**TRƯỜNG ĐẠI HỌC CNTT&TT  
KHOA CÔNG NGHỆ THÔNG TIN**

**Đàm Thanh Phương  
Hà Thị Thanh  
Trần Quang Quý**

BÀI GIẢNG

**HỌC MÁY**

**NĂM HỌC 2022-2023**

**LƯU HÀNH NỘI BỘ**

Mục lục

[**Chương I TỔNG QUAN**](#bookmark174)

1. [**Các khái niệm cơ bản** 6](#bookmark182)
   1. [Khái niệm học máy 6](#bookmark186)
   2. [Dữ liệu 6](#bookmark191)
   3. [Các bài toán cơ bản trong học máy 7](#bookmark196)
   4. [Các thuật toán học máy 9](#bookmark224)
   5. [Hàm mất mát và tham số mô hình 11](#bookmark245)
   6. [Những thách thức chính của học máy 12](#bookmark250)
   7. [Quy trình thực hiện một dự án học máy 14](#bookmark274)
2. [**Các kỹ thuật xây dựng đặc trưng** 20](#bookmark394)
   1. [Giới thiệu 20](#bookmark398)
   2. [Mô hình chung cho các bài toán học máy 20](#bookmark403)
   3. [Một số kỹ thuật trích chọn đặc trưng 23](#bookmark418)
   4. [Chuẩn hoá vector đặc trưng 25](#bookmark445)
   5. [BÀI TẬP CUỐI CHƯƠNG 26](#bookmark462)

[**Chương II CÁC THUẬT TOÁN HỌC CÓ GIÁM SÁT**](#bookmark468)

1. [**Hồi quy tuyến tính** 30](#bookmark481)
   1. [Giới thiệu 30](#bookmark485)
   2. [Xây dựng và tối ưu hàm mất mát 31](#bookmark491)
   3. [Ví dụ trên Python 33](#bookmark520)
   4. [Thảo luận 36](#bookmark547)
2. [***K* lân cận** 39](#bookmark570)
   1. [Giới thiệu 39](#bookmark574)
   2. [Phân tích toán học 39](#bookmark583)
   3. [Ví dụ trên cơ sở dữ liệu Iris 41](#bookmark589)
   4. [Thảo luận 44](#bookmark602)
3. [**Bộ phân loại naive Bayes** 46](#bookmark624)
   1. [Bộ phân loại naive Bayes 46](#bookmark628)
   2. [Các phân phối thường dùng trong NBC 48](#bookmark636)
   3. [Ví dụ 49](#bookmark653)
   4. [Thảo luận 56](#bookmark676)
4. [**Hạ Gradient** 57](#bookmark685)
   1. [Giới thiệu 57](#bookmark689)
   2. [Hạ gradient cho hàm một biến 58](#bookmark698)
   3. [Hạ gradient cho hàm nhiều biến 63](#bookmark718)
   4. [Hạ gradient với momentum 66](#bookmark724)
   5. [Nesterov accelerated gradient 69](#bookmark737)
   6. [Hạ gradient ngẫu nhiên 70](#bookmark750)
   7. [Thảo luận 72](#bookmark767)
5. [**Thuật toán học perceptron** 74](#bookmark783)
   1. [Giới thiệu 74](#bookmark787)
   2. [Thuật toán học perceptron 75](#bookmark793)
   3. [Ví dụ và minh hoạ trên Python 78](#bookmark823)
   4. [Mô hình mạng neuron đầu tiên 79](#bookmark832)
   5. [Thảo Luận 81](#bookmark838)
6. [**Hồi quy logistic** 83](#bookmark845)
   1. [Giới thiệu 83](#bookmark849)
   2. [Hàm mất mát và phương pháp tối ưu 85](#bookmark873)
   3. [Triển khai thuật toán trên Python 88](#bookmark895)
   4. [Tính chất của hồi quy logistic 91](#bookmark910)
   5. [Bài toán phân biệt hai chữ số viết tay 93](#bookmark920)
   6. [Bài toán phân loại đa lớp 94](#bookmark925)
   7. [Thảo luận 96](#bookmark946)
7. [**Hồi quy softmax** 99](#bookmark975)
   1. [Giới thiệu 99](#bookmark979)
   2. [Hàm softmax 100](#bookmark962)
   3. [Hàm mất mát và phương pháp tối ưu 103](#bookmark1000)
   4. [Ví dụ trên Python 108](#bookmark1033)
   5. [Thảo luận 111](#bookmark1042)
8. [**Máy vector hỗ trỢ** 112](#bookmark1058)
   1. [Giới thiệu 112](#bookmark1062)
   2. [Xây dựng bài toán tối ưu cho máy vector hỗ trợ 114](#bookmark1077)
   3. [Bài toán đối ngẫu của máy vector hỗ trợ 116](#bookmark1084)
   4. [Lập trình tìm nghiệm cho máy vector hỗ trợ 119](#bookmark1117)
   5. [Tóm tắt 121](#bookmark1134)
   6. [BÀI TẬP CUỐI CHƯƠNG 122](#bookmark1142)

[**Chương III CÁC THUẬT TOÁN HỌC KHÔNG GIÁM SÁT**](#bookmark1151)

1. [**Phân cụm *K*-means** 126](#bookmark1171)
   1. [Giới thiệu 126](#bookmark1175)
   2. [Phân tích toán học 127](#bookmark1181)
   3. [Ví dụ trên Python 130](#bookmark1211)
   4. [Phân cụm chữ số viết tay 134](#bookmark1233)
   5. [Tách vật thể trong ảnh 137](#bookmark1253)
   6. [Nén ảnh 138](#bookmark1262)
   7. [Thảo luận 139](#bookmark1269)
2. [**Phân tích thành phần chính** 143](#bookmark1294)
   1. [Phân tích thành phần chính 143](#bookmark1298)
   2. [Các bước thực hiện phân tích thành phần chính 148](#bookmark1319)
   3. [Liên hệ với phân tích giá tri suy biến 149](#bookmark1331)
   4. [Làm thế nào để chọn số chiều của dữ liệu mới 151](#bookmark1352)
   5. [Lưu ý về tính toán phân tích thành phần chính 151](#bookmark1357)
   6. [Một số ứng dụng 152](#bookmark1373)
   7. [Thảo luận 156](#bookmark1393)
   8. [BÀI TẬP CUỐI CHƯƠNG 156](#bookmark1400)

[**Chương IV HỆ THỐNG GƠI** ý](#bookmark1403)

1. [**Hệ thống gỢỈ ý dựa trẽn nôi dung** 160](#bookmark1411)
   1. [Giới thiệu 160](#bookmark1415)
   2. [Ma trận tiện ích 161](#bookmark1430)
   3. [Hệ thống dựa trên nội dung 163](#bookmark1445)
   4. [Bài toán MovieLens 100k 166](#bookmark1469)
   5. [Thảo luận 170](#bookmark1498)
2. [**Lọc cộng tác lân cận** 171](#bookmark1511)
   1. [Giới thiệu 171](#bookmark1515)
   2. [Lọc cộng tác theo người dùng 172](#bookmark1522)
   3. [Lọc cộng tác sản phẩm 177](#bookmark1541)
   4. [Lập trình trên Python 179](#bookmark1553)
   5. [Thảo luận 182](#bookmark1571)
3. [**Lọc cộng tác phân tích ma trận** 183](#bookmark1580)
   1. [Giới thiệu 183](#bookmark1584)
   2. [Xây dựng và tối ưu hàm mất mát 185](#bookmark1594)
   3. [Lập trình Python 187](#bookmark1613)
   4. [Thảo luận 190](#bookmark1624)

DANH SÁCH HÌNH ẢNH

1. [Minh hoạ dữ liệu và đường thẳng xấp xỉ tìm được bởi hồi quy](#bookmark538)

tuyến tính 35

1. (a) Hồi quy đa thức bậc ba (b) Hồi quy tuyến tính nhạy cảm với

nhiễu 36

1. Khảo sát sự biến thiên của một đa thức bậc hai 58
2. Nghiệm của bài toán hồi quy tuyến tính (đường thằng màu đen)

[tìm được bằng thư viện scikit-learn 64](#bookmark1227)

1. [Đường đi nghiệm của hồi quy tuyến tính với các tốc độ học khác](#bookmark721)

nhau. 65

1. Đường đi của nghiệm cho bài toán hồi quy tuyến tính với hai phương pháp Hạ gradient khác nhau. NAG cho nghiệm mượt hơn

và nhanh hơn 70

1. Ví dụ về giá tri hàm mất mát sau mỗi vòng lặp khi sử dụng

mini-batch Hạ gradient. Hàm mất mát dao động sau mỗi lần cập nhật nhưng nhìn chung giảm dần và có xu hướng hội tụ 72

1. Bài toán phân loại nhi phân trong không gian hai chiều 75
2. Các đường thẳng trong không gian hai chiều 76
3. [Biểu diễn perceptron và hồi quy tuyến tính dưới dạng mạng neuron.](#bookmark835) 80
4. Cấu trúc của một neuron thần kinh sinh học 81
5. PLA có thể có nhiều nghiệm 82
6. Ví dụ về kết quả thi dựa trên số giờ ôn tập 84
7. Một vài ví dụ về các hàm kích hoạt khác nhau 84
8. Nghiệm của hồi quy logistic 90
9. Ví dụ về dữ liệu và hàm sigmoid trong không gian hai chiều 90
10. Ví dụ về hồi quy logistic với dữ liệu hai chiều 91
11. Các chữ số bi phân loại lỗi trong bài toán phân loại nhi phân với

hai chữ số không và một 94

8.9 Mô hình neural network cho các kỹ thuật sử dụng các bộ phân loại nhi phân cho bài toán phân loại đa lớp 97

1. So sánh hàm entropy chéo và hàm bình phương khoảng cách 104
2. [Ví dụ về sử dụng hồi quy softmax cho nam lớp dữ liệu. (a) Giá](#bookmark1036)

[tri hàm mất mát qua các epoch. (b) Kết quả phân loại cuối cùng.](#bookmark1036) 109

1. Dùng PLA sẽ có nhiều đáp án 113
2. [Ý tưởng của SVM 114](#bookmark1341)
3. Minh họa véc tơ hỗ trợ, lề 114
4. Nghiệm tìm được với SVM 121
5. Ví dụ với ba cụm dữ liệu trong không gian hai chiều 127
6. Thuật toán phân cụm *K*-means qua các vòng lặp 133
7. 200 mẫu ngẫu nhiên trong bộ cơ sở dữ liệu MNIST 134
8. Ví dụ về chữ số 7 và giá tri các pixel của nó 134
9. Các tâm cụm (cột đầu) và 20 điểm ngẫu nhiên trong mỗi cụm.

Các chữ số trên mỗi hàng thuộc vào cùng một cụm 136

1. Tâm và 20 điểm gần tâm nhất của mỗi cụm 136
2. Tách vật thể trong ảnh 137
3. Kết quả ảnh khi phân cụm 138

11.10Các giá tri khởi tạo ban đầu khác nhau dẫn đến các nghiệm khác

nhau 140

1. Các cụm có số lượng chênh lệnh 141
2. [Một ví dụ về việc phân cụm *K*-means không hoạt động hiệu quả.](#bookmark1282) 141
   1. Ví dụ về phương sai của dữ liệu 144
   2. Ý tưởng chính của PCA 144
   3. Minh họa PCA là phương pháp đổi trục 147
3. Ảnh được tái tại dùng khuôn mặt riêng 155
4. PCA cho bài toán dò tìm điểm bất thường 156
5. [Ví dụ về ma trận tiện ích 162](#bookmark1434)
6. Ví dụ về hệ thống gợi ý 164
7. Ví dụ về ma trận tiện ích dựa trên số sao 173

DANH SÁCH BẢNG

0.1 BẢNG CÁC KÝ HIỆU SỬ DỤNG TRONG BÀI GIẢNG 4

1. Bảng dữ liệu về chiều cao và cân nặng của 15 người 34
2. [Ví dụ về nội dung của các van bản trong b ài toán Bắc hay Nam](#bookmark660) . . 49
3. Thời gian ôn thi và kết quả thi của 20 sinh viên 83

