc đánh nhãn rất tốn chi phí. Trong trường hợp này (mà thường sẽ rất phổ biến), thay vì lựa chọn ngẫu nhiên các mẫu để gán nhãn, sẽ thích hợp hơn nếu ta sử dụng học chủ động. Trong học chủ động, các chuyên gia sẽ tương tác với thuật toán học, cung cấp nhãn cho một vài mẫu cụ thể khi thuật toán cần. Một hướng tiếp cận phổ biến là lấy mẫu bất định (xem phần mô tả tại Học Chủ động).

7. Nhiều người sử dụng cụm từ *anomaly detection* (*phát hiện bất thường*) và *novelty detection* (*phát hiện tính mới*) với chung một nghĩa, nhưng chúng thực ra không phải thế. Trong phát hiện bất thường, thuật toán được huấn luyện trên một tập dữ liệu có thể có ngoại lai, và mục tiêu của nó thường là phát hiện những ngoại lai (trong tập huấn luyện) đó, cũng như các ngoại lai mới. Trong phát hiện tính mới, thuật toán được huấn luyện trên một tập dữ

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

liệu được ngầm định là "sạch" và mục tiêu là phát hiện những điểm mới (novelty) chỉ có trên các mẫu mới. Một vài thuật toán hoạt động tốt cho tác vụ phát hiện bất thường (cụ thể như Rừng Cô lập), một số lại hoạt động tốt trên tác vụ phát hiện tính mới (như SVM một lớp chẳng hạn).

8. Một Mô hình hỗn hợp Gauss (GMM) là một mô hình xác suất với giả định các mẫu được sinh ra bởi một hỗn hợp các phân phối Gauss khác nhau mà các tham số của những phân phối này là chưa biết. Nói cách khác, mô hình cho rằng dữ liệu được nhóm vào một số lượng cụm nhất định, có dạng ellipsoid (nhưng dạng, kích thước, góc và mật độ có thể khác nhau giữa các cụm) mà ta không biết đâu là cụm mà mỗi mẫu thuộc về. Mô hình này hoạt động tốt trong các tác vụ ước lượng mật độ, phân cụm và phát hiện bất thường.

9. Một cách để tìm số lượng cụm phù hợp khi sử dụng mô hình hỗn hợp Gauss là vẽ đồ thị tiêu chí thông tin Bayes (BIC) hoặc tiêu chí thông tin Akaike theo số cụm, rồi chọn số lượng cụm có BIC hoặc AIC nhỏ nhất. Một kỹ thuật khác là sử dụng mô hình hỗn hợp Bayes Gauss, giúp chọn số lượng cụm một cách tự động.

Đối với đáp án của bài tập 10 và 13, hãy tham khảo các Jupyter Notebook tại *https://github.com/mlbvn/handson-ml2-vn*.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phụ lục B

Danh mục Công việc trong Dự án Học Máy

Danh sách các công việc này có thể là kim chỉ nam trong các dự án Học Máy của bạn. Nó bao gồm tám bước chính sau:

1. Định hình bài toán và xem xét bức tranh tổng thể.
2. Thu thập dữ liệu.
3. Khám phá dữ liệu để có được thông tin chi tiết.
4. Chuẩn bị dữ liệu để tạo ra những các dữ liệu đầu vào phù hợp cho các thuật toán Học Máy.
5. Khám phá các mô hình Học Máy khác nhau và chọn ra những mô hình tốt nhất.
6. Tinh chỉnh các mô hình và kết hợp chúng lại thành một giải pháp hoàn hảo hơn.
7. Trình bày giải pháp của bạn.
8. Khởi chạy, giám sát, và bảo trì hệ thống của bạn.

Tất nhiên là bạn có thể tự do điều chỉnh danh sách trên cho phù hợp với nhu cầu cụ thể.

Định hình Bài toán và Xem xét Bức tranh Tổng thể

1. Xác định mục tiêu theo phương diện kinh doanh.
2. Giải pháp của bạn sẽ được sử dụng như thế nào?
3. Các giải pháp hiện tại (nếu có) là gì?
4. Bạn sẽ định hình bài toán này như thế nào (có giám sát/không giám sát, trực tuyến/ngoại tuyến, v.v.)?
5. Chất lượng của giải pháp sẽ được đo lường như thế nào?
6. Thước đo chất lượng này có phù hợp với mục tiêu kinh doanh không?
7. Chất lượng mô hình tối thiểu cần thiết để đạt được mục tiêu kinh doanh là bao nhiêu?

8. Các bài toán tương tự là gì? Bạn có thể sử dụng lại kinh nghiệm hoặc công cụ có sẵn được không?
9. Yếu tố kiến thức chuyên môn có cần thiết không?
10. Bạn sẽ giải quyết bài toán theo cách thủ công như thế nào?
11. Liệt kê những giả định mà bạn (hoặc những người khác) đã đưa ra cho đến thời điểm hiện tại.
12. Kiểm chứng các giả định nếu có thể.

Thu thập Dữ liệu

Lưu ý: hãy tự động hóa càng nhiều càng tốt để có thể thu thập được dữ liệu mới một cách dễ dàng.

1. Liệt kê loại dữ liệu và số lượng mà bạn cần.
2. Tìm và ghi lại nơi bạn có thể thu thập được dữ liệu đó.
3. Kiểm tra xem dữ liệu sẽ chiếm bao nhiêu dung lượng.
4. Kiểm tra các nghĩa vụ pháp lý và xin quyền sử dụng nếu cần.
5. Xin quyền truy cập.
6. Tạo một môi trường làm việc (có đủ dung lượng lưu trữ).
7. Thu thập dữ liệu.
8. Chuyển đổi dữ liệu thành định dạng mà bạn có thể dễ dàng thao tác (mà không làm thay đổi thông tin dữ liệu).
9. Đảm bảo các thông tin nhạy cảm phải được bảo vệ hoặc xóa bỏ (ví dụ, dạng ẩn danh).
10. Kiểm tra kích thước và dạng dữ liệu (dạng chuỗi thời gian, mẫu, không gian địa lý, v.v.).
11. Lấy mẫu để tạo tập dữ liệu kiểm tra, để nó sang một bên và không được đụng vào (không được xem lén dữ liệu!).

Khám phá Dữ liệu

Lưu ý: hãy cố gắng nhờ một chuyên gia trong ngành để hiểu rõ hơn về dữ liệu khi thực hiện các bước này.

1. Tạo một bản sao dữ liệu dành cho việc khám phá (lấy mẫu với kích thước nhỏ hơn để có thể xử lý dễ dàng nếu cần).
2. Tạo một Jupyter notebook để ghi lại quá trình khám phá dữ liệu.
3. Nghiên cứu từng thuộc tính và đặc điểm của nó:
   - Tên

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

- Dạng dữ liệu (dạng hạng mục, dạng số nguyên/số thực, dạng bị chặn/không bị chặn, dạng văn bản, dạng có cấu trúc, v.v.)
- Phần trăm của những giá trị bị thiếu
- Độ nhiễu và dạng nhiễu (ngẫu nhiên, ngoại lai, lỗi làm tròn số, v.v.)
- Tính hữu ích cho tác vụ
- Dạng phân phối của dữ liệu (Gauss, đều, logarit, v.v.)

4. Đối với các tác vụ học có giám sát, hãy xác định (các) thuộc tính mục tiêu.
5. Trực quan hóa dữ liệu.
6. Nghiên cứu mối tương quan giữa các thuộc tính.
7. Nghiên cứu cách bạn sẽ giải quyết bài toán theo cách thủ công.
8. Xác định các phép biến đổi tiềm năng mà bạn có thể áp dụng.
9. Xác định dữ liệu bổ sung hữu ích (quay lại bước Thu thập Dữ liệu).
10. Ghi lại những gì bạn đã học được.

Chuẩn bị Dữ liệu

Lưu ý:

- Làm việc trên các bản sao của dữ liệu thay vì tập dữ liệu gốc (giữ nguyên tập dữ liệu gốc).
- Viết hàm cho tất cả các phép biến đổi dữ liệu, vì năm lý do sau:
  - Dễ dàng hơn trong việc chuẩn bị dữ liệu trong lần tiếp theo khi bạn có một tập dữ liệu mới
  - Có thể sử dụng các phép biến đổi này trong những dự án tương lai
  - Làm sạch và chuẩn bị tập kiểm tra
  - Làm sạch và chuẩn bị các mẫu dữ liệu mới khi giải pháp được triển khai
  - Dễ dàng để coi các lựa chọn trong việc chuẩn bị dữ liệu như các siêu tham số

1. Làm sạch dữ liệu
   - Sửa hoặc loại bỏ các điểm ngoại lai (không bắt buộc).
   - Điền vào các giá trị còn thiếu (ví dụ với 0, trung bình, trung vị, v.v.) hoặc loại bỏ hàng (hoặc cột) chứa các giá trị đó.
2. Lựa chọn đặc trưng (không bắt buộc):
   - Loại bỏ các thuộc tính không hữu ích cho tác vụ.
3. Thiết kế đặc trưng, nếu thích hợp:
   - Biến các đặc trưng liên tục thành rời rạc.
   - Phân tách các đặc trưng (ví dụ như hạng mục, thời gian, v.v..)
   - Thêm các phép biến đổi đặc trưng tiềm năng (ví dụ như $\log(x)$, $\sqrt(x)$, x^2, v.v..)
   - Tổng hợp các đặc trưng thành đặc trưng mới tiềm năng.
4. Co giãn đặc trưng:
   - Chuẩn hóa hoặc chuẩn tắc hóa các đặc trưng.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Rút gọn Danh sách các Mô hình Tiềm năng

Lưu ý:

- Nếu tập dữ liệu lớn, bạn có thể lấy mẫu các tập huấn luyện nhỏ hơn để huấn luyện nhiều mô hình khác nhau trong một khoảng thời gian hợp lý (lưu ý rằng phương pháp này không hiệu quả với những mô hình phức tạp như các mạng nơ-ron lớn hoặc Rừng Ngẫu nhiên).
- Một lần nữa, hãy cố gắng tự động hóa các bước này nhiều nhất có thể.

1. Huấn luyện nhiều loại mô hình đơn giản khác nhau (ví dụ như tuyến tính, naive Bayes, SVM, Rừng Ngẫu nhiên, Mạng nơ-ron, v.v..) sử dụng các tham số tiêu chuẩn.
2. Đo lường và so sánh chất lượng của chúng.
   - Với mỗi mô hình, sử dụng kiểm định N-fold, tính trung bình và độ lệch chuẩn của chất lượng mô hình trên N fold.
3. Phân tích những biến quan trọng nhất của từng thuật toán.
4. Phân tích những loại lỗi mà các mô hình gặp phải.
   - Con người sẽ sử dụng dữ liệu nào để tránh những lỗi này?
5. Thực hiện nhanh một lượt lựa chọn và thiết kế đặc trưng.
6. Thực hiện nhanh một hoặc hai lượt cả năm bước ở trên.
7. Tạo danh sách rút gọn từ ba tới năm mô hình có tiềm năng nhất, ưu tiên các mô hình có các loại lỗi khác nhau.

Tinh chỉnh Hệ thống

Lưu ý:

- Bạn nên sử dụng càng nhiều dữ liệu càng tốt tại bước này, đặc biệt là trong giai đoạn cuối của quá trình tinh chỉnh.
- Như thường lệ, cố gắng tự động hóa càng nhiều càng tốt.

1. Tinh chỉnh các siêu tham số sử dụng kiểm định chéo:
   - Coi các lựa chọn biến đổi dữ liệu như các siêu tham số, đặc biệt là khi bạn không chắc chắn về cách lựa chọn chúng (ví dụ nếu bạn phân vân trong việc thay thế giá trị thiếu bằng 0 hay bằng giá trị trung vị, hay chỉ đơn giản là bỏ luôn các hàng đó).
   - Trừ khi có rất ít các giá trị siêu tham số để thử nghiệm, hãy ưu tiên tìm kiếm ngẫu nhiên thay vì tìm kiếm dạng lưới. Nếu quá trình huấn luyện tốn nhiều thời gian, bạn có thể sử dụng phương pháp tối ưu hóa Bayes (ví dụ như dùng quá trình Gauss tiên nghiệm (Gaussian process prior), như được mô tả bởi Jasper Snoek et al.).[1]
2. Thử các phương pháp Ensemble. Kết hợp các mô hình tốt nhất thường sẽ cho kết quả tốt hơn so với từng mô hình riêng biệt.

[1] Jasper Snoek et al., "Practical Bayesian Optimization of Machine Learning Algorithms," *Proceedings of the 25th International Conference on Neural Information Processing Systems* 2 (2012): 2951–2959.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

3. Một khi bạn đã tự tin với mô hình cuối cùng, hãy đo lường chất lượng trên tập kiểm tra để ước lượng lỗi tổng quát hóa.

Lưu ý

Đừng tinh chỉnh mô hình sau khi đo lường lỗi tổng quát hóa, bạn sẽ bắt đầu quá khớp tập kiểm tra.