Bảng 0.1: BẢNG CÁC KÝ HIỆU SỬ DỤNG TRONG BÀI GIẢNG

|  |  |
| --- | --- |
| Ký hiệu | Y nghĩa |
| x, *y,* N; *k* | in nghiêng, thường hoặc hoa, là các số vô hướng |
| **x**, **y** | in đậm, chữ thường, là các vector |
| **X**; **Y** | in đậm, chữ hoa, là các ma trận |
| R | tập hợp các số thực |
| N | tập hợp các số tự nhiên |
| C | tập hợp các số phức |
| Rm | tập hợp các vector thực có m phần tử |
| RmXn | tập hợp các ma trận thực có m hàng, *n* cột |
| Sn | tập hợp các ma trận vuông đối xứng bậc n |
| S+ | tập hợp các ma trận nửa xác định dương bậc n |
| S++ | tập hợp các ma trận xác định dương bậc n |
| 2 | phần tử thuộc tập hợp |
| 9 | tồn tại |
| 8 | mọi |
|  | ký hiệu là/bởi. Ví dụ *a , f* (x) nghĩa là “ký hiệu f (x) bởi a”. |
| Xí | phần tử thứ i (tính từ 1) của vector **x** |
| sgn(x) | hàm xác định dấu. Bằng 1 nếu x > 0, bằng -1 nếu x < 0. |
| exp(x) | *ex* |
| log(x) | logarit *tự nhiên* của số thực dương x |
| argmin *f* (x)  *x* | giá trị của x đe hàm f (x) đạt giá trị nhỏ nhất |
| argmax f (x)  *x* | giá trị của x đe hàm f (x) đạt giá trị lớn nhất |
| *aíj* | phần tử hàng thứ i, cột thứ *j* của ma trận **A** |
| **A**T | chuyển vị của ma trận **A** |
| **A**H | chuyển vị liên hợp (Hermitian) của ma trận phức **A** |
| **A** | nghịch đảo của ma trận vuông **A**, nếu tồn tại |
| **A**y | giả nghịch đảo của ma trận không nhất thiết vuông **A** |
| **A** ; | chuyển vị của nghịch đảo của ma trận **A**, nếu tồn tại |
| llxllp | norm của vector **x** |
| IIAIIF | Frobenius norm của ma trận **A** |
| diag(**A**) | đường chéo chính của ma trận **A** |
| trace(**A**) | trace của ma trận **A** |
| det(**A**) | định thức của ma trận vuông **A** |
| rank(**A**) | hạng của ma trận **A** |
| o.w | *otherwise* - trong các trường hợp còn lại |
| @x | đạo hàm của hàm số f theo x *2* R |
| Vxf | gradient của hàm số f theo **x** (**x** là vector hoặc ma trận) |
| vXf | gradient bậc hai của hàm số f theo **x**, còn được gọi là *Hesse* |
| © | Hadamard product (elemenwise product). Phép nhân từng phần tử của hai vector hoặc ma trận cùng kích thước. |
| oc | tỉ lệ với |
| — | đường nét liền |
| ..... | đường nét đứt |
|  | đường nét chấm (đường chấm chấm) |
| —. —. . | đường chấm gạch |
|  | nền chấm |
|  | nền sọc chéo |

Chương I

TỔNG QUAN

Chương này trình bày một số lý thuyết can bản và thuật ngữ chuyên ngành về học máy. Học viên cần nắm chắc các kiến thức sau:

1. Khái niệm về học máy, khi nào nên dùng các thuật toán học máy. Các ứng dụng tiêu biểu.
2. Phân loại các kiểu hệ thống học máy: Có giám sát, không giám sát, bán giám sát, tang cường...
3. Những thách thức chính của học máy
4. Đánh giá và tinh chỉnh mô hình
5. Quy trình của một dự án học máy.
6. Các kỹ thuật xây dựng đặc trưng.

Nội dung chương 1 được học trong 2 bài, mỗi bài 2 tiết. Đề nghi làm hết bài tập cuối chương và đọc thêm tài liệu. Cài đặt môi trường chuẩn bi lập trình.

***Bài 1***

Các khái niệm cơ bản

1. Khái niệm học máy

Học máy là một môn khoa học về cách lập trình máy tính để chúng có thể học từ dữ liệu. Hay theo một số đinh nghĩa kỹ thuật hơn dưới đây:

*Học máy là lĩnh vực nghiên cứu nhằm giúp máy tính cố khả năng học mà không cần lập trình một cách tường minh.*

*Một chương trình máy tính được gọi là “học tập” từ kinh nghiệm E để hoàn thành nhiệm vụ T với hiệu quả được đo bằng phép đánh giá P, nếu hiệu quả của nố khi thực hiện nhiệm vụ T, khi được đánh giá bởi P, cải thiện theo kinh nghiệm E.*

Nói chung, một thuật toán học máy là một thuật toán có khả nang học tập từ dữ liệu. Đe xây dựng một chương trình máy tính có khả nang học, ta cần xác đinh rõ ba yếu tố: Nhiệm vụ, phép đánh giá, và đặc biệt là kinh nghiệm E, là nguồn dữ liệu huấn luyện. Nếu không có dữ liệu và thuật toán học thì không thể xây dựng một chương trình học máy.

1. Dữ liệu

Các *nhiệm vụ* trong học máy được mô tả thông qua việc một hệ thống xử lý một điểm dữ liệu đầu vào như thế nào. Một mô hình học máy như một hàm toán học mà đầu vào là dữ liệu, đầu ra là kết quả chúng ta mong muốn hiểu biết về dữ liệu. Một điểm dữ liệu có the là một bức ảnh, một đoạn âm thanh, một van bản, hoặc một tập các hành vi của người dùng trên Internet. Đe chương trình máy tính có thể học được, các điểm dữ liệu thường được đưa về dạng tập hợp các con số mà mỗi số được gọi là một *đặc trưng.*

Vậy mọi loại dữ liệu đều biểu diễn dưới dạng *vector đặc trưng,* ma trận hay tensor. Vector đặc trưng của một điểm dữ liệu thường được ký hiệu là **x** 2 trong đó *d* là số lượng đặc trưng. Các mảng nhiều chiều được hiểu là đã bi *vector hoá* thành mảng một chiều. Kỹ thuật xây dựng vector đặc trưng cho dữ liệu được trình bày cụ the hơn trong bài 2.

*Kinh nghiệm* trong học máy là bộ dữ liệu được sử dụng để xây dựng mô hình. Trong quá trình xây dựng mô hình, bộ dữ liệu thường được chia ra làm ba tập dữ liệu không giao nhau: Tập huấn luyện, tập kiểm tra, và tập xác thực.

*Tập huấn luyện* (training set) bao gồm các điểm dữ liệu được sử dụng trực tiếp trong việc xây dựng mô hình. *Tập kiểm tra* (test set) gồm các dữ liệu được dùng để đánh giá hiệu quả của mô hình. Đe đảm bảo tính phổ quát, dữ liệu kiểm tra không được sử dụng trong quá trình xây dựng mô hình. Điều kiện cần để một mô hình hiệu quả là kết quả đánh giá trên cả tập huấn luyện và tập kiểm tra đều cao. Tập kiểm tra đại diện cho dữ liệu mà mô hình chưa từng thấy, có thể xuất hiện trong quá trình vận hành mô hình trên thực tế.

Để tang hiệu quả của mô hình trên dữ liệu kiểm tra, người ta thường sử dụng một tập dữ liệu nữa được gọi là *tập xác thực* (validation set). Tập xác thực này được sử dụng trong việc đánh giá sai số kiểm tra và từ đó lựa chọn các siêu tham số mô hình sao cho tốt nhất có the.

*Phép đánh giá P* là thước đo quá trình cải thiện tham số mô hình để biết rằng giữa đầu ra mong muốn từ mô hình và thực tế có khớp không. Phép đánh giá thường là sai số và chúng ta cần tối ưu hàm mất mát sao cho sai số nhỏ nhất có thể.

1. Các b ài toán cơ bản trong học máy
2. Bài toán phân lớp

Bài toán *Phân lớp* hay phân loại là một trong những bài toán được nghiên cứu nhiều nhất trong học máy. Trong bài toán này, chương trình được yêu cầu xác đinh *lớp/nhãn* của một điểm dữ liệu trong số *C* nhãn khác nhau. Cặp (dữ liệu, nhãn) được ký hiệu là (**x**, y) với *y* nhận một trong C giá tri trong tập đích *y*. Việc xây dựng mô hình tương đương với việc đi tìm hàm số *f* ánh xạ một điểm dữ liệu **x** vào một phần tử *y 2 y* : *y* = *f* (**x**) trên tập dữ liệu huấn luyện đã biết sao cho sau đó mô hình được áp dụng trên các dữ liệu mới với sai số chấp nhận được. Khi đó ta xác đinh được nhãn của dữ liệu mới.

*Ví dụ*: Bài toán xác đinh email rác là một bài toán phân lớp trong học máy . Trong đó,

* Nhiệm vụ *T* là xác một email mới trong hộp thư đến là email rác hay không.
* Phép đánh giá *P* có thể đinh nghĩa là tỉ lệ email rác được xác đinh theo mô hình và đúng là email rác trong thực tế.
* Kinh nghiệm *E* là cặp các (email, nhãn) thu thập được trước đó. Dữ liệu này dùng đe huấn luyện mô hình, tìm ra hàm *f* (**x**).

Có một biến thể nhỏ ở đầu ra của hàm số *f* (**x**) khi đầu ra không phải là một số mà là một vector **y** 2 Rc trong đó y**y** = (y1,y2, *■■■,yc, ■■■yc*) và *yc* là xác suất để điểm dữ liệu **x** rơi vào lớp thứ c. Lớp được chọn cuối cùng là lớp có xác suất rơi vào là cao nhất. Việc sử dụng xác suất này đôi khi rất quan trọng, nó giúp chỉ ra *độ chắc chắn* của mô hình. Nếu xác suất cao nhất là cao hơn nhiều so với các xác suất còn lại, ta nói mô hình có độ chắn chắn là cao khi phân lớp điểm dữ liệu **x**. Ngược lại, nếu độ chênh lệch giữa xác suất cao nhất và các xác suất tiếp theo là nhỏ, thì khả nang mô hình đã phân loại nhầm là cao hơn. Hồi quy logistic là một ví dụ về phân lớp sử dụng xác suất.

1. Bài toán hồi quy

Nếu tập đích *y* gồm các giá tri là các số thực thì bài toán được gọi là *hồi quy.* Trong bài toán này, ta cần xây dựng một hàm số *f* : ! R, ánh xạ mỗi véc tơ

dữ liệu với nhãn y 2 Y, đầu ra mong muốn dự đoán.

*Ví dụ*: Ước lượng giá của một can nhà rộng *x* m2, có y phòng ngủ và cách trung tâm thành phố *z* km. Ta mong muốn xây dựng hàm dự đoán dạng *f* : R3 ! R, với giá = f *(x,y,z).*

Bài toán hồi quy có thể mở rộng ra việc dự đoán nhiều đầu ra cùng một lúc, khi đó, hàm cần tìm sẽ là *f* : Rd ! Rm. Một ví dụ là bài toán tạo ảnh độ phân giải cao từ một ảnh có độ phân giải thấp hơn. Khi đó, việc dự đoán giá tri các điểm trong ảnh đầu ra là một bài toán hồi quy nhiều đầu ra.

1. Bài toán máy dịch

Trong bài toán *máy dịch,* chương trình máy tính được yêu cầu dich một đoạn van từ một ngôn ngữ sang một ngôn ngữ khác. Dữ liệu huấn luyện là các cặp van bản song ngữ. Các van bản này có thể chỉ gồm hai ngôn ngữ đang xét hoặc có thêm các ngôn ngữ trung gian.

*Ví dụ*: Google translate là một ứng dụng (với nhiều kỹ thuật học sâu khác được áp dụng) học máy trong lĩnh vực máy dich.

1. Bài toán phân cụm

*Bài toán phân cụm* là bài toán chia tập dữ liệu *X* thành các cụm nhỏ dựa trên sự liên quan nào đó giữa các dữ liệu trong mỗi cụm. Trong bài toán này, dữ liệu huấn luyện không có nhãn, mô hình tự phân chia dữ liệu thành các cụm khác nhau dựa vào độ đo sự gần gũi của các đặc trưng dữ liệu.

*Ví dụ*: Phân cụm khách hàng dựa trên hành vi mua hàng. Dựa trên việc mua bán và theo dõi của người dùng trên một trang web thương mại điện tử, mô hình có thể phân người dùng vào các cụm theo sở thích mua hàng. Từ đó, mô hình có thể quảng cáo các mặt hàng mà người dùng có thể quan tâm trên cơ sở những mặt hàng này người cùng nhóm đã mua, đã quan tâm. Những đặc trưng của khách hàng cũng cần được véc tơ hóa và sự gần gũi về sở thích mua hàng được tính toán thông qua một độ đo.

1. Bài toán hoàn thiện dữ liệu

*Bài toán hoàn thiện dữ liệu* là bài toán dự đoán các trường dữ liệu còn thiếu trong dữ liệu đích. Nhiệm vụ của bài toán này là dựa trên mối tương quan giữa các điểm dữ liệu để dự đoán những giá tri còn thiếu. Các hệ thống khuyến nghi là một ví dụ điển hình của loại bài toán này. Hệ thống sẽ dự đoán sự quan tâm của các đối tượng (user) thể hiện bằng các thang điểm, số sao đánh giá mà họ chưa đánh giá các mặt hàng (item), dựa trên mối tương quan giữa các user với nhau hoặc các item với nhau. Các phần mềm khôi phục ảnh cũ, tô màu ảnh cũ sử dụng học máy cũng là những ứng dụng thuộc phạm vi hoàn thiện dữ liệu.

Ngoài ra, có nhiều bài toán học máy khác như *bài toán xếp hạng, bài toán giảm chiều dữ liệu, bài toán phát hiện bất thường....tùy* thuộc vào yêu cầu đầu ra, dữ liệu chúng ta có và khả nang giải quyết của thuật toán.

1. Các thuật toán học máy

Phân nhóm dựa theo phương thức học có các loại thuật toán học máy sau:

1. Học có giám sát

Một thuật toán học máy được gọi là *học có giám sát* nếu việc xây dựng mô hình dự đoán mối quan hệ giữa đầu vào và đầu ra được thực hiện dựa trên các cặp (đầu vào, đầu ra) đã biết trong tập huấn luyện. Nghĩa là dữ liệu huấn luyện đã có nhãn, là đầu ra mong muốn.

Các thuật toán phân lớp và hồi quy là hai ví dụ điển hình trong nhóm này.

Diễn giải theo toán học, học có giám sát xảy ra khi việc dự đoán quan hệ giữa đầu ra **y** và dữ liệu đầu vào **x** được thực hiện dựa trên các cặp {(**x**1, **y**1), (**x**2, **y**2),... , (**x**N,**y**N)} trong tập huấn luyện. Việc huấn luyện là việc xây dựng một hàm số *f* sao cho với mọi *i* = 1,2,..., *N*, *f (****x****i)* gần với ***y****i* nhất có thể. Hơn thế nữa, khi có một điểm dữ liệu **x** nằm ngoài tập huấn luyện, đầu ra dự đoán *f* (**x**) cũng gần với đầu ra thực sự **y**.

1. Học không giám sát

Trong các thuật toán học không giám sát, dữ liệu huấn luyện chỉ bao gồm các dữ liệu đầu vào **x** mà không có đầu ra tương ứng. Các thuật toán học máy có thể không dự đoán được đầu ra nhưng vẫn trích xuất được những thông tin quan trọng dựa trên mối liên quan giữa các điểm dữ liệu.

Các thuật toán giải quyết bài toán phân cụm và giảm chiều dữ liệu là các ví dụ điển hình của nhóm này. Trong bài toán phân cụm, có the mô hình không trực tiếp dự đoán được đầu ra của dữ liệu nhưng vẫn có khả nang phân các điểm dữ liệu có đặc tính gần giống nhau vào từng nhóm.

1. Học bán giám sát

Thuật toán học bán giám sát là nhóm thuật toán học máy mà tập huấn luyện bao gồm các cặp (đầu vào, đầu ra) và dữ liệu khác chỉ có đầu vào, nghĩa là chứa cả dữ liệu có nhãn và không có nhãn trong tập huấn luyện.

Thực tế cho thấy ngày càng nhiều thuật toán rơi vào nhóm này vì việc thu thập nhãn cho dữ liệu có chi phí cao và tốn thời gian. Chẳng hạn, chỉ một phần nhỏ trong các bức ảnh y học có nhãn vì quá trình gán nhãn tốn thời gian và cần sự can thiệp của các chuyên gia.

1. Học củng cố

Mô hình học cách ra quyết đinh bằng cách giao tiếp với môi trường xung quanh. Các thuật toán thuộc nhóm này liên tục ra quyết đinh và nhận phản hồi từ môi trường để tự củng cố hành vi. Nhóm các thuật toán này có tên *học cõng* cố.Chúng có thể không yêu cầu dữ liệu huấn luyện.

Việc huấn luyện một thuật toán học củng cố thông thường dựa trên một đại lượng được gọi là *điểm thưởng.* Mô hình cần tìm ra một thuật toán tối đa điểm thưởng đó qua rất nhiều lần chơi khác nhau. Học củng cố được áp dụng nhiều trong việc tạo ra các chương trình máy tính có khả nang tự chơi cờ, tự chơi game hay xe tự lái.

Ngoài ra còn có những cách thức phân loại khác như: Học dựa trên mẫu hay học dựa trên mô hình; Học theo batch hay học trực tuyến...

1. Hàm mất mát và tham số mô hình

Mỗi mô hình học máy được mô tả bởi bộ *các tham số mô hình.* Công việc của một thuật toán học máy là đi tìm các tham số mô hình tối ưu cho mỗi bài toán, nghĩa là đạt được tốt nhất theo phép đánh giá *P*. Việc đi tìm các tham số mô hình có liên quan mật thiết đến các phép đánh giá. Mục đích chính là đi tìm các tham số mô hình sao cho các phép đánh giá đạt kết quả tốt nhất. Trong bài toán phân loại, kết quả tốt có thể được hiểu là có ít điểm dữ liệu bi phân loại sai. Trong bài toán hồi quy, kết quả tốt là khi sự sai lệch giữa đầu ra dự đoán và đầu ra thực sự là nhỏ.

Quan hệ giữa một phép đánh giá và các tham số mô hình được mô tả thông qua một hàm số gọi là h *àm mất mát.* Hàm số này thường có giá tri nhỏ khi phép đánh giá cho kết quả tốt và ngược lại. Việc đi tìm các tham số mô hình sao cho phép đánh giá trả về kết quả tốt tương đương với việc tối thiểu hàm mất mát. Như vậy, việc xây dựng một mô hình học máy chính là việc đi giải một bài toán tối ưu. Quá trình đó được coi là quá trình *học* của *máy.*

Tập hợp các tham số mô hình thường được ký hiệu bằng ớ, hàm mất mát của mô hình được ký hiệu là L(ớ) hoặc *J*(ớ). Bài toán đi tìm tham số mô hình tương đương với bài toán tối thiểu hàm mất mát:

ớ\* = argmin L(ớ). (1.1)

*e*

Trong đó, ký hiệu argmin L(ớ) được hiểu là giá tri của ớ để hàm số L(ớ) đạt giá *e*

tri nhỏ nhất, cần phân biệt với giá tri nhỏ nhất của *L(ớ),* là giá tri mà ta ít quan tâm.

**Sai số huấn luyện**: Đại lượng này là mức độ sai khác giữa đầu ra thực và đầu ra dự đoán của mô hình. Trong nhiều trường hợp, giá tri này chính là hàm mất mát khi áp dụng lên dữ liệu huấn luyện. Hàm mất mát này cần có một thừa số -2—, trong đó Nhi là số điểm dữ liệu huấn luyện, để tính giá tri trung bình mất Nhi

mát trên mỗi điểm dữ liệu. Với các bài toán hồi quy, đại lượng này thường được xác đinh bởi *sai số trung bình bình phương*: với ||y — y 112 là chuẩn Euclid của véc tơ. Với các bài toán phân loại, có nhiều cách đánh giá mô hình trên các tập dữ liệu. Chúng ta sẽ dần thấy trong các chương, bài sau.

sai số huấn luyện =

2N, V Jly - yll2

tập huấn luyện

(1.2)

**Sai số kiểm tra**: Tương tự như trên, áp dụng mô hình tìm được vào dữ liệu kiểm tra. Chú ý rằng dữ liệu kiểm tra không được sử dụng khi xây dựng mô hình.

Với các mô hình hồi quy, đại lượng này thường được đinh nghĩa bởi

sai số kiểm tra =

*ĩkt* V Iiy - yii2

tập kiểm tra

(1.3)

Việc lấy trung bình là quan trọng vì lượng dữ liệu trong tập huấn luyện và tập kiểm tra có the chênh lệch nhau.

1. Những thách thức chính của học máy

Vì nhiệm vụ chính của học máy là chọn và huấn luyện một thuật toán trên một tập dữ liệu, nên hai vấn đề có thể xảy ra là thuật toán tệ và dữ liệu xấu. Các thách thức chính đến từ đây.

1. Các thách thức đến từ dữ liệu xấu

*Không đủ dữ liệu huấn luyện.* 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).

*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.

*Dữ liệu kém chất lượng.* Nếu tập huấn luyện chứa đầy lỗi, điểm ngoại lai và nhiễu (chẳng hạn như do sai số đo lường), hệ thống sẽ gặp nhiều khó khan trong việc phát hiện các khuôn mẫu ẩn, và ít có khả nang 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 đe 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 bi thiếu một số đặc trưng (ví dụ: 5% khách hàng của bạn không cung cấp tuổi của họ), ta cần quyết đinh giữa việc bỏ qua luôn thuộc tính này, bỏ qua những mẫu bi thiếu, điền vào các giá tri còn thiếu (ví dụ, bằng tuổi trung vi), hoặc huấn luyện hai mô hình: một mô hình với đặc trưng đó và một mô hình thì không.

*Các đặc trưng không liên quan.* 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 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

1. Các thách thức đến từ thuật toán tệ

*Quá khớp dữ liệu huấn luyện.* Trong học máy, khái niệm quá khớp có nghĩa là mô hình hoạt động tốt trên dữ liệu huấn luyện nhưng lại không có tính khái quát hóa. 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.* Mức độ điều chuẩn áp dụng trong quá trình học có thể được kiểm soát bởi một siêu tham số. 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. 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.

*Dưới khớp dữ liệu huấn luyện.* Dưới khớp ngược lại với quá khớp: nó xảy ra khi mô hình quá đơn giản để học được cấu trúc của dữ liệu. 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).

1. Quy trình thực hiện một dự án học máy

Danh mục các công việc trong một dự án học máy bao gồm các bước sau:

1. Đinh 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 bi 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.
8. Khởi chạy, giám sát, và bảo trì hệ thống.

Trong đó, mỗi bước trong quy trình lại có thể gợi ý các công việc thực hiện chi tiết như sau:

1. Định hình bài toán và xem xét bức tranh tổng thể

* Xác đinh mục tiêu theo phương diện kinh doanh.
* Giải pháp sẽ được sử dụng như thế nào?
* Các giải pháp hiện tại (nếu có) là gì?
* Đinh hình bài toán này như thế nào (có giám sát/không giám sát, trực tuyến/ngoại tuyến, v.v.)?
* Chất lượng của giải pháp sẽ được đo lường như thế nào?
* Thước đo chất lượng này có phù hợp với mục tiêu kinh doanh không?
* Chất lượng mô hình tối thiểu cần thiết để đạt được mục tiêu kinh doanh là bao nhiêu?
* Các bài toán tương tự là gì? Có thể sử dụng lại kinh nghiệm hoặc công cụ có sẵn được không?
* Yếu tố kiến thức chuyên môn có cần thiết không?
* Sẽ giải quyết bài toán theo cách thủ công như thế nào?
* Liệt kê những giả đinh đã đưa ra cho đến thời điểm hiện tại.
* Kiểm chứng các giả đinh nếu có thể.

1. Thu thập dữ liệu

*Hãy tự động hóa càng nhiều càng tết để có thể thu thập được dữ liệu mới một cách dễ dàng.*

* Liệt kê loại dữ liệu và số lượng cần thiết.
* Tìm và ghi lại nơi có thể thu thập được dữ liệu đó.
* Kiểm tra xem dữ liệu sẽ chiếm bao nhiêu dung lượng.
* Kiểm tra các nghĩa vụ pháp lý và xin quyền sử dụng nếu cần.
* Xin quyền truy cập.
* Tạo một môi trường làm việc (có đủ dung lượng lưu trữ).
* Thu thập dữ liệu.
* Chuyển đổi dữ liệu thành đinh dạng có thể dễ dàng thao tác (mà không làm thay ổi thông tin dữ liệu).
* Đảm bảo các thông tin nhạy cảm phải được bảo vệ hoặc xóa bỏ (ví dụ, dạng ẩn danh).
* Kiểm tra kích thước và dạng dữ liệu (dạng chuỗi thời gian, mẫu, không gian đia lý, v.v.).
* Lấy mẫu để tạo tập dữ liệu kiểm tra, để nó sang một bên và không sử dụng trong quá trình huấn luyện.

1. Khám phá dữ liệu

*Hãy cố gắng nhờ một chuyên gia trong ngành để hiểu rò hơn về dữ liệu khi thực hiện các bước này.*

* 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).
* Tạo một Jupyter notebook để ghi lại quá trình khám phá dữ liệu.
* Nghiên cứu từng thuộc tính và đặc điểm của nó:
* Tên
* Dạng dữ liệu (dạng hạng mục, dạng số nguyên/số thực, dạng bi chặn/không bi chặn, dạng van bản, dạng có cấu trúc, v.v.)
* Phần tram của những giá tri bi 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.)
* Đối với các tác vụ học có giám sát, hãy xác đinh (các) thuộc tính mục tiêu.
* Trực quan hóa dữ liệu.
* Nghiên cứu mối tương quan giữa các thuộc tính.
* Nghiên cứu cách bạn sẽ giải quyết bài toán theo cách thủ công.
* Xác đinh các phép biến đổi tiềm nang có thể áp dụng.
* Xác đinh dữ liệu bổ sung hữu ích (quay lại bước Thu thập Dữ liệu).
* Ghi lại những gì đã học được.

1. Chuẩn bị dữ liệu

*Làm việc trên các bản sao của dữ liệu thay vĩ tập dữ liệu gốc (giữ nguyên tập dữ liệu gốc). Viết hàm cho tất cả các phép biến đoi dữ liệu, vĩ 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 đoi 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 tri còn thiếu (ví dụ với 0, trung bình, trung vi, v.v.) hoặc loại bỏ hàng (hoặc cột) chứa các giá tri đó.

1. Lựa chọn đặc trưng (không bắt buộc):

* Loại bỏ các thuộc tính không hữu ích cho tác vụ.

1. 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 nang.
* Tổng hợp các đặc trưng thành đặc trưng mới tiềm nang.

1. Co giãn đặc trưng:

* Chuẩn hóa các đặc trưng.

1. Rút gọn danh sách các mô hình tiềm năng

* *Nếu tập dữ liệu lớn, 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).*
* *Hãy cố gắng tự động hóa các bước này nhiều nhất có thể.*
* 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.
* Đo lường và so sánh chất lượng của chúng. Với mỗi mô hình, sử dụng kiểm đinh N-fold, tính trung bình và độ lệch chuẩn của chất lượng mô hình trên N fold.
* Phân tích những biến quan trọng nhất của từng thuật toán.
* Phân tích những loại lỗi mà các mô hình gặp phải.
* Thực hiện nhanh một lượt lựa chọn và thiết kế đặc trưng.
* Thực hiện nhanh một hoặc hai lượt cả nam bước ở trên.
* Tạo danh sách rút gọn từ ba tới nam mô hình có tiềm nang nhất, ưu tiên các mô hình có các loại lỗi khác nhau.