Trình bày Giải pháp

1. Ghi chép lại những gì bạn đã làm.
2. Thực hiện một bài thuyết trình hay.
   - Hãy đảm bảo rằng bạn sẽ nhấn mạnh bức tranh tổng thể trước.
3. Giải thích tại sao giải pháp của bạn đạt được mục tiêu kinh doanh.
4. Đừng quên trình bày những điểm thú vị mà bạn tìm được trong quá trình phát triển hệ thống.
   - Mô tả những thứ hoạt động và không hoạt động.
   - Liệt kê các giả định cũng như các hạn chế của hệ thống.
5. Đảm bảo những phát hiện chính của bạn được truyền đạt bằng trực quan đẹp mắt hoặc mệnh đề dễ nhớ (ví dụ như "thu nhập trung bình là nhân tố quan trọng nhất để dự đoán giá nhà ở").

Triển khai!

1. Chuẩn bị để triển khai giải pháp (đưa dữ liệu đầu vào vào hệ thống, viết unit test, v.v..).
2. Viết mã giám sát để kiểm tra định kỳ chất lượng của hệ thống trong quá trình hoạt động và kích hoạt cảnh bảo khi chất lượng đi xuống.
   - Cẩn thận với sự xuống cấp chậm của hệ thống: các mô hình có xu hướng "mục nát" khi dữ liệu thay đổi.
   - Việc đo lường chất lượng có thể yêu cầu nhân công con người (ví dụ như thông qua các dịch vụ crowdsourcing).
   - Đồng thời theo dõi chất lượng của đầu vào (ví dụ cảm biến bị trục trặc gửi đi các giá trị ngẫu nhiên, hoặc đầu ra của một nhóm khác không được cập nhật theo thời gian thực). Điều này đặc biệt quan trọng với các hệ thống học trực tuyến.
3. Huấn luyện lại các mô hình theo định kỳ với dữ liệu mới (tự động hóa nhiều nhất có thể).

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Phụ lục C

Bài toán Đối ngẫu SVM

Để hiểu về *tính đối ngẫu* (*duality*), đầu tiên ta cần hiểu về phương pháp *nhân tử Lagrange*. Ý tưởng cơ bản là biến đổi mục tiêu tối ưu có ràng buộc thành không ràng buộc, bằng cách chuyển các ràng buộc vào hàm mục tiêu. Hãy xét một ví dụ đơn giản. Giả sử ta muốn tìm các giá trị x and y để tối thiểu hóa hàm $f(x, y) = x^2 + 2y$ và thỏa mãn *đẳng thức*: $3x + 2y + 1 = 0$. Để sử dụng phương pháp nhân tử Lagrange, ta định nghĩa một hàm mới được gọi là *Lagrangian* (hoặc *hàm Lagrange*): $g(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$. Ta lấy hàm mục tiêu trừ đi tích của mỗi ràng buộc (trong trường hợp này là một) với một biến số mới được gọi là nhân tử Lagrange.

Joseph-Louis Lagrange chỉ ra rằng nếu $(\hat{x}, \hat{y})$ là nghiệm của bài toán tối ưu có ràng buộc, thì phải tồn tại một $\hat{\alpha}$ mà $(\hat{x}, \hat{y}, \hat{\alpha})$ là một *điểm dừng* của Lagrangian (điểm dừng là điểm mà tại đó tất cả các đạo hàm riêng đều bằng 0). Nói cách khác, ta có thể tính các đạo hàm riêng của $g(x,\ y,\ \alpha)$ theo từng biến độc lập x, y, và α; từ đó tìm ra những điểm mà tại đó các đạo hàm đều bằng 0. Nghiệm của bài toàn tối ưu hóa có ràng buộc (nếu có tồn tại) phải nằm trong số các điểm dừng này.

Trong ví dụ này ta có các đạo hàm riêng: $\begin{cases} \frac{\partial}{\partial x} g(x, y, \alpha) = 2x - 3\alpha \\ \frac{\partial}{\partial y} g(x, y, \alpha) = 2 - 2\alpha \\ \frac{\partial}{\partial \alpha} g(x, y, \alpha) = -3x - 2y - 1 \end{cases}$

Để tất cả các đạo hàm riêng đều bằng 0, ta cần giải phương trình $2\hat{x} - 3\hat{\alpha} = 2 - 2\hat{\alpha} = -3\hat{x} - 2\hat{y} - 1 = 0$, và ta dễ dàng tìm được $\hat{x} = \frac{3}{2}$, $\hat{y} = -\frac{11}{4}$, và $\hat{\alpha} = 1$. Đây là điểm dừng duy nhất, và bởi vì nó thỏa mãn ràng buộc cho trước, điểm này chính là nghiệm của bài toán tối ưu có ràng buộc.

Tuy nhiên phương pháp này chỉ áp dụng được cho các ràng buộc ở dạng đẳng thức. May mắn thay, dưới một số điều kiện chính quy (mà mục tiêu của SVM thỏa mãn), phương pháp này cũng có thể được tổng quát hóa cho *các ràng buộc bất đẳng thức* (ví dụ như $3x + 2y + 1 \geq 0$). Hàm *Lagrangian tổng quát* cho bài toán biên cứng được biểu diễn bởi Phương trình C.1, trong đó các biến $\alpha^{(i)}$ được gọi là nhân tử *Karush–Kuhn–Tucker* (KKT), và chúng phải lớn hơn hoặc bằng 0.

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} \left(t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - 1 \right)$$
$$\text{với} \quad \alpha^{(i)} \geq 0, \quad i = 1, 2, \cdots, m$$

Phương trình C.1. Hàm tổng quát Lagrangian cho bài toán biên cứng

Cũng giống như phương pháp nhân tử Lagrange, ta có thể tính các đạo hàm riêng và định vị các điểm dừng. Nếu có một nghiệm, nó nhất thiết phải thuộc một trong các điểm dừng $(\hat{\mathbf{w}}, \hat{b}, \hat{\alpha})$ thỏa mãn các *điều kiện KKT* sau:

- Thỏa mãn các ràng buộc của bài toán: $t^{(i)}(\hat{\mathbf{w}}^\top \mathbf{x}^{(i)} + \hat{b}) \geq 1$ với $i = 1, 2, \ldots, m$.
- Thỏa mãn điều kiện $\hat{\alpha}^{(i)} \geq 0$ với $i = 1, 2, \ldots, m$.
- Đẳng thức $\hat{\alpha}^{(i)} = 0$ hoặc ràng buộc thứ i phải là một *ràng buộc chủ động*, nghĩa là phải thỏa mãn đẳng thức: $t^{(i)}(\hat{\mathbf{w}}^\top \mathbf{x}^{(i)} + \hat{b}) = 1$. Điều kiện này được gọi là điều kiện *bù bổ sung* (*complementary slackness*). Nó ám chỉ rằng hoặc $\hat{\alpha}^{(i)} = 0$ hoặc mẫu thứ i nằm trên đường ranh giới (là một vector hỗ trợ).

Lưu ý rằng các điều kiện KKT là những điều kiện cần để một điểm dừng là nghiệm của bài toán tối ưu ràng buộc. Với một số điều kiện nhất định, chúng cũng là điều kiện đủ. May mắn thay, bài toán tối ưu SVM lại thỏa mãn các điều kiện này, và vì thế bất kỳ điểm dừng nào thỏa mãn các điều kiện KKT đều chắc chắn là một nghiệm của bài toán tối ưu ràng buộc.

Chúng ta có thể tính các đạo hàm riêng của hàm tổng quát Lagrangian theo $\mathbf{w}$ và b như trong Phương trình C.2.

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b, \alpha) = \mathbf{w} - \sum_{i=1}^{m} \alpha^{(i)} t^{(i)} \mathbf{x}^{(i)}$$
$$\frac{\partial}{\partial b} \mathcal{L}(\mathbf{w}, b, \alpha) = -\sum_{i=1}^{m} \alpha^{(i)} t^{(i)}$$

Phương trình C.2. Các đạo hàm riêng của hàm tổng quát Lagrangian

Khi các đạo hàm riêng này đều bằng 0, ta có Phương trình C.3.

$$\hat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$
$$\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} = 0$$

Phương trình C.3. Các đặc tính của điểm dừng

Nếu ta thay các kết quả này vào định nghĩa của hàm tổng quát Lagrangian, ta sẽ có Phương trình C.4.

$$\mathcal{L}(\hat{\mathbf{w}}, \hat{b}, \alpha) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} {\mathbf{x}^{(i)}}^\top \mathbf{x}^{(j)} - \sum_{i=1}^{m} \alpha^{(i)}$$
$$\text{với} \quad \alpha^{(i)} \geq 0,\ i = 1, 2, \cdots, m$$

Phương trình C.4. Dạng đối ngẫu của bài toán SVM

Mục tiêu bây giờ là tìm vector $\hat{\alpha}$ sao cho hàm này nhỏ nhất, với điều kiện là $\hat{\alpha}^{(i)} \geq 0$ trên tất cả các mẫu. Bài toán tối ưu có rằng buộc này chính là bài toán đối ngẫu mà chúng ta đang tìm kiếm.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Một khi chúng ta tìm thấy $\hat{\alpha}$ tối ưu, ta có thể tính $\hat{\mathbf{w}}$ bằng công thức đầu tiên trong Phương trình C.3. Để tính $\hat{b}$, ta có thể dựa vào việc một vector hỗ trợ phải thỏa mãn $t^{(i)}(\hat{\mathbf{w}}^\top \mathbf{x}^{(i)} + \hat{b}) = 1$, vì vậy nếu mẫu thứ k là một vector hỗ trợ (tức $\hat{\alpha}^{(k)} > 0$), ta có thể sử dụng nó để tính $\hat{b} = t^{(k)} - \hat{\mathbf{w}}^\top \mathbf{x}^{(k)}$. Tuy nhiên, ta nên ưu tiên tính giá trị trung bình của tất cả các vector hỗ trợ để có được một giá trị ổn định và chính xác hơn, như trong Phương trình C.5.

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^\top \mathbf{x}^{(i)} \right)$$

Phương trình C.5. Ước lượng độ chệch sử dụng dạng đối ngẫu

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Tài liệu tham khảo

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

[1] *On Lines and Planes of Closest Fit to Systems of Points in Space*, K. Pearson, 1901

[2] *A Logical Calculus of Ideas Immanent in Nervous Activity*, W. McCulloch, W. Pitts, 1943

[3] *A Markovian Decision Process*, R. Bellman, 1957

[4] *Single Unit Activity in Striate Cortex of Unrestrained Cats*, D. Hubel, T. Wiesel, 1958

[5] *Receptive Fields of Single Neurones in the Cat's Striate Cortex*, D. Hubel, T. Wiesel, 1959

[6] *Some methods of speeding up the convergence of iteration methods*, B. Polyak, 1964

[7] *Receptive Fields and Functional Architecture of Monkey Striate Cortex*, D. Hubel, T. Wiesel, 1968

[8] *Perception in chess*, W. Chase, H. Simon, 1973

[9] *A method for unconstrained convex minimization problem with the rate of convergence* $O(1/k^2)$, Y. Nesterov, 1983

[10] *Learning Internal Representations by Error Propagation*, D. Rumelhart, G. Hinton, R. Williams, 1986

[11] *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*, R. Williams, 1992

[12] *Stacked Generalization*, D. Wolpert, 1992

[13] *Random Decision Forests*, Tin Kam Ho, 1995

[14] *Bagging Predictors*, L. Breiman, 1996

[15] *The Lack of A Priori Distinctions Between Learning Algorithms*, D. Wolpert, 1996

[16] *Long Short-Term Memory*, S. Hochreiter, J. Schmidhuber, 1997

[17] *Sequential Minimal Optimization (SMO)*, Platt, Microsoft Research, 1998

[18] *Gradient-Based Learning Applied to Document Recognition*, Y. LeCun et al., 1998

[19] *The random subspace method for constructing decision forests*, Tin Kam Ho, 1998

[20] *Kernel Principal Component Analysis*, B. Schölkopf, A. Smola, K. Müller, 1999

[21] *Pasting small votes for classification in large databases and on-line*, L. Breiman, 1999

[22] *Nonlinear Dimensionality Reduction by Locally Linear Embedding*, S. Roweis, L. Saul, 2000

[23] *Recurrent Nets that Time and Count*, F. Gers, J. Schmidhuber, 2000

[24] *Extremely Randomized Trees*, P. Geurts, D. Ernst, L. Wehenkel, 2005

[25] *On Contrastive Divergence Learning*, M. Á. Carreira-Perpiñán, G. Hinton, 2005

[26] *Primal-dual subgradient methods for convex problems*, Y. Nesterov, 2005

[27] *Greedy Layer-Wise Training of Deep Networks*, Y. Bengio et al., 2007

[28] *A Dual Coordinate Descent Method for Large-scale Linear SVM*, Lin et al., 2008

[29] *Extracting and Composing Robust Features with Denoising Autoencoders*, P. Vincent et al., 2008

[30] *Semantic Hashing*, R. Salakhutdinov, G. Hinton, 2008

[31] *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*, P. Vincent et al., 2010

[32] *Contractive Auto-Encoders: Explicit Invariance During Feature Extraction*, S. Rifai et al., 2011

[33] *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*, J. Masci et al., 2011

[34] *Ensembles on Random Patches*, G. Louppe, P. Geurts, 2012

[35] *ImageNet Classification with Deep Convolutional Neural Networks*, Alex Krizhevsky et al., 2012

[36] *Improving neural networks by preventing co-adaptation of feature detectors*, G. Hinton et al., 2012

[37] *Ad Click Prediction: a View from the Trenches*, H. McMahan et al., 2013

[38] *An Empirical Study of Learning Rates in Deep Neural Networks for Speech Recognition*, A. Senior et al., 2013

[39] *On the difficulty of training recurrent neural networks*, R. Pascanu et al., 2013

[40] *Playing Atari with Deep Reinforcement Learning*, V. Mnih et al., 2013

[41] *Auto-Encoding Variational Bayes*, D. Kingma, M. Welling, 2014

[42] *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, N. Srivastava et al., 2014

[43] *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*, K. Cho et al., 2014

[44] *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*, Haşim Sak et al., 2014

[45] *Neural Machine Translation by Jointly Learning to Align and Translate*, D. Bahdanau et al., 2014

[46] *Sequence to Sequence learning with Neural Networks*, I. Sutskever et al., 2014

[47] *Adam: A Method for Stochastic Optimization*, D. Kingma, J. Ba, 2015

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

[48] *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, S. Ioffe, C. Szegedy, 2015

[49] *CNN Based Hashing for Image Retrieval*, J. Gua, J. Li, 2015

[50] *Deep Residual Learning for Image Recognition*, K. He, 2015

[51] *Empirical Evaluation of Rectified Activations in Convolution Network*, Bing Xu et al., 2015

[52] *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, D. Clevert, T. Unterthiner, S. Hochreiter, 2015

[53] *Going Deeper with Convolutions*, Christian Szegedy et al., 2015

[54] *GSNs: Generative Stochastic Networks*, G. Alain et al., 2015

[55] *Human-level control through deep reinforcement learning*, V. Mnih et al., 2015

[56] *On Using Very Large Target Vocabulary for Neural Machine Translation*, S. Jean et al., 2015

[57] *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, K. Xu et al., 2015

[58] *Recurrent Neural Network Regularization*, W. Zaremba et al., 2015

[59] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, Google Research , 2015

[60] *Very Deep Convolutional Networks for Large-Scale Image Recognition*, K. Simonyan, A. Zisserman, 2015

[61] *Winner-Take-All Autoencoders*, A. Makhzani, B. Frey, 2015

[62] *Adversarial Autoencoders*, A. Makhzani et al., 2016

[63] *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*, Szegedy et al., 2016

[64] *Long Short-Term Memory-Networks for Machine Reading*, J. Cheng, 2016

[65] *Revisiting Distributed Synchronous SGD*, Jianmin Cheng et al., 2016

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Chỉ mục

A

accelerated K-Means – K-Điểm trung bình tăng tốc, 213
accuracy – độ chính xác
  đánh giá bằng kiểm định chéo, 78
  định nghĩa, 79
  ví dụ, 2
activation function – hàm kích hoạt
  Logistic (sigmoid), 126
active constraint – ràng buộc chủ động, 259
active learning – học chủ động, 223
AdaBoost, 174
Adaptive Boosting – Boosting Thích ứng, 174
affinity – ái lực, 207
affinity propagation – lan truyền ái lực, 227
agent – tác nhân, 12
agglomerative clustering – tích cụm, 226
ái lực – affinity, 207
Akaike information criterion (AIC) – tiêu chí thông tin Akaike, 234
algorithm – thuật toán
  anomaly detection – phát hiện bất thường, 240
  BIRCH algorithm – thuật toán BIRCH, 227
  CART training algorithm – thuật toán huấn luyện CART, 156, 158
  clustering algorithm – thuật toán phân cụm, 9
  Expectation-Maximization (EM) algorithm – thuật toán Kỳ vọng-Cực đại hóa, 230
  greedy algorithm – thuật toán tham lam, 158
  hierarchical clustering algorithm – thuật toán phân cụm phân cấp, 9
  Isolation Forest algorithm – thuật toán Rừng Cô lập, 240
  K-Means algorithm – thuật toán K-Điểm trung bình, 208
  Lloyd–Forgy algorithm – thuật toán Lloyd–Forgy, 208
  Mean-Shift algorithm – thuật toán Dịch-Trung bình, 227
  one-class SVM algorithm – thuật toán SVM một lớp, 240
  Randomized PCA – thuật toán PCA Ngẫu nhiên, 196
  supervised learning – học có giám sát, 7
  tầm quan trọng của dữ liệu so với, 21
  thuật toán isomap, 202
  unsupervised learning – học không giám sát, 8
  visualization algorithm – thuật toán trực quan hóa, 9
alpha channel – kênh alpha, 219
ánh xạ đặc trưng – feature map, 199
anomaly detection – phát hiện bất thường
  các thuật toán khác, 240
  dùng hỗn hợp Gauss, 232
  mục tiêu, 205
  dùng phân cụm, 207
  ví dụ, 10
area under the curve (AUC) – diện tích dưới đường cong, 87
association rule learning – học luật kết hợp, 11
attribute – thuộc tính, 7
average absolute deviation – trung bình độ lệch tuyệt đối, 36

B

bagging và pasting
  đánh giá out-of-bag, 170
  trong Scikit-Learn, 169
  tổng quan, 168
bài toán dự đoán – prediction problem, 7, 15, 165
bài toán đa hồi quy – multiple regression problem, 34
bài toán đối ngẫu – dual problem, 149
bài toán gốc – primal problem, 149
bài toán hồi quy – regression problem
  Decision Tree – Cây Quyết định, 160
  định nghĩa, 7

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

k-Nearest Neighbors regression – hồi quy k-Điểm Gần nhất, 20
Lasso Regression – Hồi quy Lasso, 122
Linear Regression – Hồi quy Tuyến tính, 99–104
Logistic Regression – Hồi quy Logistic, 126–134
multiple regression problem – bài toán đa hồi quy, 34
multivariate regression problem – bài toán hồi quy đa biến, 34
Polynomial Regression – Hồi quy Đa thức, 114
Ridge Regression – Hồi quy Ridge, 119
Softmax Regression – Hồi quy Softmax, 131–134
SVM regression – SVM hồi quy, 144
univariate regression problem – bài toán hồi quy đơn biến, 34
bài toán hồi quy đa biến – multivariate regression problem, 34
bài toán hồi quy đơn biến – univariate regression problem, 34
bài toán NP-đầy đủ – NP-Complete problem, 158
bài toán phân loại – classification problem
binary classifier – bộ phân loại nhị phân, 77
bộ phân loại AdaBoost, 174
bộ phân loại Extra-Trees, 172
hard margin classification – phân loại biên cứng, 137
large margin classification – phân loại biên lớn, 136
linear SVM classification – phân loại SVM tuyến tính, 136
multiclass classification – bài toán phân loại đa lớp, 89
multilabel classification – phân loại đa nhãn, 94
multioutput classification – phân loại đa đầu ra, 95
nonlinear SVM classification – phân loại SVM phi tuyến, 139–144
performance measure – phép đo chất lượng, 78–88
phân tích lỗi, 91
soft margin classification – phân loại biên mềm, 137
tập dữ liệu MNIST, 75
ví dụ, 7
bài toán phân loại đa lớp – multiclass classification, 89
bài toán Quy hoạch Toàn phương – Quadratic Programming (QP) problem, 148
Batch Gradient Descent – Hạ Gradient theo Batch, 107
batch learning – học theo batch, 13
Bayesian Gaussian Mixture model – mô hình hỗn hợp Bayes Gauss, 236
Bayesian information criterion (BIC) – tiêu chí thông tin Bayes, 234
Better Life Index, 16
bias term – hệ số điều chỉnh, 99
bias/variance trade-off – đánh đổi độ chệch/phương sai, 119
biến bù – slack variable, 148
biến đã quan sát – observed variable, 229
biến tiềm ẩn – latent variable, 229
biểu diễn one-hot – one-hot encoding, 59
biểu đồ tần suất – histogram, 43
biểu đồ tần suất nặng đuôi – tail-heavy histogram, 44
biểu quyết mềm – soft voting, 168
binary classifier – bộ phân loại nhị phân, 77
binary tree – cây nhị phân, 156
BIRCH algorithm – thuật toán BIRCH, 227
black box model – mô hình hộp đen, 157
black box stochastic variational inference (BBSVI) – suy luận biến phân hộp đen ngẫu nhiên, 238
blender – bộ kết hợp, 182
boosting
AdaBoost, 174
Gradient Boosting, 177
tổng quan, 174
boosting giả thuyết – hypothesis boosting, 174
Boosting Thích ứng – Adaptive Boosting, 174
bộ học mạnh – strong learners, 166
bộ học meta – meta learner, 182
bộ học yếu – weak learners, 166
bộ kết hợp – blender, 182
bộ lọc thư rác – spam filter, 1, 2
bộ phân loại biểu quyết cứng – hard voting classifier, 166
bộ phân loại biểu quyết theo đa số – majority-vote classifier, 166
bộ phân loại đa thức – multinomial classifier, 89
bộ phân loại Extra-Trees – Extra-Trees classifier,

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

172
bộ phân loại nhị phân – binary classifier, 77
bộ tối ưu – optimizer
Stochastic Gradient Descent (SGD) – Hạ Gradient Ngẫu nhiên, 78, 109
bước cực đại hóa – maximization step, 230
bước kỳ vọng – expectation step, 230

C

các điều kiện Mercer – Mercer's conditions, 151
calculus – giải tích, 99
California Housing Prices dataset – tập dữ liệu Giá nhà ở California, 32
CART training algorithm – thuật toán huấn luyện CART, 156, 158
categorical distribution – phân phối hạng mục, 229
căn bậc hai trung bình bình phương sai số – Root Mean Square Error (RMSE), 34, 106
cắt tỉa – pruning, 160
cận dưới bằng chứng – evidence lower bound (ELBO), 238
cây nhị phân – binary tree, 156
Cây Phân loại và Hồi quy (CART) – Classification and Regression Tree (CART), 156, 158
Cây Quyết định – Decision Tree
CART training algorithm – thuật toán huấn luyện CART, 158
computational complexity – độ phức tạp tính toán, 158
estimating class probabilities – ước lượng xác suất các lớp, 157
evaluating – đánh giá, 64
Gini impurity versus entropy – độ pha tạp Gini với entropy, 159
instability drawback – điểm yếu bất ổn, 162
lợi ích, 154
making prediction – đưa ra dự đoán, 155
regression task – tác vụ hồi quy, 160
regularization hyperparameters – các siêu tham số điều chuẩn, 159
training and visualizing – huấn luyện và trực quan hóa, 154
centroid – tâm cụm, 207
chỉ số F1 – F1 score, 82
chiến lược một-còn lại – one-versus-the-rest (OvR) strategy, 89
chiến lược một-một – one-versus-one (OvO) strategy, 89
chiến lược một-toàn bộ – one-versus-all (OvA) strategy, 89
chính sách – policy, 12
chi-squared test – phép kiểm định chi-bình phương, 160
chuẩn bị dữ liệu
bộ biến đổi tùy chỉnh, 60
feature scaling – co giãn đặc trưng, 61
làm sạch dữ liệu, 55
lợi ích của hàm, 55
pipeline biến đổi, 61
xử lý thuộc tính văn bản và hạng mục, 57
chuẩn Euclid – Euclidean norm, 36
chuẩn hóa – normalization, 61
chuẩn Manhattan – Manhattan norm, 36
chuẩn tắc hóa – standardization, 61
Classification and Regression Tree (CART) – Cây Phân loại và Hồi quy (CART), 156, 158
classification problem – bài toán phân loại
binary classifier – bộ phân loại nhị phân, 77
bộ phân loại AdaBoost, 174
bộ phân loại Extra-Trees, 172
hard margin classification – phân loại biên cứng, 137
large margin classification – phân loại biên lớn, 136
linear SVM classification – phân loại SVM tuyến tính, 136
multiclass classification – bài toán phân loại đa lớp, 89
multilabel classification – phân loại đa nhãn, 94
multioutput classification – phân loại đa đầu ra, 95
nonlinear SVM classification – phân loại SVM phi tuyến, 139–144
performance measure – phép đo chất lượng, 78–88
phân tích lỗi, 91
soft margin classification – phân loại biên mềm, 137
tập dữ liệu MNIST, 75
ví dụ, 7
closed-form solution – nghiệm dạng đóng, 100
clustering algorithm – thuật toán phân cụm

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

các thuật toán khác, 226
DBSCAN, 224
cho học bán giám sát, 221
K-Means – K-Điểm trung bình, 208–218
mục tiêu, 205
cho phân vùng ảnh, 207, 218
cho tiền xử lý, 220
tổng quan, 206
ứng dụng, 9, 206
Co giãn Đa chiều – Multidimensional Scaling (MDS), 202
co giãn đặc trưng – feature scaling, 61
co giãn min-max – min-max scaling, 61
color segmentation – phân vùng màu, 218
complementary slackness – bù bổ sung, 259
component – thành phần, 33
compressing – nén, 195
confusion matrix – ma trận nhầm lẫn, 80
constrained optimization – tối ưu có điều kiện, 148
convergence – tính hội tụ, 104
convex function – hàm lồi, 106
core instance – mẫu chủ chốt, 224
corpus development – làm giàu kho ngữ liệu, 21
correlation coefficient – hệ số tương quan, 51
cost function – hàm chi phí
cross-entropy loss (log loss) – mất mát entropy chéo (mất mát logarit), 132
hinge loss – mất mát hinge, 138, 152
mean absolute error (MAE) – trung bình sai số tuyệt đối, 36
mean squared error – trung bình bình phương sai số, 106
vai trò, 18
công cụ tìm kiếm – search engine, 207
cross-entropy loss (log loss) – mất mát entropy chéo (mất mát logarit), 132
cross-validation – kiểm định chéo, 28, 64, 78
curse of dimensionality – lời nguyền chiều, 187
customer segmentation – phân nhóm khách hàng, 207
cực tiểu – local minimum, 106