1. Tinh chỉnh hệ thống

*Nên sử dụng càng nhiều dữ liệu càng tốt tại bước này, đặc biệt là trong giai đoạn cuối của quá trình tinh chỉnh. Cố gắng tự động hóa càng nhiều càng tốt.*

* Tinh chỉnh các siêu tham số sử dụng kiểm đinh chéo.
* 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.
* Khi đã tự tin với mô hình cuối cùng, hãy đo lường chất lượng trên tập kiểm tra để ước lượng lỗi tổng quát hóa. Đừng tinh chỉnh mô hình sau khi đo lường lỗi tổng quát hóa, sẽ bắt đầu bi quá khớp tập kiểm tra.

1. Trình b ày giải pháp

* Ghi chép lại những gì đã làm.
* Thực hiện một bài thuyết trình hay.
* Giải thích tại sao giải pháp đưa ra đạt được mục tiêu kinh doanh.
* Trình bày những điểm thú vi của hệ thống. Mô tả những thứ hoạt động và không hoạt động. Liệt kê các giả đinh cũng như các hạn chế của hệ thống.
* Đảm bảo những phát hiện chính được truyền đạt bằng trực quan đẹp mắt và phát biểu dễ nhớ.

1. Triển khai

* Chuẩn bi để triển khai giải pháp (đưa dữ liệu đầu vào vào hệ thống, viết unit test, v.v..).
* Viết mã giám sát để kiểm tra đinh kỳ chất lượng của hệ thống trong quá trình hoạt động và kích hoạt cảnh bảo khi chất lượng đi xuống.
* Huấn luyện lại các mô hình theo đinh kỳ với dữ liệu mới (tự động hóa nhiều nhất có thể).

Các kỹ thuật xây dựng đặc trưng

1. Giới thiệu

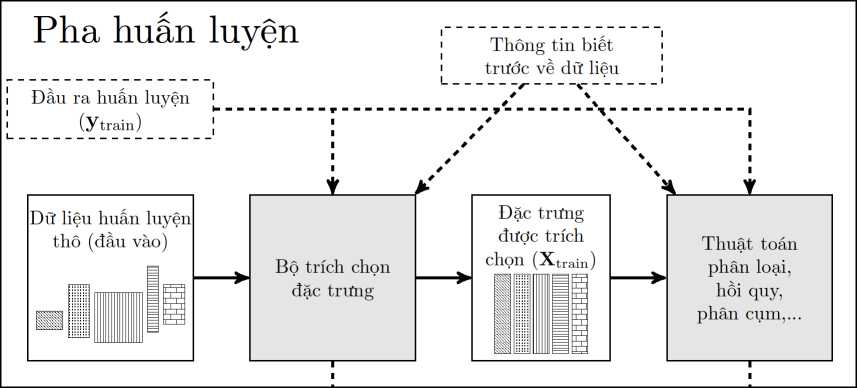
Mỗi điểm dữ liệu trong một mô hình học máy thường được biểu diễn bằng một vector được gọi là *vector đặc trưng.* Trong cùng một mô hình, các vector đặc trưng của các điểm thường có kích thước như nhau. Điều này là cần thiết vì các mô hình bao gồm các phép toán với ma trận và vector, các phép toán này yêu cầu dữ liệu có chiều phù hợp để có thể thực hiện được. Tuy nhiên, dữ liệu thực tế thường ở dạng thô với kích thước khác nhau hoặc kích thước như nhau nhưng số chiều quá lớn gây trở ngại trong việc lưu trữ. Vì vậy, việc lựa chọn, tính toán đặc trưng phù hợp cho mỗi bài toán là một bước quan trọng.

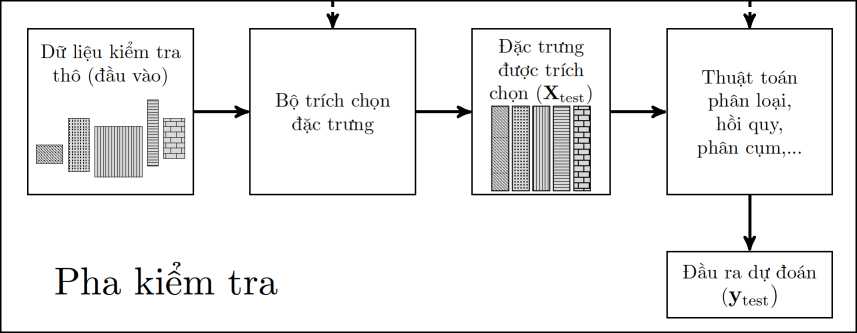
Khi làm việc với các bài toán học máy , nhìn chung ta chỉ có được dữ liệu thô chưa qua chỉnh sửa và chọn lọc. Ngoài ra, ta có thể phải loại bỏ những dữ liệu nhiễu và đưa dữ liệu thô với kích thước khác nhau về cùng một chuẩn. Dữ liệu chuẩn này phải đảm bảo giữ được những thông tin đặc trưng của dữ liệu thô ban đầu. Không những thế, ta cần thiết kế những phép biến đổi đe có những đặc trưng phù hợp cho từng bài toán. Quá trình quan trọng này được gọi là *trích chọn đặc trưng.*

Để có cái nhìn tổng quan, chúng ta cần đặt bước trích chọn đặc trưng này trong cả quy trình xây dựng một mô hình học máy .

1. Mô hình chung cho các bài toán học máy

Phần lớn các mô hình học máy có thể được minh hoạ trong Hình 2.1. Có hai pha lớn trong mỗi bài toán học máy là *pha huấn luyện* và *pha kiểm tra.* Pha huấn luyện xây dựng mô hình dựa trên dữ liệu huấn luyện. Dữ liệu kiểm tra được sử dụng để đánh giá hiệu quả mô hình.





**Hình 2.1.** Mô hình chung trong các bài toán học máy

1. Pha huấn luyện

Có hai khối cần được thiết kế: Bộ trích chọn đặc trưng và Lựa chọn thuật toán huấn luyện.

Khối trích chọn đặc trưng có nhiệm vụ tạo ra một vector đặc trưng cho mỗi điểm dữ liệu đầu vào. Vector đặc trưng này thường có kích thước như nhau, bất kể dữ liệu đầu vào có kích thước như thế nào.

Đầu vào của khối trích chọn đặc trưng có thể là các yếu tố sau:

*• Dữ liệu huấn luyện đầu vào ở dạng thô* bao gồm tất cả các thông tin ban đầu. Ví dụ, dữ liệu thô của một ảnh là giá tri của từng điểm ảnh, của một van bản là từng từ, từng câu; của một file âm thanh là một đoạn tín hiệu; của thời tiết là thông tin về hướng gió, nhiệt độ, độ am không khí,... Dữ liệu thô này thường không ở dạng vector, không có số chiều như nhau hoặc một vài thông tin bị khuyết. Thậm chí chúng có thể có số chiều như nhau nhưng rất lớn.

* *Dữ liệu huấn luyện đầu ra*: dữ liệu này có thể được sử dụng hoặc không. Trong các thuật toán học không giám sát, ta không biết đầu ra nên hiển nhiên không có giá trị này. Trong các thuật toán học có giám sát, đôi khi dữ liệu này cũng không được sử dụng. Ví dụ, việc giảm chiều dữ liệu có thể không cần sử dụng dữ liệu đầu ra. Nếu dữ liệu đầu vào đã là các vector cột cùng chiều, ta chỉ cần nhân vào bên phải của chúng một ma trận chiếu ngẫu nhiên. Ma trận này có số hàng ít hìn số cột để đảm bảo số chiều thu được nhỏ hìn số chiều ban đầu. Việc làm này mặc dù làm mất đi thông tin, trong nhiều trường hợp vẫn mang lại hiệu quả vì đã giảm được lượng tính toán ở phần sau. Đôi khi ma trận chiếu không phải là ngẫu nhiên mà có thể được *học* dựa trên toàn bộ dữ liệu thô ban đầu.

Trong nhiều trường hợp khác, dữ liệu đầu ra của tập huấn luyện cũng được sử dụng để tạo bộ trích chọn đặc trưng. Việc giữ lại nhiều thông tin không quan trọng bằng việc giữ lại các thông tin có ích.

* *Các thông tin biết trước về dữ liệu*: Ngoài dữ liệu huấn luyện, các thông tin biết trước ngoài lề cũng có tác dụng trong việc xây dựng bộ trích chọn đặc trưng. Chẳng hạn, có thể dùng các bộ lọc để giảm nhiễu nếu dữ liệu là âm thanh, hoặc dùng các bộ dò cạnh để tìm ra cạnh của các vật thể trong dữ liệu ảnh. Nếu dữ liệu là ảnh các tế bào và ta cần đưa ảnh về kích thước nhỏ hìn, ta cần lưu ý về độ phân giải của tế bảo của ảnh trong kích thước mới. Ta cần xây dựng một bộ trích chọn đặc trưng phù hợp với từng loại dữ liệu.

Sau khi xây dựng bộ trích chọn đặc trưng, dữ liệu thô ban đầu được đưa qua và tạo ra các vector đặc trưng tưìng ứng gọi là *đặc trưng đã trích xuất.* Những đặc trưng này được dùng để huấn luyện các thuật toán học máy chính như phân loại, phân cụm, hồi quy,... trong khối thuật toán.

1. Pha kiểm tra

ở pha kiểm tra, vector đặc trưng của một điểm dữ liệu thô mới được tạo bởi bộ trích chọn đặc trưng thu được từ pha huấn luyện. Vector đặc trưng này được đưa vào thuật toán chính đã tìm được để đưa ra quyết tra. Có một lưu ý quan trọng là khi xây dựng bộ trích chọn đặc trưng và các thuật toán chính, ta không được sử dụng dữ liệu kiểm tra. Các công việc đó được thực hiện chỉ dựa trên dữ liệu huấn luyện.

* 1. Một số kỹ thuật trích chọn đặc trưng
     1. Trực tiếp lấy dữ liệu thô

Xét bài toán với dữ liệu là các bức ảnh xám có kích thước *m* X *n* điểm ảnh. Cách đơn giản nhất để tạo ra vector đặc trưng cho bức ảnh này là xếp chồng các cột của ma trận điểm ảnh để được một vector *m X n* phần tử. Vector này có thể được coi là vector đặc trưng với mỗi đặc trưng là giá tri của một điểm ảnh. Việc làm đơn giản này đã làm mất thông tin về vi trí tương đối giữa các điểm ảnh vì các điểm ảnh gần nhau theo phương ngang trong bức ảnh ban đầu không còn gần nhau trong vector đặc trưng. Tuy nhiên, trong nhiều trường hợp, kỹ thuật này vẫn mang lại kết quả khả quan.

* + 1. Lựa chọn đặc trưng

Đôi khi, việc trích chọn đặc trưng đơn giản là chọn ra các thành phần phù hợp trong dữ liệu ban đầu. Việc làm này thường xuyên được áp dụng khi một lượng dữ liệu thu được không có đầy đủ các thành phần hoặc dữ liệu có quá nhiều chiều mà phần lớn không mang nhiều thông tin hữu ích.

* + 1. Giảm chiều dữ liệu

Giả sử dữ liệu ban đầu là một vector **x** 2 RD, **A** là một ma trận trong *RdxD* và **z** = **Ax** 2 Rd. Nếu *d < D,* ta thu được một vector với số chiều nhỏ hơn. Đây là một kỹ thuật phổ biến trong giảm chiều dữ liệu. Ma trận **A** được gọi là *ma trận chiếu* (projection matrix), có the là một ma trận ngẫu nhiên. Tuy nhiên, việc chọn một ma trận chiếu ngẫu nhiên đôi khi mang lại kết quả tệ không mong muốn vì thông tin có thể bi thất thoát quá nhiều. Một phương pháp phổ biến để tối thiểu lượng thông tin mất đi có tên là *phân tích thành phần chính* (principal component analysis).

*Lưu ý:* Kỹ thuật xây dựng đặc trưng không nhất thiết luôn làm giảm số chiều dữ liệu, đôi khi vector đặc trưng có thể có có kích thước lớn hơn dữ liệu thô ban đầu nếu việc này mang lại hiệu quả tốt hơn.

* + 1. Túi từ

Túi từ là một kỹ thuật tạo véc tơ đặc trưng từ van bản. Chúng ta xây dựng từ điển gồm tất cả các từ được sử dụng trong bài toán. Sau đó xây dựng véc tơ đặc trưng của câu thông qua việc đếm số lần xuất hiện của từ trong câu xét trên từ điển đã có. *Ví dụ.* Xét hai câu

(1) "John likes to watch movies. Mary likes movies too."

và

(2) "John also likes to watch football games."

Dựa trên hai van bản này, ta có danh sách các từ được sử dụng, được gọi là *từ điển* (dictionary hoặc codebook) với mười *từ* như sau:

["John", "likes", "to", "watch", "movies", "also", "football", "games", " Mary", "too"]

Với mỗi van bản, ta sẽ tạo ra một vector đặc trưng có số chiều bằng 10, mỗi phần tử đại diện cho số từ tưìng ứng xuất hiện trong van bản đó. Với hai van bản trên, ta sẽ có hai vector đặc trưng:

1. [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
2. [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

Van bản (1) có một từ "John", hai từ "likes", không từ "also", không từ "football ",... nên ta thu được vector tưìng ứng như trên.