D

danh mục công việc cho dự án Học Máy, 33, 253
data – dữ liệu, *Xem thêm* chuẩn bị dữ liệu; trực quan hóa dữ liệu; dữ liệu huấn luyện
compression – nén, 195
data mismatch – dữ liệu không tương đồng, 28
decompressing – giải nén, 195
dữ liệu địa lý, 49
giảm chiều, 193
nguồn, 31
noisy data – dữ liệu chứa nhiễu, 17
phân tích bằng phân cụm, 207
preprocessing – tiền xử lý, 220
reconstruction error – lỗi khôi phục, 195
skewed dataset – tập dữ liệu lệch, 79
tải, 40
tầm quan trọng so với thuật toán, 21
tập dữ liệu Giá nhà ở California, 32
tập dữ liệu Iris, 128
tập dữ liệu MNIST, 75
data snooping bias – thiên kiến do dòm ngó dữ liệu, 45
data visualization – trực quan hóa dữ liệu
attribute combination – kết hợp thuộc tính, 54
computing correlations – tính toán độ tương quan, 51
dữ liệu địa lý, 49
test, training, and exploration set – tập kiểm tra, huấn luyện, và khám phá, 49
DataViz, *Xem* trực quan hóa dữ liệu
DBSCAN (density-based spatial clustering of applications with noise), 224
DBSCAN phân cấp – Hierarchical DBSCAN (HDBSCAN), 226
decision function – hàm quyết định, 82
Decision Stump – Gốc cây Quyết định, 177
Decision Tree – Cây Quyết định
CART training algorithm – thuật toán huấn luyện CART, 158
computational complexity – độ phức tạp tính toán, 158
estimating class probabilities – ước lượng xác suất các lớp, 157
evaluating – đánh giá, 64
Gini impurity versus entropy – độ pha tạp Gini với entropy, 159
instability drawback – điểm yếu bất ổn, 162
lợi ích, 154
making prediction – đưa ra dự đoán, 155
regression task – tác vụ hồi quy, 160
regularization hyperparameters – các siêu tham số điều chuẩn, 159

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

training and visualizing – huấn luyện và trực quan hóa, 154
decompression – giải nén, 195
deep belief network (DBN) – mạng niềm tin sâu, 12
deep neural network (DNN) – mạng nơ-ron sâu
định nghĩa, x
density estimation – ước lượng mật độ, 205, 231
development set (dev set) – tập phát triển, 28
diện tích dưới đường cong – area under the curve (AUC), 87
dimensionality reduction – giảm chiều
các kỹ thuật khác, 202
curse of dimensionality – lời nguyền chiều, 187
hướng tiếp cận, 188–190
LLE (Locally Linear Embedding) – Embedding Tuyến tính Cục bộ, 200
mục tiêu, 10
PCA (Principal Component Analysis) – Phân tích Thành phần Chính, 191–200
dùng phân cụm, 207
tổng quan, 186
dual problem – bài toán đối ngẫu, 149, 258
duck typing, 60
dummy attribute – thuộc tính giả, 59
dung sai – tolerance, 109
dự án mẫu
chuẩn bị dữ liệu, 55–63, 255
danh mục công việc cho dự án Học Máy, 33, 253
data visualization – trực quan hóa dữ liệu, 49–55, 254
dữ liệu thực, 31
lựa chọn phép đo chất lượng, 34
lựa chọn và huấn luyện mô hình, 63, 256
model fine-tuning – tinh chỉnh mô hình, 66–71, 256
mục tiêu của dự án, 32
phát biểu bài toán, 33, 253
tải dữ liệu, 37–48, 254
tổng quan, 31
triển khai, theo dõi, và bảo trì hệ thống, 71, 257
xác minh các giả định, 36
dự đoán biểu quyết theo đa số – majority-vote prediction, 164
dữ liệu – data, *Xem thêm* chuẩn bị dữ liệu; trực quan hóa dữ liệu; dữ liệu huấn luyện
compression – nén, 195
data mismatch – dữ liệu không tương đồng, 28
decompressing – giải nén, 195
dữ liệu địa lý, 49
giảm chiều, 193
nguồn, 31
noisy data – dữ liệu chứa nhiễu, 17
phân tích bằng phân cụm, 207
preprocessing – tiền xử lý, 220
reconstruction error – lỗi khôi phục, 195
skewed dataset – tập dữ liệu lệch, 79
tải, 40
tầm quan trọng so với thuật toán, 21
tập dữ liệu Giá nhà ở California, 32
tập dữ liệu Iris, 128
tập dữ liệu MNIST, 75
dữ liệu chứa nhiễu – noisy data, 17
dữ liệu huấn luyện – training data
đặc trưng không liên quan, 24
định nghĩa, 2
hold out – giữ lại, 27
kém chất lượng, 23
không đủ, 21
không mang tính đại diện, 22
overfitting – quá khớp, 24
underfitting – dưới khớp, 26
dừng sớm – early stopping, 124
dưới khớp – underfitting, 26

Đ

đại số tuyến tính – linear algebra, 99
đánh đổi độ chệch/phương sai – bias/variance trade-off, 119
đạo hàm dưới – subderivative, 152
đạo hàm riêng – partial derivative, 107
đặc trưng – feature, 7
đặc trưng đa thức – polynomial feature, 140
điểm số silhouette – silhouette score, 215
điểm thưởng – reward, 12
điều chuẩn – regularization
các siêu tham số cho Cây Quyết định, 159
định nghĩa, 25
kỹ thuật shrinkage, 179
điều chuẩn Tikhonov – Tikhonov regularization, 119
Định lý Không có Bữa trưa Miễn phí – No Free Lunch (NFL) theorem, 29

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

định lý Mercer – Mercer's theorem, 151
định thời học – learning schedule, 110
định thức hiệp phương sai nhỏ nhất – Fast-MCD (minimum covariance determinant), 240
độ chính xác – accuracy
  đánh giá bằng kiểm định chéo, 78
  định nghĩa, 79
  ví dụ, 2
độ nhạy – sensitivity, 81
độ pha tạp – impurity, 156, 159
độ pha tạp entropy – entropy impurity measure, 159
độ pha tạp Gini – Gini impurity measure, 159
đồ thị quá trình học – learning curve, 115–119
đồ thị silhouette – silhouette diagram, 216
độ trách nhiệm (phân cụm) – responsibility (clustering), 230
độc lập và phân phối giống nhau – independent and identically distributed (IID), 112
đường cong đặc trưng hoạt động của bộ nhận – receiver operating characteristic (ROC) curve, 86

E

early stopping – dừng sớm, 124
Elastic Net, 124
embedding, 60
Embedding Lân cận Ngẫu nhiên theo Phân phối t – t-Distributed Stochastic Neighbor Embedding (t-SNE), 202
Embedding Tuyến tính Cục bộ – LLE (Locally Linear Embedding), 200
ensemble, 165
ensemble Cây Siêu Ngẫu nhiên – Extremely Randomized Trees ensemble, 173
Ensemble Learning – Học Ensemble
  bagging và pasting, 168–171
  boosting, 174–182
  định nghĩa, 165
  lợi ích, 66
  Random Forest – Rừng Ngẫu nhiên, 172
  random patch and random subspace – patch và không gian con ngẫu nhiên, 171
  Rừng Ngẫu nhiên, 165
  stacking, 182
  trường hợp tối ưu, 167
  ví dụ, 165
Ensemble method – phương pháp Ensemble, 165
entropy impurity measure – độ pha tạp entropy, 159
epoch, 111
Euclidean norm – chuẩn Euclid, 36
evidence lower bound (ELBO) – cận dưới bằng chứng, 238
expectation step – bước kỳ vọng, 230
Expectation-Maximization (EM) algorithm – thuật toán Kỳ vọng-Cực đại hóa, 230
explained variance ratio – tỉ lệ phương sai được giải thích, 194
exploration set – tập khám phá, 49
Extra-Trees classifier – bộ phân loại Extra-Trees, 172
Extremely Randomized Trees ensemble – ensemble Cây Siêu Ngẫu nhiên, 173

F

F1 score – chỉ số F1, 82
false positive rate (FPR) – tỷ lệ dương tính giả, 86
Fast-MCD (minimum covariance determinant) – định thức hiệp phương sai nhỏ nhất, 240
feature – đặc trưng, 7
feature engineering – thiết kế đặc trưng, 24
feature extraction – trích xuất đặc trưng, 10, 24
feature map – ánh xạ đặc trưng, 199
feature scaling – co giãn đặc trưng, 61
feature selection – lựa chọn đặc trưng, 24
feature space – không gian đặc trưng, 197
feature vector – vector đặc trưng, 99
final trained model – mô hình đã được huấn luyện cuối cùng, 18
fitness function – hàm khớp, 18
fold, 65, 79
Full Gradient Descent, 108
fully-specified model architecture – kiến trúc mô hình được định nghĩa hoàn toàn, 18

G

Gaussian mixture model (GMM) – mô hình hỗn hợp Gauss
  Bayesian Gaussian Mixture model – mô hình hỗn hợp Bayes Gauss, 236
  các thuật toán phát hiện bất thường và tính mới khác, 240

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

chọn số cụm, 234
mô hình đồ thị, 228
cho phát hiện bất thường, 232
tổng quan, 227
variant – biến thể, 228
Gaussian Radial Basis Function (GRBF) – hàm tương tự Gauss, 141
generalization error – sai số khái quát, 27
generalized Lagrangian – Lagrangian tổng quát, 258
generative model – mô hình sinh, 230
giả định đa tạp – manifold assumption, 190
giả thuyết đa tạp – manifold hypothesis, 190
giả thuyết gốc – null hypothesis, 160
giá trị nhỏ nhất – global minimum, 106
giải nén – decompression, 195
giải tích – calculus, 99
giảm chiều – dimensionality reduction
các kỹ thuật khác, 202
curse of dimensionality – lời nguyền chiều, 187
hướng tiếp cận, 188–190
LLE (Locally Linear Embedding) – Embedding Tuyến tính Cục bộ, 200
mục tiêu, 10
PCA (Principal Component Analysis) – Phân tích Thành phần Chính, 191–200
dùng phân cụm, 207
tổng quan, 186
giảm chiều phi tuyến – nonlinear dimensionality reduction (NLDR), 200
Gini impurity measure – độ pha tạp Gini, 159
giữ lại – hold out, 27
global minimum – giá trị nhỏ nhất, 106
Gốc cây Quyết định – Decision Stump, 177
Gradient Boosted Regression Trees (GBRT), 177
Gradient Boosting, 177
Gradient Boosting Ngẫu nhiên – Stochastic Gradient Boosting, 181
Gradient Descent (GD) – Hạ Gradient
Batch Gradient Descent – Hạ gradient theo Batch, 107
Mini-batch Gradient Descent – Hạ Gradient theo Mini-batch, 112
Stochastic Gradient Descent – Hạ Gradient Ngẫu nhiên, 109
tổng quan, 98, 104
Gradient Tree Boosting, 177
greedy algorithms, 158

H

Hạ Gradient – Gradient Descent (GD)
Batch Gradient Descent – Hạ gradient theo Batch, 107
Mini-batch Gradient Descent – Hạ Gradient theo Mini-batch, 112
Stochastic Gradient Descent – Hạ Gradient Ngẫu nhiên, 109
tổng quan, 98, 104
Hạ Gradient Ngẫu nhiên – Stochastic Gradient Descent (SGD), 78, 109
Hạ Gradient theo Batch – Batch Gradient Descent, 107
Hạ Gradient theo Mini-batch – Mini-batch Gradient Descent, 112
Hàm (sigmoid) Logistic – Logistic (sigmoid) function, 126
hàm chi phí – cost function
cross-entropy loss (log loss) – mất mát entropy chéo (mất mát logarit), 132
hinge loss – mất mát hinge, 138, 152
mean absolute error (MAE) – trung bình sai số tuyệt đối, 36
mean squared error – trung bình bình phương sai số, 106
vai trò, 18
hàm hợp lý – likelihood function, 234
hàm khớp – fitness function, 18
hàm kích hoạt – activation function
Logistic (sigmoid), 126
hàm kích hoạt sigmoid (Logistic) – sigmoid (Logistic) activation function, 126
hàm lồi – convex function, 106
hàm lợi ích – utility function, 18
hàm mật độ xác suất – probability density function (PDF), 205, 231
hàm mất mát – loss function, *Xem* hàm chi phí
hàm mất mát hinge – hinge loss function, 138, 152
hàm quyết định – decision function, 82
hàm softmax – softmax function, 131
hàm tương tự – similarity function, 141
hàm tương tự Gauss – Gaussian Radial Basis Function (GRBF), 141
hàm tương tự Gauss – Radial Basis Function (RBF), 141
hard clustering – phân cụm cứng, 209
hard margin classification – phân loại biên cứng, 137

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

hard voting classifier – bộ phân loại biểu quyết cứng, 166
harmonic mean – trung bình điều hòa, 82
hạt nhân – kernel, 151, 197
hạt nhân chuỗi – string kernel, 143
hạt nhân chuỗi con – string subsequence kernel, 143
hạt nhân đa thức – polynomial kernel, 151
hạt nhân sigmoid – sigmoid kernel, 151
hệ số chặn – intercept term, 99
hệ số điều chỉnh – bias term, 99
hệ số silhouette – silhouette coefficient, 215
hệ số tương quan – correlation coefficient, 51
hệ số tương quan chuẩn – standard correlation coefficient, 51
hệ số tương quan Pearson r – Pearson's r, 51
hệ thống đề xuất – recommender system, 207
hierarchical clustering algorithm – thuật toán phân cụm phân cấp, 9
Hierarchical DBSCAN (HDBSCAN) – DBSCAN phân cấp, 226
high-dimensional training set – tập huấn luyện nhiều chiều, 186
hinge loss function – hàm mất mát hinge, 138, 152
Hinton, Geoffrey, x
histogram – biểu đồ tần suất, 43
học bán giám sát – semi-supervised learning
  định nghĩa, 11
  dùng phân cụm, 207, 221
  ví dụ, 12
học biểu diễn – representation learning, 60
học chủ động – active learning, 223
học có giám sát – supervised learning
  các tác vụ phổ biến, 7
  các thuật toán được đề cập, 8
  định nghĩa, 7
học dựa trên mẫu – instance-based learning, 15, 20
học dựa trên mô hình – model-based learning, 16
Học Đa Tạp – Manifold Learning, 190
Học Ensemble – Ensemble Learning
  bagging và pasting, 168–171
  boosting, 174–182
  định nghĩa, 165
  lợi ích, 66
  Random Forest – Rừng Ngẫu nhiên, 172
  random patch and random subspace – patch và không gian con ngẫu nhiên, 171
  Rừng Ngẫu nhiên, 165
  stacking, 182
  trường hợp tối ưu, 167
  ví dụ, 165
học gia tăng – incremental learning, 14
học không giám sát – unsupervised learning
  các tác vụ phổ biến, 9
  các thuật toán được đề cập, 9
  clustering – phân cụm, 206–227
  định nghĩa, 8
  Gaussian mixture model (GMM) – mô hình hỗn hợp Gauss, 227–240
  tổng quan, 205
học luật kết hợp – association rule learning, 11
Học Máy – Machine Learning (ML)
  định nghĩa, 1
  hướng tiếp cận trong việc học, xi
  kiến thức cần có, xi
  ký hiệu, 35, 146
  lịch sử, x
  lợi ích, 2
  những chủ đề được đề cập, xii
  tài liệu khác, xiv
  testing and validating – kiểm tra và đánh giá, 27–29
  thách thức, 21–27
  tổng quan, 26
  ứng dụng, x, 5
học ngoài bộ nhớ chính – out-of-core learning, 14
học ngoại tuyến – offline learning, 13
Học Tăng Cường – Reinforcement Learning (RL)
  tổng quan, 12
học theo batch – batch learning, 13
học trực tuyến – online learning, 13, 78
hold out – giữ lại, 27
holdout validation – kiểm định giữ lại, 28
Hồi quy Đa thức – Polynomial Regression, 98, 114
hồi quy k-Điểm Gần nhất – k-Nearest Neighbors regression, 20
Hồi quy Lasso – Lasso Regression, 122
Hồi quy Logistic – Logistic Regression
  decision boundary – ranh giới quyết định, 128
  huấn luyện và hàm chi phí, 127
  phân loại, 7
  Softmax Regression – Hồi quy Softmax, 131

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

tổng quan, 126
ước lượng xác suất, 126
Hồi quy Logistic Đa thức – Multinomial Logistic Regression, 131
Hồi quy Logit – Logit Regression, *Xem* Hồi quy Logistic
Hồi quy Ridge – Ridge Regression, 119
Hồi quy Softmax – Softmax Regression, 131
huấn luyện mô hình – training model
dự án mẫu, 63
định nghĩa, 18
Gradient Descent – Hạ Gradient, 104–113
learning curve – đồ thị quá trình học, 115–119
Linear Regression – Hồi quy Tuyến tính, 99–104
Logistic Regression – Hồi quy Logistic, 126–134
Polynomial Regression – Hồi quy Đa thức, 114, 115
regularized linear model – mô hình tuyến tính điều chuẩn, 119–126
tổng quan, 98
hyperparameter – siêu tham số
định nghĩa, 26
hyperparameter tuning – tinh chỉnh siêu tham số, 27, 66
learning rate – tốc độ học, 104
regularization hyperparameters – các siêu tham số điều chuẩn, 159
hyperplane – siêu mặt phẳng, 146
hypothesis boosting – boosting giả thuyết, 174

I

identity matrix – ma trận đơn vị, 121
image segmentation – phân vùng ảnh, 207, 218
impurity – độ pha tạp, 156, 159
incremental learning – học gia tăng, 14
Incremental PCA (IPCA) – PCA gia tăng, 197
independent and identically distributed (IID) – độc lập và phân phối giống nhau, 112
inequality constraint – ràng buộc bất đẳng thức, 258
inertia, 212
inference – suy luận, 20
information theory – lý thuyết thông tin, 159
initialization – khởi tạo
centroid initialization method – phương pháp khởi tạo tâm cụm, 212
random initialization – khởi tạo ngẫu nhiên, 104
inlier – mẫu bình thường, 232
instability – sự bất ổn, 162
instance segmentation – phân vùng thực thể, 218
instance-based learning – học dựa trên mẫu, 15, 20
intercept term – hệ số chặn, 99
inverse transformation – phép biến đổi nghịch đảo, 196
isolated environment – môi trường độc lập, 38
Isolation Forest algorithm – thuật toán Rừng Cô lập, 240
isomap algorithm – thuật toán isomap, 202

K

Karush–Kuhn–Tucker (KKT) multiplier – nhân tử Karush–Kuhn–Tucker, 258
K-Điểm trung bình – K-Means
accelerated and mini-batch – tăng tốc và mini-batch, 213
cải tiến, 212
centroid initialization method – phương pháp khởi tạo tâm cụm, 212
hạn chế, 217
hard and soft clustering – phân cụm cứng và mềm, 209
cho học bán giám sát, 221
image segmentation – phân vùng ảnh, 218
K-Means algorithm – Thuật toán K-Điểm trung bình, 210
optimal cluster number – số cụm tối ưu, 214
scaling input feature – co giãn đặc trưng đầu vào, 218
cho tiền xử lý, 220
tổng quan, 208
K-Điểm trung bình mini-batch – mini-batch K-Means, 213
K-Điểm trung bình tăng tốc – accelerated K-Means, 213
Keras
lợi ích, xi
kernel – hạt nhân, 151, 197
Kernel PCA (kPCA) – PCA Hạt nhân, 197–200
kernel trick – thủ thuật hạt nhân, 140, 199

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

kernelized SVM – SVM hạt nhân, 150
kênh alpha – alpha channel, 219
K-fold cross-validation – kiểm định chéo K-fold, 65, 79
khái quát hóa theo stack – stacked generalization, 182
khoảng cách Levenshtein – Levenshtein distance, 143
không gian ban đầu – original space, 197
không gian con – subspace, 188
không gian đặc trưng – feature space, 197
không gian tham số – parameter space, 107
khởi tạo – initialization
  centroid initialization method – phương pháp khởi tạo tâm cụm, 212
  random initialization – khởi tạo ngẫu nhiên, 104
khởi tạo môi trường làm việc, 37
khởi tạo ngẫu nhiên – random initialization, 104
kiểm định chéo – cross-validation, 28, 64, 78
kiểm định chéo K-fold – K-fold cross-validation, 65, 79
kiểm định giữ lại – holdout validation, 28
kiểm tra và đánh giá – testing and validation
  data mismatch – dữ liệu không tương đồng, 28
  hyperparameter tuning – tinh chỉnh siêu tham số, 27
  model selection – lựa chọn mô hình, 27
kiến trúc mô hình được định nghĩa hoàn toàn – fully-specified model architecture, 18
K-Means – K-Điểm trung bình
  accelerated and mini-batch – tăng tốc và mini-batch, 213
  cải tiến, 212
  centroid initialization method – phương pháp khởi tạo tâm cụm, 212
  hạn chế, 217
  hard and soft clustering – phân cụm cứng và mềm, 209
  cho học bán giám sát, 221
  image segmentation – phân vùng ảnh, 218
  K-Means algorithm – Thuật toán K-Điểm trung bình, 210
  optimal cluster number – số cụm tối ưu, 214
  scaling input feature – co giãn đặc trưng đầu vào, 218
  cho tiền xử lý, 220
  tổng quan, 208
k-Nearest Neighbors regression – hồi quy k-Điểm Gần nhất, 20
Kullback–Leibler divergence – phân kỳ Kullback–Leibler, 133

L

label – nhãn, 7, 34, 208
label propagation – lan truyền nhãn, 222
Lagrange multiplier – nhân tử Lagrange, 258
làm giàu kho ngữ liệu – corpus development, 21
lan truyền ái lực – affinity propagation, 227
lan truyền nhãn – label propagation, 222
landmark – mốc, 141
large margin classification – phân loại biên lớn, 136
Lasso Regression – Hồi quy Lasso, 122
latent variable – biến tiềm ẩn, 229
law of large numbers – luật số lớn, 166
lấy mẫu bất định – uncertainty sampling, 223
lấy mẫu stratified – stratified sampling, 47
lấy và sử dụng ví dụ mã nguồn, xv
leaf node – nút lá, 155
learning curve – đồ thị quá trình học, 115–119
learning rate – tốc độ học, 14, 104
learning schedule – định thời học, 110
Levenshtein distance – khoảng cách Levenshtein, 143
liblinear library – thư viện liblinear, 143
libsvm library – thư viện libsvm, 144
likelihood function – hàm hợp lý, 234
linear algebra – đại số tuyến tính, 99
Linear Discriminant Analysis (LDA) – Phân tích Phân biệt Tuyến tính, 202
linear model – mô hình tuyến tính, 17
Linear Regression model – mô hình Hồi quy Tuyến tính
  cách huấn luyện, 98, 100
  độ phức tạp tính toán, 103
  Normal Equation – Phương trình Pháp tuyến, 100
  tổng quan, 99
linear SVM classification – phân loại SVM tuyến tính, 136
LLE (Locally Linear Embedding) – Embedding Tuyến tính Cục bộ, 200
Lloyd-Forgy algorithm – thuật toán Lloyd–Forgy, 208

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

local minimum – cực tiểu, 106
Local Outlier Factor (LOF) – Nhân tố Ngoại lai Cục bộ, 240
log loss – mất mát logarit, 127
Logistic (sigmoid) function – Hàm (sigmoid) Logistic, 126
Logistic Regression – Hồi quy Logistic
  decision boundary – ranh giới quyết định, 128
  huấn luyện và hàm chi phí, 127
  phân loại, 7
  Softmax Regression – Hồi quy Softmax, 131
  tổng quan, 126
  ước lượng xác suất, 126
logit, 127
Logit Regression – Hồi quy Logit, *Xem* Hồi quy Logistic
log-odd, 127
loss function – hàm mất mát, *Xem* hàm chi phí
lỗi khôi phục – reconstruction error, 195
lời giải bài tập, 243–252
lời nguyền chiều – curse of dimensionality, 187
luật số lớn – law of large numbers, 166
lũy thừa chuẩn – normalized exponential, 131
lựa chọn đặc trưng – feature selection, 24
lựa chọn mô hình – model selection, 17, 27, 63
lượng phạt – penalty, 12
lý thuyết thông tin – information theory, 159
lý thuyết thông tin của Shannon – Shannon's information theory, 159

M

ma trận đơn vị – identity matrix, 121
ma trận nhầm lẫn – confusion matrix, 80
ma trận tham số – parameter matrix, 131
ma trận thưa – sparse matrix, 59
Machine Learning (ML) – Học Máy
  định nghĩa, 1
  hướng tiếp cận trong việc học, xi
  kiến thức cần có, xi
  ký hiệu, 35, 146
  lịch sử, x
  lợi ích, 2
  những chủ đề được đề cập, xii
  tài liệu khác, xiv
  testing and validating – kiểm tra và đánh giá, 27–29
  thách thức, 21–27
  tổng quan, 26
  ứng dụng, x, 5
majority-vote classifier – bộ phân loại biểu quyết theo đa số, 166
majority-vote prediction – dự đoán biểu quyết theo đa số, 164
mạng niềm tin sâu – deep belief network (DBN), 12
mạng nơ-ron sâu – deep neural network (DNN)
  định nghĩa, x
Manhattan norm – chuẩn Manhattan, 36
manifold assumption – giả định đa tạp, 190
manifold hypothesis – giả thuyết đa tạp, 190
Manifold Learning – Học Đa Tạp, 190
margin violation – vi phạm biên, 137
maximization step – bước cực đại hóa, 230
maximum a-posteriori (MAP) estimation – ước lượng hậu nghiệm cực đại, 235
maximum likelihood estimate (MLE) – ước lượng hợp lý cực đại, 235
máy Boltzmann giới hạn – restricted Boltzmann machine (RBM), 12
Máy Vector Hỗ trợ – Support Vector Machine (SVM)
  decision function and prediction – hàm quyết định và dự đoán, 146
  dual problem – bài toán đối ngẫu, 149
  kernelized SVM – SVM hạt nhân, 150
  linear SVM classification – phân loại SVM tuyến tính, 136
  lợi ích, 136
  nonlinear SVM classification – phân loại SVM phi tuyến, 139–144
  online SVM – SVM trực tuyến, 152
  SVM regression – SVM hồi quy, 144
  training objective – mục tiêu huấn luyện, 147
mất mát entropy chéo (mất mát logarit) – cross-entropy loss (log loss), 132
mất mát logarit – log loss, 127
mẫu bình thường – inlier, 232
mẫu chủ chốt – core instance, 224
mẫu huấn luyện – training instance, 2, 188
mẫu huấn luyện – training sample, 2
mean absolute error (MAE) – trung bình sai số tuyệt đối, 36
mean field variational inference – suy luận biến phân trường trung bình, 238
Mean-Shift algorithm – thuật toán Dịch-Trung bình, 227