Có một vài điều cần lưu ý trong kỹ thuật túi từ:

* Với những ứng dụng thực tế, từ điển có số lượng từ lớn hìn rất nhiều, có thể đến cả triệu, như vậy vector đặc trưng thu được sẽ rất dài. Một van bản chỉ có một câu, và một tiểu thuyết nghìn trang đều được biểu diễn bằng các vector có kích thước như nhau.
* Có rất nhiều từ trong từ điển không xuất hiện trong một van bản. Như vậy các vector đặc trưng thu được thường có nhiều phần tử bằng không. Các vector đó được gọi là *vector thưa.* Để việc lưu trữ được hiệu quả hìn, ta không lưu mọi thành phần của một vector thưa mà chỉ lưu vi trí của các phần tử khác không và giá tri tưìng ứng. Chú ý rằng nếu có hìn một nửa số phần tử khác không, việc làm này lại phản tác dụng. Tuy nhiên, trường hợp này ít xảy ra vì hiếm có van bản chứa tới một nửa số từ trong từ điển.
* Các từ hiếm gặp được xử lý như thế nào? Một kỹ thuật thường dùng là thêm phần tử <Unknown> vào trong từ điển. Mọi từ không có trong từ điển đều được coi là <Unknown>.
* Tuy nhiên, những từ hiếm đôi khi lại mang những thông tin quan trọng nhất mà chỉ loại van bản đó có. Đây là một nhược điểm của túi từ. Có một phưìng pháp cải tiến giúp khắc phục nhược điểm này tên là *term frequency-inversedocument frequency* (TF-IDF) dùng để xác đinh tầm quan trọng của một từ trong một van bản dựa trên toàn bộ van bản trong cơ sở dữ liệu.

• Nhược điểm lớn nhất của túi từ là nó không mang thông tin về thứ tự của các từ, cũng như sự liên kết giữa các câu, các đoạn van trong van bản. Thứ tự của các từ trong van bản thường mang thông tin quan trọng. Ví dụ, ba câu sau đây: “Thịt nướng chi Tươi?”, “Thịt chi nướng tươi”, và “Nướng thit chi Tươi” khi được trích chọn đặc trưng bằng túi từ sẽ cho ra ba vector giống hệt nhau, mặc dù ý nghĩa khác hẳn nhau.

1. Chuẩn hoá vector đặc trưng

Các điểm dữ liệu đôi khi được đo đạc bằng những đơn vi khác nhau, chẳng hạn mét và feet. Đôi khi, hai thành phần của dữ liệu ban đầu chênh lệch nhau lớn, chẳng hạn một thành phần có khoảng giá tri từ 0 đến 1000, thành phần kia chỉ có khoảng giá tri từ 0 đến 1. Lúc này, chúng ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

*Chú ý*: Việc chuẩn hóa này chỉ được thực hiện khi vector dữ liệu đã có cùng chiều.

Sau đây là một vài phương pháp chuẩn hóa thường dùng.

1. Chuyển khoảng giá trị

Phương pháp đơn giản nhất là đưa tất cả các đặc trưng về cùng một khoảng, ví dụ [0,1] hoặc [—1,1]. Để muốn đưa đặc trưng thứ *i* của một vector đặc trưng **x** về khoảng [0,1], ta sử dụng công thức

*. Xi -* min(xi)

(2.1)

*X- —*

max(xi) — min(xi)

trong đó xi và xi lần lượt là giá tri đặc trưng ban đầu và giá tri đặc trưng sau khi được chuẩn hóa. min(xi), max(xi) là giá tri nhỏ nhất và lớn nhất của đặc trưng thứ *i* xét trên toàn bộ dữ liệu huấn luyện.

1. Chuẩn hoá theo phân phối chuẩn

Một phương pháp khác thường được sử dụng là đưa mỗi đặc trưng về dạng một phân phối chuẩn có kỳ vọng là 0 và phương sai là 1. Công thức chuẩn hóa là

xi —

*x i xi*

*ơi*

(2.2)

với xi, ơi lần lượt là kỳ vọng và độ lệch chuẩn của đặc trưng đó xét trên toàn bộ dữ liệu huấn luyện.

1. Chuẩn hoá các véc tơ về cùng đô dài 1

Một lựa chọn khác cũng được sử dụng rộng rãi là biến vector dữ liệu thành vector có độ dài Euclid bằng 1. Việc này có thể được thực hiện bằng cách chia mỗi vector đặc trưng cho chuẩn '2 của nó:

(2.3)

**l|x||**2

1. BÀI TẬP CUỐI CHƯƠNG

**Bài 1.** Đinh nghĩa của Học Máy là gì?

**Bài 2.** Liệt kê bốn loại bài toán mà Học Máy giải quyết tốt?

**Bài 3.** Tập huấn luyện đã gán nhãn là gì?

**Bài 4.** Hai tác vụ học có giám sát phổ biến nhất là gì?

**Bài 5.** Liệt kê bốn tác vụ học không giám sát phổ biến?

**Bài 6.** Chúng ta sẽ sử dụng loại thuật toán Học Máy nào để cho phép rô bốt đi lại trong các đia hình

**Bài 7.** Chúng ta sẽ sử dụng loại thuật toán nào để phân nhóm khách hàng thành nhiều nhóm?

**Bài 8.** Chúng ta sẽ đặt bài toán phát hiện thư rác là bài toán học có giám sát hay học không giám sát?

**Bài 9.** Hệ thống học trực tuyến là gì?

**Bài 10.** Thế nào là học ngoài bộ nhớ chính?

**Bài 11.** Loại thuật toán nào dựa vào phép đo độ tương đồng để đưa ra dự đoán?

**Bài 12.** Sự khác biệt giữa tham số mô hình và siêu tham số của thuật toán là gì?

**Bài 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?

**Bài 14.** Chúng ta có thể liệt kê bốn thách thức chính trong Học Máy không?

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

**Bài 16.** Tập kiểm tra là gì và tại sao chúng ta lại muốn sử dụng nó?

**Bài 17.** Mục đích của tập kiểm đinh là gì?

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

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

Chương II

CÁC THUẬT TOÁN HỌC CÓ GIÁM SÁT

Chương này trình bày các thuật toán học có giám sát điển hình, bao gồm các thuật toán phân lớp và hồi quy cũng như thuật toán tối ưu bổ trợ. Sau khi kết thúc chương này, sinh viên nắm chắc các kỹ thuật học có giám sát, biết huấn luyện các mô hình trên dữ liệu thực tế:

1. Hồi quy tuyến tính.
2. K lân cận.
3. Bộ phân loại Navie Bayes.
4. Hạ Gradient.
5. Thuật toán học Perceptron.
6. Hồi quy Logistic
7. Hồi quy Softmax
8. Máy véc tơ hỗ trợ

Nội dung chương 2 được học trong 8 bài, mỗi bài 2 tiết. Đề nghi làm hết bài tập cuối chương, bài tập thực hành và đọc thêm tài liệu theo hướng dẫn.

Hồi quy tuyến tính

*Hồi quy tuyến tính* là một thuật toán hồi quy mà đầu ra là một hàm số tuyến tính của đầu vào. Đây là thuật toán đơn giản nhất trong nhóm các thuật toán học có giám sát.

1. Giới thiệu

Xét bài toán ước lượng giá của một can nhà rộng x1 m2, có x2 phòng ngủ và cách trung tâm thành phố x3 km. Giả sử có một tập dữ liệu của 10000 can nhà trong thành phố đó. Liệu rằng khi có một can nhà mới với các thông số về diện tích x1, số phòng ngủ x2 và khoảng cách tới trung tâm x3, chúng ta có thể dự đoán được giá *y* của can nhà đó không? Nếu có thì hàm dự đoán *y* = *f* (**x**) sẽ có dạng như thế nào. ở đây, vector đặc trưng **x** = [x1,x2,x3]T là một vector cột chứa dữ liệu đầu vào, đầu ra *y* là một số thực dương.

Nhận thấy rằng giá nhà cao nến diện tích lớn, nhiều phòng ngủ và gần trung tâm thành phố. Từ đó, ta có thể mô hình đầu ra là một hàm đơn giản của đầu vào:

y ~ y = *f* (**x**) = *Wixi* + *W2x2* + *W3x3 = XT***w**, (3.1)

trong đó **w** = [w1,w2,w3]t là *vector trọng so* (weight vector) cần tìm. Mối quan hệ y f (**x**) như trong [(3.1)](#bookmark488) là một mối quan hệ tuyến tính.

Bài toán trên đây là bài toán dự đoán giá tri của đầu ra dựa trên vector đặc trưng đầu vào. Ngoài ra, giá tri của đầu ra có thể nhận rất nhiều giá tri thực dương khác nhau. Vì vậy, đây là một bài toán hồi quy. Mối quan hệ *y* = **x**T**w** là một mối quan hệ tuyến tính. Tên gọi *hồi quy tuyến tính* xuất phát từ đây.

1. Xây dựng và tối ưu hàm mất mát

Tổng quát, nếu mỗi điểm dữ liệu được mô tả bởi một vector đặc trưng *d* chiều **x** 2 hàm dự đoán đầu ra được viết dưới dạng

*T*

(3.2)

*y = W1X1* + *W2X2* + + *WdXd =* **x w**.

1. Sai số dự đoán

Sau khi xây dựng được mô hình dự đoán đầu ra như [(3.2)](#bookmark494), ta cần tìm một phép đánh giá phù hợp với bài toán. Với bài toán hồi quy nói chung, ta mong muốn sự sai khác e giữa đầu ra thực sự y và đầu ra dự đoán y là nhỏ nhất:

*2e = 2(y - ỳ) = 2(y - x* w) *.* (3-3)

ở đây, bình phương được lấy vì e = y — *y* có thể là một số âm. Việc sai số là nhỏ nhất có thể được mô tả bằng cách lấy tri tuyệt đối |e| = |y — y|. Tuy nhiên, cách làm này ít được sử dụng vì hàm tri tuyệt đối không khả vi tại gốc toạ độ, không thuận tiện cho việc tối ưu. Hệ số 1 sẽ bi triệt tiêu khi lấy đạo hàm của e theo tham số mô hình **w**.

1. Hàm mất mát

Điều tương tự xảy ra với tất cả các cặp dữ liệu *(****x****i, yì); i =* 1, 2,..., *N*, với *N* là số lượng dữ liệu trong tập huấn luyện. Việc tìm mô hình tốt nhất tương đương với việc tìm **w** để hàm số sau đạt giá tri nhỏ nhất:

1 X

L(**w**) = 2NE(yi - **x**T**w**)2. (3.4)

v i=1

Hàm số L(**w**) chính là hàm mất mát của mô hình hồi quy tuyến tính với tham số *ỡ =* **w**. Ta luôn mong muốn sự mất mát là nhỏ nhất, điều này có thể đạt được bằng cách tối thiểu hàm mất mát theo **w**:

**w**\* = argmin L(**w**).

(3.5)

**w**

**w**\* là nghiệm cần tìm của bài toán.

Trung bình sai số

*Trong học máy , hàm mất mát thường là trung bình cộng của sai số tại*

*mỗi điểm. Về mặt toán học, hệ số 2N không ảnh hưởng tới nghiệm của bài*

*toán. Tuy nhiên, việc lấy trung bình này quan trọng vì số lượng điểm dữ*

*liệu trong tập huấn luyện có thể thay đểi. Việc tính toán mất mát trên từng điểm dữ liệu sẽ hữu ích hơn trong việc đánh giá chất lượng mô hình. Ngoài ra, việc lấy trung bình cũng giúp tránh hiện tượng tràn số khi số lượng điểm*

*dữ liệu lớn.*

Trước khi xây dựng nghiệm cho bài toán tối ưu hàm mất mát, ta thấy rằng hàm số này có thể được viết gọn lại dưới dạng ma trận, vector, và chuẩn như sau: với **y** = *[y1 ,y2,... ,yN*]T, **X** = [**x**1, **x**2,..., **x**N]. Như vậy L(**w**) là một hàm số liên quan tới bình phương của chuẩn '2.

\_ , . 1 V *, rj!* .. 1

L(w) = 2^E (y ~x<w) =2n

*i=1*

|  |  | T |  | 2 |  |
| --- | --- | --- | --- | --- | --- |
| y1 |  | x1T |  |  |  |
| y2 | — | x2T | **w** | = | *2N*lly“XTwll2 (3.6) |
| yv\_ |  | \* |  | 2 |  |

1. Nghiệm của hồi quy tuyến tính

Nhận thấy rằng hàm mất mát L(**w**) có gradient tại mọi **w**. Giá tri tối ưu của **w** có thể tìm được thông qua việc giải phương trình đạo hàm của L(**w**) theo **w** bằng không. Gradient của hàm số này tương đối đơn giản:

N**X**(**X**T**w - y**) (3.7)

Phương trình gradient bằng không:

~~V~~~~^~~~~(w)~~ = **0** , **XX**T**w** = **Xy** (3.8)

Nếu ma trận vuông ***XX****T* khả nghịch, phương trình [(3.8)](#bookmark511) có nghiệm duy nhất **w** = (**XX**T)-1**Xy**.

Nếu ma trận **XX**T không khả nghịch, phương trình [(3.8)](#bookmark511) vô nghiệm hoặc có vô số nghiệm. Lúc này, một nghiệm đặc biệt của phương trình có thể được xác định dựa vào *giả nghịch đảo.* Người ta chứng minh được rằng với mọi ma trận **X**, luôn tồn tại duy nhất một giá trị **w** có chuẩn '2 nhỏ nhất giúp tối thiểu II**X**T**w — y** kF. Cụ thể, **w** = (**XX**T)y**Xy** trong đó (**XX**T)y là giả nghịch đảo của **XX**T. Giả nghịch đảo của một ma trận luôn tồn tại kể cả khi ma trận đó không vuông. Khi ma trận là vuông và khả nghịch, giả nghịch đảo chính là nghịch đảo. Tổng quát, nghiệm của bài toán tối ưu [(3.5)](#bookmark503) là

**w** = (**XX**T )y**Xy** (3.9)

Hàm số tính giả nghịch đảo của một ma trận bất kỳ có sẵn trong thư viện numpy.

1. Hệ số điều chỉnh

Hàm dự đoán đầu ra của hồi quy tuyến tính thường có thêm một *hệ số điều chỉnh* (bias) *b:*

*f* (**x**) = **x**T**w** + b (3.10)

Nếu b = 0, đường thẳng/mặt phẳng **y** = **x**T**w** + b luôn đi qua gốc toạ độ. Việc thêm hệ số b khiến mô hình linh hoạt hìn. Hệ số điều chỉnh này cũng là một tham số mô hình.

Để ý thấy rằng, nếu coi mỗi điểm dữ liệu có thêm một đặc trưng *x0* = 1, ta sẽ có *y =* ***xT w*** + b = *W1x1* + *w2x2* + • • • + *wdxd* + *bx0 =* ***x****T* **w** (3.11) trong đó **x** = *[x0,x1,x2,... ,xN*]T và **w** = *[b,w1,w2,... ,WN*]. Nếu đặt **X** = [**x**, **x**2,... , **x**N], ta có nghiệm của bài toán tối thiểu hàm mất mát

w = argmin-1-**||y -** *XTw* 112 = (X*XT*)tX**y** (3.12)

**w**

Kỹ thuật thêm một đặc trưng *x0* = 1 vào vector đặc trưng và ghép hệ số điều chỉnh b vào vector trọng số **w** như trên còn được gọi là *thủ thuật gộp hệ số điều chỉnh* (bias trick).

1. Ví dụ trên Python
2. Bài toán

Xét một ví dụ đìn giản có thể áp dụng hồi quy tuyến tính. Chúng ta sẽ so sánh nghiệm của bài toán khi giải theo phưìng trình [(3.12)](#bookmark517) và nghiệm tìm được khi dùng thư viện scikit-learn của Python.

Giả sử có dữ liệu cân nặng và chiều cao của 15 người trong Bảng 3.1. Dữ liệu của hai người có chiều cao 155 cm và 160 cm được tách ra làm tập kiểm tra, phần còn lại tạo thành tập huấn luyện.

Bài toán đặt ra là liệu có thể dự đoán cân nặng của một người dựa vào chiều cao của họ không? Có thể thấy là cân nặng thường tỉ lệ thuận với chiều cao, vì vậy hồi quy tuyến tính là một mô hình phù hợp.

Bảng 3.1: Bảng dữ liệu về chiều cao và cân nặng của 15 người

|  |  |  |  |
| --- | --- | --- | --- |
| **Chiều cao (cm)** | **Cân nặng (kg)** | **Chiều cao (cm)** | **Cân nặng (kg)** |
| 147 | 49 | 168 | 60 |
| 150 | 50 | 170 | 72 |
| 153 | 51 | 173 | 63 |
| 155 | 52 | 175 | 64 |
| 158 | 54 | 178 | 66 |
| 160 | 56 | 180 | 67 |
| 163 | 58 | 183 | 68 |
| 165 | 59 |  |  |

1. Hiển thị dữ liệu trên đồ thị

Trước tiên, ta khai báo dữ liệu huấn luyện.

**from** future **import** print\_function

**import** numpy as np

**import** matplotlib.pyplot as plt

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | = np.array([[147, 150 | , 153, 158, | 163, | 165, 168, 170, | 173, | 175, 178, 180 |
|  | 183]]).T *# height* | *(cm), input* | *data,* | *each row is a* | *data* | *point* |
| *#* | *weight (kg)* |  |  |  |  |  |
| y | = np.array([ 49, 50, | 51, 54, 58 | , 59, | 60, 62, 63, 64 | , 66, | 67, 68]) |

Các điểm dữ liệu được minh hoạ bởi các điểm hình tròn trong Hình [3.1](#bookmark538). Ta thấy rằng dữ liệu được sắp xếp gần như theo một đường thẳng, vậy mô hình hồi quy tuyến tính sau đây có khả nang cho kết quả tốt, với w\_0 là hệ số điều chỉnh *b:*

(cân nặng) = w\_1\*(chiều cao) + w\_0

1. Nghiệm theo công thức

Tiếp theo, ta tìm các hệ số w\_1 và w\_0 dựa vào công thức [(3.12)](#bookmark517). Giả nghịch đảo của một ma trận A trong Python được tính bằng numpy.linalg.pinv(A).

* *Building Xbar*

one = np.ones((X.shape[0], 1))

Xbar = np.concatenate((one, X), axis =1) *# each row is one data point*

* *Calculating weights of the linear regression model*

A = np.dot(Xbar.T, Xbar)

b = np.dot(Xbar.T, y)

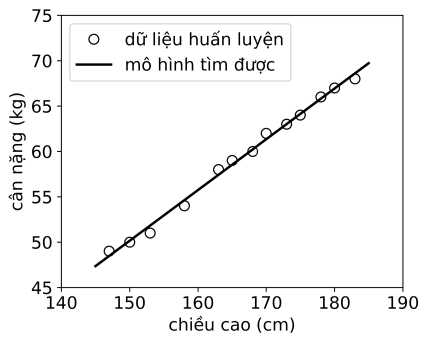
w = np.dot(np.linalg.pinv(A), b)

* *weights*

w\_0, w\_1 = w[0], w[1]

Đường thẳng mô tả mối quan hệ giữa đầu vào và đầu ra được minh hoạ trong

Hình [3.1](#bookmark538). Ta thấy rằng các điểm dữ liệu nằm khá gần đường thẳng dự đoán. Vậy

**Hình 3.1.** Minh hoạ dữ liệu và đưồng thang xấp xỉ tìm được bởi hồi quy tuyến tính

mô hình hồi quy tuyến tính hoạt động tốt với tập dữ liệu huấn luyện. Bây giờ, chúng ta sử dụng mô hình này để dự đoán dữ liệu trong tập kiểm tra.

y1 = w\_1\*155 + w\_0

y2 = w\_1\*160 + w\_0

**print**('Input 155cm, true output 52kg, predicted output %.2fkg.'

%(y1) )

%(y2) )

**print**('Input 160cm, true output 56kg, predicted output %.2fkg.'

Kết quả:

Input 155cm, true output 52kg, predicted output 52.94kg.

Input 160cm, true output 56kg, predicted output 55.74kg.

Chúng ta thấy rằng đầu ra dự đoán khá gần đầu ra thực sự.

1. Nghiệm theo thư viện scikit-learn

Tiếp theo, chúng ta sẽ sử dụng thư viện scikit-learn để tìm nghiệm.

**from** sklearn **import** datasets, linear\_model

* *fit the model by Linear Regression*

regr = linear\_model.LinearRegression()

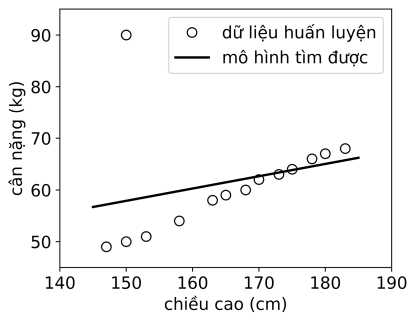
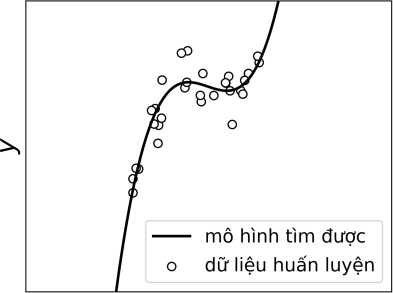
regr.fit(X, y) *# in scikit-learn, each sample is one row*

* *Compare two results*

**print**("scikit-learn's solution: w\_1 = ", regr.coef\_[0], "w\_0 = ",\ regr.intercept\_)

**print**("our solution : w\_1 = ", w[1], "w\_0 = ", w[0])

Kết quả:



hổi quy đa thức

*X*

**(a)**

**(b)**

**Hình 3.2.** (a) Hồi quy đa thức bậc ba (b) Hồi quy tuyến tính nhạy cảm vói nhiễu.

scikit-learn solution: w\_1 = [ 0.55920496] w\_0 = [-33.73541021] our solution : w\_1 = [ 0.55920496] w\_0 = [-33.73541021]

Chúng ta thấy rằng hai kết quả thu được là như nhau.

1. Thảo luận
2. Hồi quy đa thức

Hàm số *y* ~ *f* (**x**) = **x**T**w** + *b* là một hàm tuyến tính theo cả **w** và **x**. Hồi quy tuyến tính có thể áp dụng cho các mô hình chỉ cần tuyến tính theo **w**. Ví dụ

y « W1X1 + W2X2 + W3*X1* + W4 sin(x2) + W5X1X2 + Wo (3.13)

là một hàm tuyến tính theo **w** nhưng không tuyến tính theo **x**. Bài toán này vẫn có thể được giải bằng hồi quy tuyến tính. Với mỗi vector đặc trưng **x** = [x1 *,x2*]T, ta tính vector đặc trưng mới **x** = [x1; x2,x2, sin(x2);X1 x2]T rồi áp dụng hồi quy tuyến tính với dữ liệu mới này. Tuy nhiên, việc tìm ra các hàm số sin(x2) hay x1 x2 là tương đối không tự nhiên. *Hồi quy đa thức* thường được sử dụng nhiều hơn với các vector đặc trưng mới có dạng [1,x1 , x1*,...* ]T (đặc trưng đa thức). Một ví dụ về hồi quy đa thức bậc 3 được thể hiện trong Hình 3.2a.

1. Hạn chế của hồi quy tuyến tính

Hạn chế đầu tiên của hồi quy tuyến tính là nó rất *nhạy cảm với nhiễu.* Trong ví dụ về mối quan hệ giữa chiều cao và cân nặng bên trên, nếu có chỉ một cặp dữ liệu nhiễu (150 cm, 90kg) thì kết quả sẽ sai khác đi rất nhiều (xem Hình 3.2b).

Một kỹ thuật giúp tránh hiện tượng này là loại bỏ các nhiễu trong quá trình tìm nghiệm. Việc làm này có thể phức tạp và tương đối tốn thời gian. Có một cách khác giúp tránh công việc loại bỏ nhiễu là sử dụng *mất mát Huber.* Hồi quy tuyến tính với mất mát Huber được gọi là *hồi quy Huber,* được khẳng đinh là có khả nang kháng nhiễu tốt hơn.

Hạn chế thứ hai của hồi quy tuyến tính là nó *không biễu diễn được các mô hình phức tạp.* Mặc dù phương pháp này có thể được áp dụng nếu quan hệ giữa đầu ra và đầu vào là phi tuyến, mối quan hệ này vẫn đơn giản hơn nhiều so với các mô hình thực tế. Hơn nữa, việc tìm ra các đặc trưng *xị,* sin(x2), *X1* x2 như trên là không khả thi khi số chiều dữ liệu lớn lên.

1. Điều chuẩn

Có một kỹ thuật nhỏ giúp tránh trường hợp **XX**T không khả nghịch là biến nó thành **A** = **XX**T + A**I** với A là một số dương nhỏ và **I** là ma trận đơn vị với bậc phù hợp.

Ma trận **A** là khả nghịch vì nó là một ma trận xác định dương. Thật vậy, với mọi **w** = **0**,

**w**T**Aw** = **w**T(**XX**T + A**I**)**w** = **w**T**XX**T**w** + A**w**T**w** = **||X**T**w||**2 + A||**w||**2 > 0.

(3.14) Lúc này, nghiệm của bài toán là **y** = (**XX**T + A**I**) 1 **Xy**.

Xét hàm mất mát

L2(w) = 2N (l|y\_ XTw|2+A| w II2). (3.15)

Phương trình gradient theo **w** bằng không:

~~VL~~~~2~~~~(w)~~ = **0** , ^(**X**(**X**T**w - y**) + A**w**) = **0** , (**XX**T + A**I**)**w** = **Xy** (3.16)

Ta thấy **w** = (**XX**T + A**I**)\_1**Xy** chính là nghiệm của bài toán tối thiểu L2(**w**) trong [(3.15)](#bookmark562). Mô hình học máy với hàm mất mát [(3.15)](#bookmark562) còn được gọi là *hồi quy ridge.* Ngoài việc giúp phương trình gradient theo hệ số bằng không có nghiệm duy nhất, hồi quy ridge còn giúp mô hình tránh được quá khớp. Ngoài ra còn có các hồi quy dạng điều chuẩn khác như hồi quy Lasso, hồi quy ElasticNet.

1. Phương pháp tối ưu khác

Hồi quy tuyến tính là một mô hình đơn giản, lời giải cho phương trình đạo hàm bằng không cũng không phức tạp. Trong hầu hết các trường hợp, việc giải các phương trình gradient bằng không tương đối phức tạp. Tuy nhiên, nếu ta tính được đạo hàm của hàm mất mát, các tham số mô hình có thể được giải bằng một

phương pháp hữu dụng có tên *Hạ gradient - gradient descent (GD).* Trên thực tế, một vector đặc trưng có the có kích thước rất lớn, dẫn đến ma trận **XX**T cũng có kích thước lớn và việc tính ma trận nghịch đảo có thể không lợi về mặt tính toán. Gradient descent sẽ giúp tránh được việc tính ma trận nghịch đảo. Thuật toán GD tối ưu hàm mất mát L(**w**) bắt đầu bằng một điểm dự đoán **w**0, sau đó sử dụng quy tắc cập nhật:

**W**t+1 = ***Wt -*** 1/VWL(**w**t)

(3.17)

Chi tiết về thuật toán GD sẽ được trình bày trong bài [6](#bookmark683).