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

measure of similarity – phép đo độ tương đồng, 15
Mercer's conditions – các điều kiện Mercer, 151
Mercer's theorem – định lý Mercer, 151
meta learner – bộ học meta, 182
metric – phép đo
  area under the curve (AUC) – diện tích dưới đường cong, 87
  confusion matrix – ma trận nhầm lẫn, 80
  F1 score – chỉ số F1, 82
  mean absolute error (MAE) – trung bình sai số tuyệt đối, 36
  mean squared error – trung bình bình phương sai số, 161
  precision, 80–85
  recall, 81–85
  RMSE, 34
  ROC curve – đường cong ROC, 86
mini-batch, 13, 112
Mini-batch Gradient Descent – Hạ Gradient theo Mini-batch, 112
mini-batch K-Means – K-Điểm trung bình mini-batch, 213
min-max scaling – co giãn min-max, 61
model – mô hình
  định nghĩa, 18
  fine-tuning – tinh chỉnh, 66–71
  huấn luyện, *Xem thêm* huấn luyện mô hình
  parametric versus nonparametric – tham số và phi tham số, 159
  training – huấn luyện, 18, 63
  white versus black box – hộp trắng và hộp đen, 157
model parameter – tham số mô hình, 17
model selection – lựa chọn mô hình, 17, 27, 63
model-based learning – học dựa trên mô hình, 16
mô hình – model
  định nghĩa, 18
  fine-tuning – tinh chỉnh, 66–71
  huấn luyện, *Xem thêm* huấn luyện mô hình
  parametric versus nonparametric – tham số và phi tham số, 159
  training – huấn luyện, 18, 63
  white versus black box – hộp trắng và hộp đen, 157
mô hình đã được huấn luyện cuối cùng – final trained model, 18
mô hình Hồi quy Tuyến tính – Linear Regression model
  cách huấn luyện, 98, 100
  độ phức tạp tính toán, 103
  Normal Equation – Phương trình Pháp tuyến, 100
  tổng quan, 99
mô hình hỗn hợp Bayes Gauss – Bayesian Gaussian Mixture model, 236
mô hình hỗn hợp Gauss – Gaussian mixture model (GMM)
  Bayesian Gaussian Mixture model – mô hình hỗn hợp Bayes Gauss, 236
  các thuật toán phát hiện bất thường và tính mới khác, 240
  chọn số cụm, 234
  mô hình đồ thị, 228
  cho phát hiện bất thường, 232
  tổng quan, 227
  variant – biến thể, 228
mô hình hộp đen – black box model, 157
mô hình hộp trắng – white box model, 157
mô hình phi tham số – nonparametric model, 159
mô hình sinh – generative model, 230
mô hình tham số – parametric model, 159
mô hình tuyến tính – linear model, 17
mô hình tuyến tính điều chuẩn – regularized linear model
  Elastic Net, 124
  Lasso Regression – Hồi quy Lasso, 122
  Ridge Regression – Hồi quy Ridge, 119
  tổng quan, 119
mô phỏng luyện kim – simulated annealing, 110
mốc – landmark, 141
môi trường độc lập – isolated environment, 38
multiclass classification – bài toán phân loại đa lớp, 89
Multidimensional Scaling (MDS) – Co giãn Đa chiều, 202
multilabel classification – phân loại đa nhãn, 94
multinomial classifier – bộ phân loại đa thức, 89
Multinomial Logistic Regression – Hồi quy Logistic Đa thức, 131
multioutput classification – phân loại đa đầu

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

ra, 95
multiple regression problem – bài toán đa hồi quy, 34
multivariate regression problem – bài toán hồi quy đa biến, 34

N
nén – compressing, 195
nghịch ảnh – pre-image, 199
nghịch ảnh khôi phục – reconstruction pre-image, 199
nghiệm dạng đóng – closed-form solution, 100
nhãn – label, 7, 34, 208
Nhận dạng Ký tự Quang học – Optical Character Recognition (OCR), 1
Nhân tố Ngoại lai Cục bộ – Local Outlier Factor (LOF), 240
nhiễu do lấy mẫu – sampling noise, 22
No Free Lunch (NFL) theorem – Định lý Không có Bữa trưa Miễn phí, 29
noisy data – dữ liệu chứa nhiễu, 17
nonlinear dimensionality reduction (NLDR) – giảm chiều phi tuyến, 200
nonlinear SVM classification – phân loại SVM phi tuyến, 139–144
nonparametric model – mô hình phi tham số, 159
Normal Equation – Phương trình Pháp tuyến, 100
normalization – chuẩn hóa, 61
normalized exponential – lũy thừa chuẩn, 131
novelty detection – phát hiện tính mới, 11, 233, 240
NP-Complete problem – bài toán NP-đầy đủ, 158
null hypothesis – giả thuyết gốc, 160
NumPy
  cài đặt, 37
  chuỗi hóa mảng lớn, 66
  hàm array_split(), 197
  hàm inv(), 101
  hàm randint(), 95
  hàm svd(), 193
  lớp memmap, 197
nút gốc – root node, 155
nút lá – leaf node, 155

O
observed variable – biến đã quan sát, 229
offline learning – học ngoại tuyến, 13
one-class SVM algorithm – thuật toán SVM một lớp, 240
one-hot encoding – biểu diễn one-hot, 59
one-versus-all (OvA) strategy – chiến lược một-toàn bộ, 89
one-versus-one (OvO) strategy – chiến lược một-một, 89
one-versus-the-rest (OvR) strategy – chiến lược một-còn lại, 89
online learning – học trực tuyến, 13, 78
online SVM – SVM trực tuyến, 152
Optical Character Recognition (OCR) – Nhận dạng Ký tự Quang học, 1
optimizer – bộ tối ưu
  Stochastic Gradient Descent (SGD) – Hạ Gradient Ngẫu nhiên, 78, 109
original space – không gian ban đầu, 197
outlier detection – phát hiện ngoại lai, 207, 232
out-of-core learning – học ngoài bộ nhớ chính, 14
out-of-sample error – sai số ngoài mẫu, 27
overfitting – quá khớp
  định nghĩa, 24

P
p (posterior) distribution – phân phối hậu nghiệm, 238
p (prior) distribution – phân phối tiên nghiệm, 237
parameter matrix – ma trận tham số, 131
parameter space – không gian tham số, 107
parameter vector – vector tham số, 99
parametric model – mô hình tham số, 159
partial derivative – đạo hàm riêng, 107
pasting, *Xem thêm* bagging và pasting
patch và không gian con ngẫu nhiên – random patch and random subspace, 171
PCA (Principal Component Analysis) – Phân tích Thành phần Chính
  bảo toàn phương sai, 191
  chiếu xuống d chiều, 193
  chọn số chiều, 194
  compression – nén, 195
  explained variance ratio – tỉ lệ phương sai được giải thích, 194
  gia tăng, 197
  Kernel PCA (kPCA) – PCA Hạt nhân, 197–200
  ngẫu nhiên, 196
  cho phát hiện bất thường và tính mới, 240

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

principal component axis – trục thành phần chính, 192
trong Scikit-Learn, 194
tổng quan, 191
PCA gia tăng – Incremental PCA (IPCA), 197
PCA Hạt nhân – Kernel PCA (kPCA), 197–200
PCA Ngẫu nhiên, 196
Pearson's r – hệ số tương quan Pearson r, 51
penalty – lượng phạt, 12
performance measure – phép đo chất lượng, *Xem* phép đo
phát hiện bất thường – anomaly detection
các thuật toán khác, 240
dùng hỗn hợp Gauss, 232
mục tiêu, 205
dùng phân cụm, 207
ví dụ, 10
phát hiện ngoại lai – outlier detection, 207, 232
phát hiện tính mới – novelty detection, 11, 233, 240
phân cụm cứng – hard clustering, 209
phân cụm mềm – soft clustering, 209
phân cụm phổ – spectral clustering, 227
phân kỳ Kullback–Leibler – Kullback–Leibler divergence, 133
phân loại biên cứng – hard margin classification, 137
phân loại biên lớn – large margin classification, 136
phân loại biên mềm – soft margin classification, 137
phân loại đa đầu ra – multioutput classification, 95
phân loại đa nhãn – multilabel classification, 94
phân loại SVM phi tuyến – nonlinear SVM classification, 139–144
phân loại SVM tuyến tính – linear SVM classification, 136
phân nhóm khách hàng – customer segmentation, 207
phân phối hạng mục – categorical distribution, 229
phân phối hậu nghiệm – p (posterior) distribution, 238
phân phối tiên nghiệm – p (prior) distribution, 237
Phân tích Giá trị Suy biến – Singular Value Decomposition (SVD), 103, 192
phân tích lỗi, 91
Phân tích Phân biệt Tuyến tính – Linear Discriminant Analysis (LDA), 202
Phân tích Thành phần Chính – PCA (Principal Component Analysis)
bảo toàn phương sai, 191
chiếu xuống d chiều, 193
chọn số chiều, 194
compression – nén, 195
explained variance ratio – tỉ lệ phương sai được giải thích, 194
gia tăng, 197
Kernel PCA (kPCA) – PCA Hạt nhân, 197–200
ngẫu nhiên, 196
cho phát hiện bất thường và tính mới, 240
principal component axis – trục thành phần chính, 192
trong Scikit-Learn, 194
tổng quan, 191
phân vùng ảnh – image segmentation, 207, 218
phân vùng màu – color segmentation, 218
phân vùng theo nhóm – semantic segmentation, 218
phân vùng thực thể – instance segmentation, 218
phép biến đổi – transformation
phép biến đổi nghịch đảo, 196
pipeline biến đổi, 61
tùy chỉnh, 60
phép biến đổi nghịch đảo – inverse transformation, 196
phép chiếu – projection, 188
phép chiếu ngẫu nhiên – random projection, 202
phép đo – metric
area under the curve (AUC) – diện tích dưới đường cong, 87
confusion matrix – ma trận nhầm lẫn, 80
F1 score – chỉ số F1, 82
mean absolute error (MAE) – trung bình sai số tuyệt đối, 36
mean squared error – trung bình bình phương sai số, 161
precision, 80–85
recall, 81–85
RMSE, 34

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

ROC curve – đường cong ROC, 86
phép đo chất lượng – performance measure, *Xem* phép đo
phép đo độ tương đồng – measure of similarity, 15
phép kiểm định chi-bình phương – chi-squared test, 160
phép xoay tập dữ liệu – training set rotation, 162
phương pháp Ensemble – Ensemble method, 165
phương sai – variance
bảo toàn, 191
explained variance ratio – tỉ lệ phương sai được giải thích, 194
Phương trình Pháp tuyến – Normal Equation, 100
pipeline, 33
policy – chính sách, 12
polynomial feature – đặc trưng đa thức, 140
polynomial kernel – hạt nhân đa thức, 151
Polynomial Regression – Hồi quy Đa thức, 98, 114
precision, 80–85
prediction problem – bài toán dự đoán, 7, 15, 165
pre-image – nghịch ảnh, 199
preprocessing – tiền xử lý, 220
primal problem – bài toán gốc, 149
probability density function (PDF) – hàm mật độ xác suất, 205, 231
projection – phép chiếu, 188
pruning – cắt tỉa, 160
p-value – trị số p, 160

Q

quá khớp – overfitting
định nghĩa, 24
Quadratic Programming (QP) problem – bài toán Quy hoạch Toàn phương, 148

R

Radial Basis Function (RBF) – hàm tương tự Gauss, 141
Random Forest – Rừng Ngẫu nhiên
Extra-Trees, 172
lợi ích, 165
tổng quan, 172
random initialization – khởi tạo ngẫu nhiên, 104
random patch and random subspace – patch và không gian con ngẫu nhiên, 171
random projection – phép chiếu ngẫu nhiên, 202
recall, 81–85
receiver operating characteristic (ROC) curve – đường cong đặc trưng hoạt động của bộ nhận, 86
recommender system – hệ thống đề xuất, 207
reconstruction error – lỗi khôi phục, 195
reconstruction pre-image – nghịch ảnh khôi phục, 199
regression problem – bài toán hồi quy
Decision Tree – Cây Quyết định, 160
định nghĩa, 7
k-Nearest Neighbors regression – hồi quy k-Điểm Gần nhất, 20
Lasso Regression – Hồi quy Lasso, 122
Linear Regression – Hồi quy Tuyến tính, 99–104
Logistic Regression – Hồi quy Logistic, 126–134
multiple regression problem – bài toán đa hồi quy, 34
multivariate regression problem – bài toán hồi quy đa biến, 34
Polynomial Regression – Hồi quy Đa thức, 114
Ridge Regression – Hồi quy Ridge, 119
Softmax Regression – Hồi quy Softmax, 131–134
SVM regression – SVM hồi quy, 144
univariate regression problem – bài toán hồi quy đơn biến, 34
regularization – điều chuẩn
các siêu tham số cho Cây Quyết định, 159
định nghĩa, 25
kỹ thuật shrinkage, 179
regularization term – số hạng điều chuẩn, 119
regularized linear model – mô hình tuyến tính điều chuẩn
Elastic Net, 124
Lasso Regression – Hồi quy Lasso, 122
Ridge Regression – Hồi quy Ridge, 119
tổng quan, 119
Reinforcement Learning (RL) – Học Tăng Cường
tổng quan, 12
representation learning – học biểu diễn, 60
residual error – sai số phần dư, 177
responsibility (clustering) – độ trách nhiệm