***Bài 4***

*K* lân cận

* 1. Giới thiệu
     1. K lân cận

*K lân cận - KNN* là một trong những thuật toán học có giám sát đơn giản nhất. Khi huấn luyện, thuật toán này gần như *không học* một điều gì từ dữ liệu huấn luyện mà *ghi nhớ* lại một cách máy móc toàn bộ dữ liệu đó. Mọi tính toán được thực hiện tại pha kiểm tra. KNN có thể được áp dụng vào các bài toán phân loại và hồi quy. KNN còn được gọi là một thuật toán *lười học.*

KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới chỉ dựa trên thông tin của K điểm dữ liệu gần nhất trong tập huấn luyện.

Với KNN, mọi điểm trong tập huấn luyện đều được mô hình mô tả một cách chính xác. Việc này khiến quá khớp dễ xảy ra với KNN.

Mặc dù có nhiều hạn chế, KNN vẫn là giải pháp đầu tiên nên nghĩ tới khi giải quyết một bài toán học máy. Khi làm các bài toán học máy nói chung, không có mô hình đúng hay sai, chỉ có mô hình cho kết quả tốt hơn. Chúng ta luôn cần một mô hình đơn giản để giải quyết bài toán, sau đó mới dần tìm cách tang chất lượng của mô hình.

* 1. Phân tích toán học

Không có hàm mất mát hay bài toán tối ưu nào cần thực hiện trong quá trình huấn luyện KNN. Mọi tính toán được tiến hành ở bước kiểm tra. Vì KNN ra quyết đinh dựa trên các điểm gần nhất nên có hai vấn đề ta cần lưu tâm. Thứ nhất, khoảng cách được đinh nghĩa như thế nào. Thứ hai, cần phải tính toán khoảng cách như thế nào cho hiệu quả.

Với vấn đề thứ nhất, mỗi điểm dữ liệu được thể hiện bằng một vector đặc trưng, khoảng cách giữa hai điểm chính là khoảng cách giữa hai vector đó. Có nhiều loại khoảng cách khác nhau tuỳ vào bài toán, nhưng khoảng cách được sử dụng nhiều nhất là khoảng cách Euclid.

Vấn đề thứ hai cần được lưu tâm hơn, đặc biệt với các bài toán có tập huấn luyện lớn và vector dữ liệu có kích thước lớn. Giả sử các vector huấn luyện là các cột của ma trận **X** 2 với *d* và *N* lớn. KNN sẽ phải tính khoảng cách từ một

điểm dữ liệu mới **z** 2 đến toàn bộ N điểm dữ liệu đã cho và chọn ra K khoảng cách nhỏ nhất. Nếu không có cách tính hiệu quả, khối lượng tính toán sẽ rất lớn.

Tiếp theo, chúng ta cùng thực hiện một vài phân tích toán học để tính các khoảng cách một cách hiệu quả. ở đây khoảng cách được xem xét là khoảng cách Euclid.

***Khoảng cách từ một điểm tới từng điểm trong một tập hợp***

Khoảng cách Euclid từ một điểm **z** tới một điểm ***x*** trong tập huấn luyện được đinh nghĩa bởi II **z** — x||2- Người ta thường dùng bình phương khoảng cách Euclid IIz — **x.**j 112 đe tránh phép tính can bậc hai. Việc bình phương này không ảnh hưởng tới thứ tự của các khoảng cách. Để ý rằng

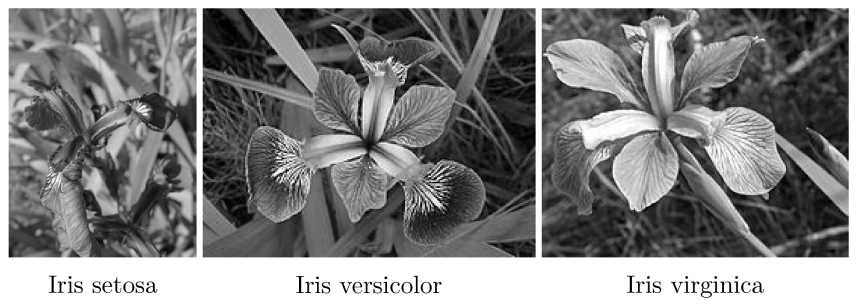
llz - xi|2 =(z - xi)T (z - x) = ||z||2 + 11x112 -2xT z (4.1)

Để tìm ra ***x*** gần với **z** nhất, số hạng đầu tiên có thể được bỏ qua. Hơn nữa, nếu có nhiều điểm dữ liệu trong tập kiểm tra, các giá tri IIXII2 có thể được tính và lưu trước vào bộ nhớ. Khi đó, ta chỉ cần tính các tích vô hướng **x**T**z**.

***Khoảng cách giữa từng cặp điểm trong hai tập hợp***

Thông thường, tập kiểm tra bao gồm nhiều điểm dữ liệu tạo thành một ma trận **Z**. Ta phải tính từng cặp khoảng cách giữa mỗi điểm trong tập kiểm tra và một điểm trong tập huấn luyện. Nếu mỗi tập có 1000 phần tử, có một triệu khoảng cách cần tính. Nếu không có phương pháp tính hiệu quả, thời gian thực hiện sẽ rất dài.

Khi làm việc trên python, chúng ta có thể sử dụng hàm cdist trong scipy.spatial .distance, hoặc hàm pairwise\_distances trong sklearn.metrics.pairwise. Các hàm này giúp tính khoảng cách từng cặp điểm trong hai tập hợp khá hiệu quả. Phần còn lại của bài này sẽ trực tiếp sử dụng thư viện scikit-learn cho KNN.



**Hình 4.1.** Ba loại hoa lan trong bộ co sở dữ liệu hoa Iris.

* 1. Ví dụ trên cơ sở dữ liệu Iris
     1. Bộ cơ sở dữ liệu hoa Iris

Bộ dữ liệu hoa Iris [(https://goo.gl/eUy83R)](https://goo.gl/eUy83R) là một bộ dữ liệu nhỏ. Bộ dữ liệu này bao gồm thông tin của ba nhãn hoa Iris khác nhau: Iris setosa, Iris virginica và Iris versicolor. Mỗi nhãn chứa thông tin của 50 bông hoa với dữ liệu là bốn thông tin: chiều dài, chiều rộng đài hoa, và chiều dài, chiều rộng cánh hoa. Hình [4.1](#bookmark587) là ví dụ về hình ảnh của ba loại hoa. Chú ý rằng các điểm dữ liệu không phải là các bức ảnh mà chỉ là một vector đ°c trưng bốn chiếu gồm các thông tin ỏ trên.

* + 1. Thí nghiệm

Trong phần này, 150 điểm dữ liệu được tách thành tập huấn luyện và tập kiểm tra. KNN dựa vào trông tin trong tập huấn luyện để dự đoán mỗi dữ liệu trong tập kiểm tra tưìng ứng với loại hoa nào. Kết quả này được đối chiếu với đầu ra thực sự để đánh giá hiệu quả của KNN.

Trước tiên, chúng ta cần khai báo vài thư viện. Bộ dữ liệu hoa Iris có sẵn trong thư viện scikit-learn.

**from** future **import** print\_function

**import** numpy as np

**from** sklearn **import** neighbors, datasets

**from** sklearn.model\_selection **import** train\_test\_split *# for splitting data* **from** sklearn.metrics **import** accuracy\_score *# for evaluating results*

iris = datasets.load\_iris() iris\_X = iris.data

iris\_y = iris.target

Tiếp theo, 130 mẫu dữ liệu được lấy ra ngẫu nhiên tạo thành tập huấn luyện, 20 mẫu còn lại được dùng để kiểm tra.

**print**('Labels:', np.unique(iris\_y))

*# split train and test*

np.random.seed(7)

X\_train, X\_test, y\_train, y\_test = train\_test\_split( iris\_X, iris\_y, test\_size=130)

**print**('Training size:', X\_train.shape[0], ', test size:', X\_test.shape[0])

Labels: [0 1 2]

Training size: 20 , test size: 130

Dòng np.random.seed(7) để đảm bảo kết quả chạy ở các lần khác nhau là giống nhau. Có thể thay 7 bằng một số tự nhiên 32 bit bất kỳ.

***Kết quả với 1NN***

Tới đây, ta trực tiếp sử dụng thư viện scikit-learn cho KNN. Xét ví dụ đầu tiên với K =1.

model = neighbors.KNeighborsClassifier(n\_neighbors =1, p = 2) model.fit(X\_train, y\_train) y pred = model.predict(X\_test)

**print**("Accuracy of 1NN: %.2f %%" %(100\*accuracy\_score(y\_test, y\_pred)))

Kết quả:

Accuracy of 1NN: 92.31 %

Kết quả thu được là 92.31% (tỉ lệ số mẫu được phân loại chính xác trên tổng số mẫu). ở đây, n\_neighbors = 1 chỉ ra rằng chỉ điểm gần nhất được lựa chọn, tức K =1, p = 2 chính là dùng chuẩn '2 đe tính khoảng cách.

**Kết quả với *7NN***

Như đã đề cập, 1NN rất dễ gây ra quá khớp. Đe hạn chế việc này, ta có thể tang lượng điểm lân cận lên, ví dụ 7 điểm, kết quả được xác đinh dựa trên đa số. model = neighbors.KNeighborsClassifier(n\_neighbors =7, p = 2) model.fit(X\_train, y\_train)

y pred = model.predict(X\_test)

**print**("Accuracy of 7NN with major voting: %.2f %%"\

%(100\*accuracy\_score(y\_test, y pred)))

Kết quả:

Accuracy of 7NN with major voting: 93.85 %

Nhận thấy rằng khi sử dụng nhiều điểm lân cận hơn, độ chính xác đã tang lên. Phương pháp dựa trên đa số trong lân cận còn được gọi là *bầu chọn đa số.*

***Đánh trọng số cho các điểm lân cận***

Trong kỹ thuật bầu chọn đa số phía trên, các điểm trong bảy điểm gần nhất đều có vai trò như nhau và giá tri “lá phiếu” của mỗi điểm này cũng như nhau. Cách bầu chọn này có the thiếu công bằng vì các điểm gần hơn nên có tầm ảnh hưởng lớn hơn. Đe thực hiện việc này, ta chỉ cần đánh trọng số khác nhau cho từng điểm trong bảy điểm gần nhất này. Cách đánh trọng số phải thoả mãn điều kiện điểm lân cận hơn được đánh trọng số cao hơn. Một cách đơn giản là lấy nghịch đảo của khoảng cách tới điểm lân cận. Trong trường hợp tồn tại khoảng cách bằng không, tức điểm kiểm tra trùng với một điểm huấn luyện, ta trực tiếp lấy đầu ra của điểm huấn luyện đó.

Đe thực hiện việc này trong scikit-learn, ta chỉ cần gán weights = 'distance'. Giá trị mặc định của weights là 'uniform', tương ứng với việc coi tất cả các điểm lân cận có giá trị bằng nhau như trong bầu chọn đa số.

model = neighbors.KNeighborsClassifier(n\_neighbors =7, p = 2, \ weights = 'distance') model.fit(X\_train, y\_train) y pred = model.predict(X\_test)

**print**("Accuracy of 7NN (1/distance weights): %.2f %%" %(100\*accuracy\_score( y\_test, y pred)))

Kết quả:

Accuracy of 7NN (1/distance weights): 94.62 %

Độ chính xác tiếp tục được tang lên.

***KNN với trọng số tự định nghĩa***

Ngoài hai cách đánh trọng số weights = ‘uniform’ và weights = ‘distance’, scikit- learn còn cung cấp cách đánh trọng số tùy chọn. Ví dụ, một cách đánh trọng số phổ biến khác thường được dùng là

/ — ||z - Xi ||2 \  
\ a2 *J*

*wi =* exp

trong đó *wi* là trọng số của điểm gần thứ *i* (**x**i) của điểm dữ liệu đang xét **z**, *a* là một số dương. Hàm số này cũng thỏa mãn điều kiện điểm càng gần **X** thì trọng số càng cao (cao nhất bằng 1). Với hàm số này, ta có thể lập trình như sau:

**def** myweight(distances): sigma2 = .4 *# we can change this number* **return** np.exp(-distances\*\*2/sigma2)

model = neighbors.KNeighborsClassifier(

n\_neighbors =7, p = 2, weights = myweight) model.fit(X\_train, y\_train) y pred = model.predict(X\_test)

**print**("Accuracy of 7NN (customized weights): %.2f %%"\ %(100\*accuracy\_score(y\_test, y\_pred)))

Kết quả:

Accuracy of 7NN (customized weights): 95.38 %

Kết quả tiếp tục tang lên một chút. Với từng bài toán, chúng ta có thể thay các thuộc tính của KNN bằng các giá tri khác nhau và chọn ra giá tri tốt nhất thông qua xác thực chéo.