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

(phân cụm), 230
restricted Boltzmann machine (RBM) – máy Boltzmann giới hạn, 12
reward – điểm thưởng, 12
Ridge Regression – Hồi quy Ridge, 119
Root Mean Square Error (RMSE) – căn bậc hai trung bình bình phương sai số, 34, 106
root node – nút gốc, 155
Rừng Ngẫu nhiên – Random Forest
  Extra-Trees, 172
  lợi ích, 165
  tổng quan, 172

S

sai số khái quát – generalization error, 27
sai số ngoài mẫu – out-of-sample error, 27
sai số phần dư – residual error, 177
SAMME (Stagewise Additive Modeling sử dụng hàm mất mát Multiclass Exponential), 177
sampling bias – thiên kiến lấy mẫu, 23
sampling noise – nhiễu do lấy mẫu, 22
Scikit-Learn
  anomaly and novelty detection – phát hiện bất thường và tính mới, 240
  bộ biến đổi, 60
  bộ phân loại biểu quyết, 167
  các lớp SVM phân loại, 144
  các thuật toán phân cụm, 226
  cài đặt, 37
  CART training algorithm – thuật toán huấn luyện CART, 156, 158
  cấu trúc tập dữ liệu kiểu từ điển, 75
  chia tập dữ liệu thành các tập con, 46
  chuỗi biến đổi, 61
  data centering – căn giữa dữ liệu, 193
  đánh giá out-of-bag, 170
  đổi văn bản sang dạng số, 58
  feature scaling – co giãn đặc trưng, 137
  giảm chiều, 202
  hàm cross_val_score(), 78
  hàm mean_squared_error(), 64
  hồi quy tuyến tính, 102
  huấn luyện ensemble GBRT, 178
  huấn luyện gia tăng, 181
  linear model – mô hình tuyến tính, 19
  LLE (Locally Linear Embedding) – Embedding Tuyến tính Cục bộ, 200, 202
  lợi ích, xi
  lớp DecisionTreeRegressor, 160
  lớp ExtraTreesClassifier, 173
  lớp GridSearchCV, 67
  lớp IncrementalPCA, 197
  lớp KernelPCA, 197
  lớp SGDClassifier, 78
  lưu mô hình, 66
  mô hình SVM, 138
  nguyên tắc thiết kế, 56
  PCA, 194
  phiên bản AdaBoost, 177
  presorting data – tiền sắp xếp dữ liệu, 159
  Randomized PCA – thuật toán PCA Ngẫu nhiên, 196
  siêu tham số max_depth, 160
  siêu tham số random_state, 163
  stratified sampling – lấy mẫu stratified, 48
  SVD đầy đủ, 196
  tính năng kiểm định chéo K-fold, 65
  tính toán độ quan trọng của đặc trưng, 173
  tính toán phép đo cho bộ phân loại, 81–95
  tolerance hyperparameter – siêu tham số dung sai, 143
  triển khai, theo dõi, và bảo trì hệ thống, 71
  tự động khôi phục, 200
  vector one-hot, 59
  xử lý giá trị bị thiếu, 56
search engine – công cụ tìm kiếm, 207
semantic segmentation – phân vùng theo nhóm, 218
semi-supervised learning – học bán giám sát
  định nghĩa, 11
  dùng phân cụm, 207, 221
  ví dụ, 12
sensitivity – độ nhạy, 81
Shannon's information theory – lý thuyết thông tin của Shannon, 159
shrinkage, 179
siêu mặt phẳng – hyperplane, 146
siêu tham số – hyperparameter
  định nghĩa, 26
  hyperparameter tuning – tinh chỉnh siêu tham số, 27, 66
  learning rate – tốc độ học, 104
  regularization hyperparameters – các siêu tham số điều chuẩn, 159
sigmoid (Logistic) activation function – hàm kích hoạt sigmoid (Logistic), 126

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

sigmoid kernel – hạt nhân sigmoid, 151
silhouette coefficient – hệ số silhouette, 215
silhouette diagram – đồ thị silhouette, 216
silhouette score – điểm số silhouette, 215
similarity function – hàm tương tự, 141
simulated annealing – mô phỏng luyện kim, 110
Singular Value Decomposition (SVD) – Phân tích Giá trị Suy biến, 103, 192
skewed dataset – tập dữ liệu lệch, 79
slack variable – biến bù, 148
soft clustering – phân cụm mềm, 209
soft margin classification – phân loại biên mềm, 137
soft voting – biểu quyết mềm, 168
softmax function – hàm softmax, 131
Softmax Regression – Hồi quy Softmax, 131
số hạng điều chuẩn – regularization term, 119
spam filter – bộ lọc thư rác, 1, 2
sparse matrix – ma trận thưa, 59
spectral clustering – phân cụm phổ, 227
stacked generalization – khái quát hóa theo stack, 182
stacking, 182
standard correlation coefficient – hệ số tương quan chuẩn, 51
standardization – chuẩn tắc hóa, 61
stationary point – điểm dừng, 258
statistical mode – yếu vị thống kê, 169
statistical significance – ý nghĩa thống kê, 160
Stochastic Gradient Boosting – Gradient Boosting Ngẫu nhiên, 181
Stochastic Gradient Descent (SGD) – Hạ Gradient Ngẫu nhiên, 78, 109
stratified sampling – lấy mẫu stratified, 47
string kernel – hạt nhân chuỗi, 143
string subsequence kernel – hạt nhân chuỗi con, 143
strong learners – bộ học mạnh, 166
subderivative – đạo hàm dưới, 152
subspace – không gian con, 188
supervised learning – học có giám sát
  các tác vụ phổ biến, 7
  các thuật toán được đề cập, 8
  định nghĩa, 7
support vector – vector hỗ trợ, 137
Support Vector Machine (SVM) – Máy Vector Hỗ trợ
  decision function and prediction – hàm quyết định và dự đoán, 146
  dual problem – bài toán đối ngẫu, 149, 258
  kernelized SVM – SVM hạt nhân, 150
  linear SVM classification – phân loại SVM tuyến tính, 136
  lợi ích, 136
  nonlinear SVM classification – phân loại SVM phi tuyến, 139–144
  online SVM – SVM trực tuyến, 152
  SVM regression – SVM hồi quy, 144
  training objective – mục tiêu huấn luyện, 147
suy luận – inference, 20
suy luận biến phân – variational inference, 238
suy luận biến phân hộp đen ngẫu nhiên – black box stochastic variational inference (BBSVI), 238
suy luận biến phân trường trung bình – mean field variational inference, 238
sự bất ổn – instability, 162
SVM hạt nhân – kernelized SVM, 150
SVM trực tuyến – online SVM, 152

T

tác nhân – agent, 12
tail-heavy histogram – biểu đồ tần suất nặng đuôi, 44
tâm cụm – centroid, 207
tập dữ liệu Giá nhà ở California – California Housing Prices dataset, 32
tập dữ liệu Iris, 128
tập dữ liệu lệch – skewed dataset, 79
tập dữ liệu MNIST, 75
tập huấn luyện – training set, 2, 27, 186
tập huấn luyện nhiều chiều – high-dimensional training set, 186
tập huấn luyện phát triển – train-dev set, 28
tập khám phá – exploration set, 49
tập kiểm định – validation set, 28
tập kiểm tra – test set, 27, 45
tập phát triển – development set (dev set), 28
t-Distributed Stochastic Neighbor Embedding (t-SNE) – Embedding Lân cận Ngẫu nhiên theo Phân phối t, 202
TensorFlow, căn bản
  lợi ích, xi
TensorFlow, triển khai mô hình trên quy mô lớn
  triển khai trên nền tảng AI, 72

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

test set – tập kiểm tra, 27, 45
testing and validation – kiểm tra và đánh giá
  data mismatch – dữ liệu không tương đồng, 28
  hyperparameter tuning – tinh chỉnh siêu tham số, 27
  model selection – lựa chọn mô hình, 27
tham số biến phân – variational parameter, 238
tham số mô hình – model parameter, 17
thành phần – component, 33
theoretical information criterion – tiêu chí thông tin lý thuyết, 234
thiên kiến do dòm ngó dữ liệu – data snooping bias, 45
thiên kiến lấy mẫu – sampling bias, 23
thiết kế đặc trưng – feature engineering, 24
thủ thuật hạt nhân – kernel trick, 140, 199
thuật toán – algorithm
  anomaly detection – phát hiện bất thường, 240
  BIRCH algorithm – thuật toán BIRCH, 227
  CART training algorithm – thuật toán huấn luyện CART, 156, 158
  clustering algorithm – thuật toán phân cụm, 9
  Expectation-Maximization (EM) algorithm – thuật toán Kỳ vọng-Cực đại hóa, 230
  greedy algorithm – thuật toán tham lam, 158
  hierarchical clustering algorithm – thuật toán phân cụm phân cấp, 9
  Isolation Forest algorithm – thuật toán Rừng Cô lập, 240
  K-Means algorithm – thuật toán K-Điểm trung bình, 208
  Lloyd–Forgy algorithm – thuật toán Lloyd–Forgy, 208
  Mean-Shift algorithm – thuật toán Dịch-Trung bình, 227
  one-class SVM algorithm – thuật toán SVM một lớp, 240
  Randomized PCA – thuật toán PCA Ngẫu nhiên, 196
  supervised learning – học có giám sát, 7
  tầm quan trọng của dữ liệu so với, 21
  thuật toán isomap, 202
  unsupervised learning – học không giám sát, 8
  visualization algorithm – thuật toán trực quan hóa, 9
thuật toán BIRCH – BIRCH algorithm, 227
thuật toán Dịch-Trung bình – Mean-Shift algorithm, 227
thuật toán huấn luyện CART – CART training algorithm, 156, 158
thuật toán isomap – isomap algorithm, 202
thuật toán Kỳ vọng-Cực đại hóa – Expectation-Maximization (EM) algorithm, 230
thuật toán Lloyd–Forgy – Lloyd-Forgy algorithm, 208
thuật toán phân cụm – clustering algorithm
  các thuật toán khác, 226
  DBSCAN, 224
  cho học bán giám sát, 221
  K-Means – K-Điểm trung bình, 208–218
  mục tiêu, 205
  cho phân vùng ảnh, 207, 218
  cho tiền xử lý, 220
  tổng quan, 206
  ứng dụng, 9, 206
thuật toán phân cụm phân cấp – hierarchical clustering algorithm, 9
thuật toán Rừng Cô lập – Isolation Forest algorithm, 240
thuật toán SVM một lớp – one-class SVM algorithm, 240
thuật toán trực quan hóa – visualization algorithm, 9
thuộc tính – attribute, 7
thuộc tính giả – dummy attribute, 59
thư viện liblinear – liblinear library, 143
thư viện libsvm – libsvm library, 144
tỉ lệ phương sai được giải thích – explained variance ratio, 194
tích cụm – agglomerative clustering, 226
tiền xử lý – preprocessing, 220
tiêu chí thông tin Akaike – Akaike information criterion (AIC), 234
tiêu chí thông tin Bayes – Bayesian information criterion (BIC), 234
tiêu chí thông tin lý thuyết – theoretical information criterion, 234
Tikhonov regularization – điều chuẩn Tikhonov, 119
tính hội tụ – convergence, 104
toán tử argmax, 132
tolerance – dung sai, 109
tốc độ học – learning rate, 14, 104
tối ưu có điều kiện – constrained optimization,

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

148
train-dev set – tập huấn luyện phát triển, 28
training data – dữ liệu huấn luyện
đặc trưng không liên quan, 24
định nghĩa, 2
hold out – giữ lại, 27
kém chất lượng, 23
không đủ, 21
không mang tính đại diện, 22
overfitting – quá khớp, 24
underfitting – dưới khớp, 26
training instance – mẫu huấn luyện, 2, 188
training model – huấn luyện mô hình
dự án mẫu, 63
định nghĩa, 18
Gradient Descent – Hạ Gradient, 104–113
learning curve – đồ thị quá trình học, 115–119
Linear Regression – Hồi quy Tuyến tính, 99–104
Logistic Regression – Hồi quy Logistic, 126–134
Polynomial Regression – Hồi quy Đa thức, 114, 115
regularized linear model – mô hình tuyến tính điều chuẩn, 119–126
tổng quan, 98
training sample – mẫu huấn luyện, 2
training set – tập huấn luyện, 2, 27, 186
training set rotation – phép xoay tập dữ liệu, 162
transformation – phép biến đổi
phép biến đổi nghịch đảo, 196
pipeline biến đổi, 61
tùy chỉnh, 60
trị số p – p-value, 160
trí tuệ đám đông, 165
trích xuất đặc trưng – feature extraction, 10, 24
true negative rate (TNR) – tỷ lệ âm tính thật, 86
true positive rate (TPR) – tỷ lệ dương tính thật, 81
trung bình điều hòa – harmonic mean, 82
trung bình độ lệch tuyệt đối – average absolute deviation, 36
trung bình sai số tuyệt đối – mean absolute error (MAE), 36
trực quan hóa dữ liệu
dimensionality reduction – giảm chiều, 186
trực quan hóa dữ liệu – data visualization
attribute combination – kết hợp thuộc tính, 54
computing correlations – tính toán độ tương quan, 51
dữ liệu địa lý, 49
test, training, and exploration set – tập kiểm tra, huấn luyện, và khám phá, 49
tỷ lệ âm tính thật – true negative rate (TNR), 86
tỷ lệ dương tính giả – false positive rate (FPR), 86
tỷ lệ dương tính thật – true positive rate (TPR), 81

U

uncertainty sampling – lấy mẫu bất định, 223
underfitting – dưới khớp, 26
univariate regression problem – bài toán hồi quy đơn biến, 34
unsupervised learning – học không giám sát
các tác vụ phổ biến, 9
các thuật toán được đề cập, 9
clustering – phân cụm, 206–227
định nghĩa, 8
Gaussian mixture model (GMM) – mô hình hỗn hợp Gauss, 227–240
tổng quan, 205
utility function – hàm lợi ích, 18

Ư

ước lượng hậu nghiệm cực đại – maximum a-posteriori (MAP) estimation, 235
ước lượng hợp lý cực đại – maximum likelihood estimate (MLE), 235
ước lượng mật độ – density estimation, 205, 231

V

validation set – tập kiểm định, 28
variance – phương sai
bảo toàn, 191
explained variance ratio – tỉ lệ phương sai được giải thích, 194
variational inference – suy luận biến phân, 238
variational parameter – tham số biến phân, 238
vector

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

feature vector – vector đặc trưng, 99
parameter vector – vector tham số, 99
vector cột, 100
vector subgradient, 123
vector cột, 100
vector đặc trưng – feature vector , 99
vector hỗ trợ – support vector, 137
vector subgradient, 123
vector tham số – parameter vector, 99
vi phạm biên – margin violation, 137
visualization algorithm – thuật toán trực quan hóa, 9

W

weak learners – bộ học yếu, 166
white box model – mô hình hộp trắng, 157

X

XGBoost, 182

Y

ý nghĩa thống kê – statistical significance, 160
yếu vị thống kê – statistical mode, 169

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Giới thiệu về Tác giả

Aurélien Géron là một nhà tư vấn đồng thời là giảng viên về Học Máy. Từng là nhân viên của Tập đoàn Google, ông đã dẫn dắt nhóm phân loại video của Youtube từ năm 2013 đến năm 2016. Ông cũng là người sáng lập và là Giám đốc Công nghệ (CTO) tại nhiều công ty khác nhau như: Wifirst, một nhà cung cấp dịch vụ Internet không dây hàng đầu tại Pháp; Polycon-seil, một công ty tư vấn tập trung vào mảng viễn thông, truyền thông và chiến lược; và Kiwisoft, một công ty chuyên tập trung vào Học Máy và Quyền riêng tư của Dữ liệu.

Trước đó, ông từng là Kỹ sư làm việc trong nhiều lĩnh vực: Tài chính (JP Morgan và Société Générale), Quốc phòng (Bộ quốc phòng Canada - Canada DOD), và Chăm sóc Sức khỏe (Truyền máu). Ông cũng đã xuất bản một số cuốn sách kỹ thuật (về kiến trúc của C++, WiFi, và Internet), và giảng dạy về Khoa học Máy tính tại một số trường Kỹ thuật của Pháp.

Một vài sự thật thú vị: ông ấy đã dạy ba đứa con của mình đếm số nhị phân bằng ngón tay (lên đến 1,023), ông ấy từng nghiên cứu Vi sinh vật học và Di truyền học Tiến hóa trước khi chuyển sang Kỹ thuật Phần mềm, và chiếc dù của ông ấy không bung ra ở lần nhảy thứ hai.

Lời bạt

Con vật trên trang bìa của cuốn sách này là một con kỳ nhông lửa (Salamandra salamandra), đây là một loài lưỡng cư được tìm thấy hầu hết ở châu Âu. Lớp da đen bóng của chúng có những chấm vàng lớn trên đầu và lưng mang độc tố có tên là alkloid. Đây có thể là nguồn gốc cho tên gọi chung của loài lưỡng cư này, khi tiếp xúc với những chất độc sẽ gây ra hiện tượng tăng thông khí (hyperventilation) và co giật.

Kỳ nhông lửa thường sống trong các khu rừng râm mát, ẩn náu trong các khe ẩm và dưới các khúc gỗ gần vũng nước hoặc các vực nước ngọt, nơi tạo điều kiện cho chúng sinh sản. Mặc dù dành phần lớn cuộc đời ở trên cạn, nhưng chúng lại sinh con ở dưới nước. Thức ăn của chúng chủ yếu là các loài côn trùng, nhện, giun đất, ốc sên và sa giông. Một cá thể kỳ nhông lửa trưởng thành có thể dài đến một foot (khoảng 30,48 cm), và nếu sống trong môi trường nuôi nhốt thì chúng có thể sống đến 50 năm.

Số lượng cá thể kỳ nhông lửa đã giảm do nạn tàn phá rừng và nạn buôn lậu, thế nhưng mối đe dọa lớn nhất mà chúng phải đối mặt là lớp da nhạy của chúng đang phải tiếp xúc với các chất ô nhiễm và vi khuẩn. Kể từ năm 2014, chúng đã tuyệt chủng ở một số vùng của Hà Lan và Bỉ do sự ảnh hưởng của một bệnh nhiễm nấm du nhập (có tên khoa học là B. Salamandrivorans), lây lan bởi sa giông và chim.

Đã có nhiều loại động vật xuất hiện trên các bìa sách của O'Reilly đang đứng trước nguy cơ tuyệt chủng; tất cả các loài vật này đều quan trọng đối với thế giới. Hình ảnh minh họa của trang bìa được chắp bút bởi Karen Montgomery, dựa trên tác phẩm Minh họa Lịch sử Tự nhiên của John G. Wood. Phông chữ trên bìa là Inter của Rasmus Andersson. Phông chữ văn bản là Knuth's Computer Modern và phông chữ mã lập trình là JetBrains Mono của Philipp Nurullin.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

NHÓM DỊCH THUẬT MACHINE LEARNING CƠ BẢN

Những đầu sách kinh điển về Trí tuệ Nhân tạo, Học Máy,... tất cả những gì bạn cần để bắt đầu sự nghiệp của mình với chủ đề này.

Tìm hiểu thêm các dự án của MLBVN tại github.com/mlbvn

THỰC HÀNH HỌC MÁY VỚI
SCIKIT-LEARN, KERAS & TENSORFLOW
TẬP 1 - KIẾN THỨC NỀN TẢNG VỀ HỌC MÁY

CHỊU TRÁCH NHIỆM XUẤT BẢN, NỘI DUNG
Giám đốc – Tổng Biên tập: Trần Chí Đạt

CHỊU TRÁCH NHIỆM BẢN THẢO
Phó Giám đốc - Phó Tổng biên tập: Ngô Thị Mỹ Hạnh
Biên tập Nội dung: Nguyễn Tiến Phát - Bùi Hữu Lộ
Biên tập Sách điện tử: Nguyễn Tiến Phát - Bùi Hữu Lộ
Thiết kế bìa: Đoàn Võ Duy Thanh
Trình bày sách: Đoàn Võ Duy Thanh, Nguyễn Văn Cường, Phạm Minh Đức

---

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG
Website: www.nxbthongtintruyenthong.vn
Sàn giao dịch sách in: book365.vn, ebook365.vn

Trụ sở chính
Tầng 6, Tòa nhà 115 Trần Duy Hưng, Trung Hòa, Q. Cầu Giấy, TP. Hà Nội
- Điện thoại: 024.3577.2143 — Điện thoại Phát hành: 024.3577.2138
- Fax: 024.3557.9858, 3577.2194 — Email: nxb.tttt@mic.gov.vn

Chi nhánh TP.HCM
Số 211 Nguyễn Gia Trí, Phường 25, Quận Bình Thạnh, TP. Hồ Chí Minh
- Điện thoại: 028.3512.7750, 3512.7751 — Fax: 028.3512.7751
- Email: cnsg.nxbtttt@mic.gov.vn

Chi nhánh Đà Nẵng
Số 42 Trần Quốc Toản, Quận Hải Châu, TP. Đà Nẵng
- Điện thoại: 0236.3897.467 — Fax: 0236.3843.359
- Email: cndn.nxbtttt@mic.gov.vn

Chi nhánh Tây Nguyên
Số 46 Đường Y Jút, Phường Thống Nhất, TP. Buôn Ma Thuột, Tỉnh Đắk Lắk
- Điện thoại: 0262.3808.088 — Email: cntn.nxbtttt@mic.gov.vn

Số xác nhận Đăng ký Xuất bản: 2557-2021/CXBIPH/3 - 97/TTTT
Số quyết định Xuất bản: 78/QĐ - NXB TTTT ngày 27/07/2021
Nộp lưu chiểu Quý III năm 2021.
ISBN: 978-604-80-5661-2

Thân gửi bạn **Nguyễn An Khang** (*nguyenankhang1505@gmail.com*)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

O'REILLY®

Thực hành Học Máy với Scikit-Learn, Keras, và TensorFlow

TẬP 1

Thông qua một loạt đột phá, Học Sâu đã thúc đẩy toàn bộ lĩnh vực Học Máy. Bây giờ, ngay cả các lập trình viên không biết gì về công nghệ này cũng có thể sử dụng các công cụ đơn giản, hiệu quả để triển khai các chương trình có khả năng học từ dữ liệu. Ấn bản cập nhật của cuốn sách bán chạy nhất này sử dụng các ví dụ cụ thể, lý thuyết tinh gọn và các Python framework được sử dụng rộng khắp trong nhiều sản phẩm thương mại để giúp bạn hiểu trực quan về các khái niệm và công cụ để xây dựng những hệ thống thông minh.

Bạn sẽ học được một loạt các kỹ thuật mà bạn có thể nhanh chóng sử dụng. Với các bài tập trong mỗi chương giúp bạn áp dụng những gì đã học, tất cả những gì bạn cần là kinh nghiệm lập trình để bắt đầu. Tất cả mã nguồn đều có trên Github. Nó đã được cập nhật lên TensorFlow 2 và phiên bản mới nhất của Scikit-Learn.

- **Tìm hiểu các kiến thức nền tảng về Học Máy thông qua Dự án sử dụng Scikit-Learn và Pandas từ đầu tới cuối.**
- **Xây dựng và huấn luyện nhiều kiến trúc mạng nơ-ron cho bài toán phân loại và hồi quy sử dụng TensorFlow 2.**
- **Khám phá khả năng phát hiện đối tượng, phân đoạn ngữ nghĩa, cơ chế tập trung, mô hình ngôn ngữ, GAN, v.v.**
- **Khám phá Keras API, API cao cấp chính thức cho TensorFlow 2.**
- **Sản xuất mô hình TensorFlow bằng cách sử dụng API dữ liệu của TensorFlow, các chiến lượng phân phối API, TF-Transform, và TF-Serving.**
- **Triển khai trên Google Cloud AI Platform hoặc trên thiết bị di động.**
- **Khai thác các kỹ thuật học tập không giám sát như Giảm chiều, phân cụm, và phát hiện bất thường.**
- **Tạo tác nhân học tập tự chủ với Học tăng cường, bao gồm cả việc sử dụng thư viện TF-Agents.**

"Nguồn tài nguyên hữu ích để học về Học Máy. Bạn sẽ tìm thấy những lời giải thích trực quan, rõ ràng và vô số lời khuyên thiết thực."

—François Chollet
Tác giả của Keras và cuốn sách Deep Learning with Python

"Cuốn sách là một giới thiệu tuyệt vời về lý thuyết và thực hành giải các bài toán với mạng Nơ-ron. Tôi giới thiệu nó cho bất kỳ ai quan tâm đến việc thực hành Học Máy."

—Pete Warden
Trưởng nhóm Thiết bị Di động tại TensorFlow

Aurélien Géron là một nhà tư vấn đồng thời là giảng viên về Học Máy. Từng là nhân viên của Tập đoàn Google, ông đã dẫn dắt nhóm phân loại video của Youtube từ năm 2013 đến năm 2016. Ông cũng là người sáng lập và là Giám đốc Công nghệ của Wifirst (một nhà cung cấp dịch vụ Internet không dây hàng đầu tại Pháp) từ 2002 đến 2012.

Thân gửi bạn Nguyễn An Khang (nguyenankhang1505@gmail.com)
Cuốn sách này đã được bảo hộ, bản quyền thuộc về MLBVN Group & FUNiX.

NHÓM DỊCH THUẬT
MACHINE LEARNING CƠ BẢN

9 786048 056612