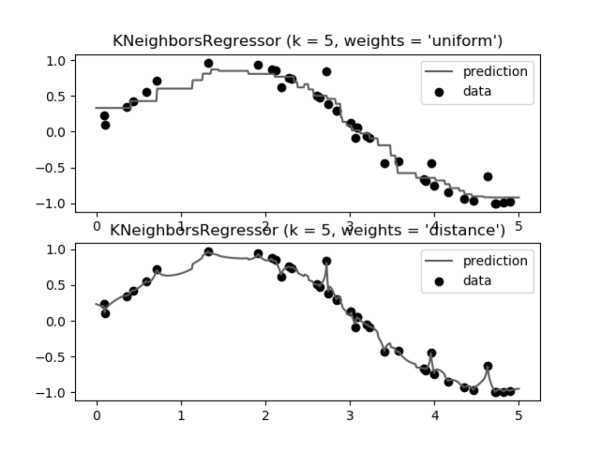
* 1. Thảo luận
     1. KNN cho bài toán hồi quy

Với bài toán hồi quy, chúng ta cũng hoàn toàn có thể sử dụng phương pháp tương tự: đầu ra của một điểm được xác đinh dựa trên đầu ra của các điểm lân cận và khoảng cách tới chúng. Giả sử **X**1,... , **X**K là K điểm lân cận của một điểm dữ liệu **z** với đầu ra tương ứng là *y1,... ,yK*. Giả sử các trọng số ứng với các lân cận này là *w1,..., WK*. Kết quả dự đoán đầu ra của **z** có thể được xác đinh bởi

*Wiyi* + *W2y2* + + *Wk Wk*

(4.2)

*W1* + W2 + + WK



**Hình 4.2.** KNN cho bài toán hồi quy.

Hình [4.2](#bookmark609) là một ví dụ về KNN cho hồi quy với K = 5, sử dụng hai cách đánh trọng số khác nhau. Ta có thể thấy rằng weights = 'distance' có xu hướng gây ra quá khớp.

* + 1. Ưu điểm của KNN
* Độ phức tạp tính toán của quá trình huấn luyện gần như bằng 0. Việc tính bình phương chuẩn '2 của mỗi điểm dữ liệu huấn luyện có thể được thực hiện trước trong bước này.
* Việc dự đoán kết quả của dữ liệu mới tương đối đơn giản sau khi đã xác đinh được các điểm lân cận.
* KNN không không cần giả sử về phân phối của từng nhãn.
  + 1. Nhược điểm của KNN
* KNN nhạy cảm với nhiễu khi K nhỏ.
* Khi sử dụng KNN, phần lớn tính toán nằm ở pha kiểm tra. Trong đó việc tính khoảng cách tới từng điểm dữ liệu huấn luyện tốn nhiều thời gian, đặc biệt là với các cơ sở dữ liệu có số chiều lớn và có nhiều điểm dữ liệu. K càng lớn thì độ phức tạp càng cao. Ngoài ra, việc lưu toàn bộ dữ liệu trong bộ nhớ cũng ảnh hưởng tới hiệu nang của KNN.

***Bài 5***

Bộ phân loại naive Bayes

1. Bộ phân loại naive Bayes

Xét các bài toán phân loại với *C* nhãn khác nhau. Thay vì tìm ra chính xác nhãn của mỗi điểm dữ liệu **x** 2 Rd, ta có thể đi tìm xác suất để kết quả rơi vào mỗi nhãn: *p(y =* c|**x**), hoặc viết gọn thành p(c|**x**). Biểu thức này được hiểu là xác suất để đầu ra là nhãn *c* biết rằng đầu vào là vector **x**. Nếu tính được biểu thức này, ta có the giúp xác đinh nhãn của mỗi điểm dữ liệu bằng cách chọn ra nhãn có xác suất rơi vào cao nhất:

c = argmax p(c|**x**)

(5.1)

*c2{1,...,C* }

Nhìn chung, khó có cách tính trực tiếp p(c|**x**). Thay vào đó, quy tắc Bayes thường được sử dụng:

* <ì fii') lì ‘I V '| — -I T\*ÍT 111 -I V I — Q 11 Q... u (-vr /' I 1 *í* 5Ã o')
* aigmax p(c|**x**) — aigmax -— — aigmax p(**x**|c)p(c) (5.2)

*c c* p(**x**) *c*

Dấu bằng thứ hai xảy ra theo quy tắc Bayes, dấu bằng thứ ba xảy ra vì p(**x**) ở mẫu số không phụ thuộc vào c. Tiếp tục quan sát, p(c) có thể được hiểu là xác suất để một điểm *bất kỳ* rơi vào nhãn c. Nếu tập huấn luyện lớn, p(c) có thể được xác đinh bằng phương pháp ước lượng hợp lý cực đại (MLE) - là tỉ lệ giữa số điểm thuộc nhãn c và số điểm trong tập huấn luyện. Nếu tập huấn luyện nhỏ, giá tri này có thể được xác đinh bằng phương pháp ước lượng hậu nghiệm cực đại (MAP).

Thành phần còn lại p(**x**|c) là phân phối của các điểm dữ liệu trong nhãn c. Thành phần này thường rất khó tính toán vì **x** là một biến ngẫu nhiên nhiều chiều. Đe có thể ước lượng được phân phối đó, tập huấn luyện phải rất lớn. Nhằm đơn giản hoá việc tính toán, người ta thường giả sử rằng các thành phần của biến ngẫu nhiên **x** độc lập với nhau khi đã biết c:

*d*

*p(****x****jc) = p(xi,x2, ..*. ,Xd|c) Ị Ị *p(xijc)* (5.3)

*i=1*

Giả thiết các chiều của dữ liệu độc lập với nhau là quá chặt và trên thực tế, ít khi tìm được dữ liệu mà các thành phần hoàn toàn độc lập với nhau. Tuy nhiên, giả thiết *ngây thơ (naive*) này đôi khi mang lại những kết quả tốt bất ngờ. Giả thiết về sự độc lập của các chiều dữ liệu này được gọi là *naive Bayes.* Một phương pháp xác đinh nhãn của dữ liệu dựa trên giả thiết này có tên là *phân loại naive Bayes (NBC).*

Nhờ giả thiết độc lập, NCB có tốc độ huấn luyện và kiểm tra rất nhanh. Việc này rất quan trọng trong các bài toán với dữ liệu lớn.

ở bước huấn luyện, các phân phối p(c) và *p(xi*|c), *i = 1,... ,d* được xác đinh dựa vào dữ liệu huấn luyện. Việc xác đinh các giá tri này có thể được thực hiện bằng MLE hoặc MAP.

ở bước kiểm tra, nhãn của một điểm dữ liệu mới **x** được xác đinh bởi

*d*

*c =* argmaxp(c) I |p(xi|c). (5.4)

*c2{1„..,C}* **tĩ**

Khi d lớn và các xác suất nhỏ, biểu thức ở vế phải của [(5.4)](#bookmark633) là một số rất nhỏ, khi tính toán có thể gặp sai số. Đe giải quyết việc này, [(5.4)](#bookmark633) thường được viết lại dưới dạng tương đương bằng cách lấy log của vế phải:

*(*

*d \*

log(p(c)) + log(p(xi|c)n . (5.5)

i=i /

Việc này không ảnh hưởng tới kết quả vì log là một hàm đồng biến trên tập các số dương.

Sự đơn giản của NBC mang lại hiệu quả đặc biệt trong các bài toán phân loại van bản, ví dụ bài toán lọc tin nhắn hoặc email rác. Trong phần sau của bài này, chúng ta cùng xây dựng một bộ lọc email rác tiếng Anh đơn giản. Cả quá trình huấn luyện và kiểm tra của NBC đều cực kỳ nhanh so với các phương pháp phân loại phức tạp khác. Việc giả sử các thành phần trong dữ liệu là độc lập với nhau khiến cho việc tính toán mỗi phân phối p(**x**i|c) trở nên đơn giản.

Việc tính toán p(**x**i|c) phụ thuộc vào loại dữ liệu. Có ba loại phân bố xác suất phổ biến là *Gaussian naive Bayes, multinomial naive Bayes,* và *Bernoulli Naive.* Chúng ta cùng xem xét từng loại.

1. Các phân phối thường dùng trong NBC
2. Gaussian naive Bayes

Mô hình này được sử dụng chủ yếu trong loại dữ liệu mà các thành phần là các biến liên tục. Với mỗi chiều dữ liệu *i* và một nhãn c, *xi* tuân theo một phân phối chuẩn có kỳ vọng và phương sai *Ơ2P*

( I X ( I 2 \ 1 *[ (xi dci)*

(5.6)

*p(xijc) = plx^ci^i*) = ~~. 2~~ exp ~~2~~

v2^2 V *2c*

Trong đó, bộ tham số *0* = *Ơ2i}* được xác đinh bằng MLE dựa trên các điểm

trong tập huấn luyện thuộc nhãn c.

1. Multinomial naive Bayes

Mô hình này chủ yếu được sử dụng trong bài toán phân loại van bản mà vector đặc trưng được xây dựng dựa trên ý tưởng bag of words (BoW). Lúc này, mỗi van bản được biểu diễn bởi một vector có độ dài *d* là số từ trong từ điển. Giá tri của thành phần thứ *i* trong mỗi vector là số lần từ thứ *i* xuất hiện trong van bản đó. Khi đó, p(xi|c) tỉ lệ với tần suất từ thứ *i* (hay đặc trưng thứ *i* trong trường hợp tổng quát) xuất hiện trong các van bản có nhãn c. Giá tri này có thể được tính bởi

Aci = p(xi|c) = *N*

(5.7)

Trong đó:

*• Nci* là tổng số lần từ thứ *i* xuất hiện trong các van bản của nhãn c. Nó chính là tổng tất cả thành phần thứ *i* của các vector đặc trưng ứng với nhãn c.

*• Nc* là tổng số từ, kể cả lặp, xuất hiện trong nhãn c. Nói cách khác, *Nc* là tổng độ dài của tất cả các van bản thuộc nhãn c. Có thể suy ra rằng Nc V*di=* 1 *Nci,*

từ đó d=1 *Xci* = 1.

Cách tính này có một hạn chế là nếu có một từ mới chưa bao giờ xuất hiện trong nhãn *c* thì biểu thức [(5.7)](#bookmark567) sẽ bằng không, dẫn đến vế phải của [(5.4)](#bookmark633) bằng không bất kể các giá tri còn lại lớn thế nào (xem thêm ví dụ ở mục sau). Đe giải quyết việc này, một kỹ thuật được gọi là *làm mềm Laplace* (Laplace smoothing) được áp dụng:

C \_ Nci + a  
ci Nc + *da*

(5.8)

với *a* là một số dương, thường bằng 1, để tránh trường hợp tử số bằng không. Mẫu số được cộng với *da* để đảm bảo tổng xác suất 52d= 1 Aci = 1. Như vậy, mỗi nhãn c được mô tả bởi một bộ các số dương có tổng bằng 1: Ac = {Ac1,..., Acd}.

1. Bernoulli Naive Bayes

Mô hình này được áp dụng cho các loại dữ liệu mà mỗi thành phần là một giá tri nhi phân - bằng 0 hoặc 1. Ví dụ, cũng với loại van bản nhưng thay vì đếm tổng số lần xuất hiện của một từ trong van bản, ta chỉ cần quan tâm từ đó có xuất hiện hay không.

Khi đó, p(xi|c) được tính bởi

*p(xi\c) = p(i\c)xi*(1 - p(i\c))1\_xi (5.9)

với p(i\c) được hiểu là xác suất từ thứ i xuất hiện trong các van bản của class c, *Xi* bằng 1 hoặc 0 tuỳ vào việc từ thứ i có xuất hiện hay không.

1. Ví dụ
2. Bắc hay Nam

Giả sử trong tập huấn luyện có các van bản d1, d2, d3, d4 như trong Bảng [5.1](#bookmark660). Mỗi van bản này thuộc vào một trong hai nhãn: B *(Bắc*) hoặc N *(Nam*). Hãy xác đinh nhãn của van bản d5.

Bảng 5.1: Ví dụ về nội dung của các văn bản trong bài toán Bắc hay Nam

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Văn bản** | **Nội dung** | **Nhãn** |
| Dữ liệu huấn luyện | d1 | hanoi pho chaolong hanoi | B |
| d2 | hanoi buncha pho omai | B |
| d3 | pho banhgio omai | B |
| d4 | saigon hutiu banhbo pho | N |
| Dữ liệu kiểm tra | d5 | hanoi hanoi buncha hutiu | ? |

Ta có thể dự đoán rằng d5 có nhãn Bắc.

Bài toán này có thể được giải quyết bằng NBC sử dụng multinomial Naive Bayes hoặc Bernoulli naive Bayes. Chúng ta sẽ cùng làm ví dụ với mô hình thứ nhất và triển khai code cho cả hai mô hình. Việc mô hình nào tốt hơn phụ thuộc vào mỗi bài toán. Ta có thể thử cả hai để chọn ra mô hình tốt hơn.

Nhận thấy rằng ở đây có hai nhãn B và N, ta cần đi tìm p(B) và p(N) dựa trên tần số xuất hiện của mỗi nhãn trong tập huấn luyện. Ta có

3 1

p(B) = 4; p(N) = 4 • (5.10)

Tập hợp toàn bộ các từ trong tập huấn luyện là

*V =* {hanoi, pho, chaolong, buncha, omai, banhgio, saigon, hutiu, banhbog



Pha kiểm tra

Pha huấn luyện

| d1: X1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d2: X2 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| d3: X3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Tổng | 3 | 3 | 1 | 1 | 2 | 1 | 0 | 0 | 0 |
| Âb | 4/20 | 4/20 | 2/20 | 2/20 | 3/20 | 2/20 | 1/20 | 1/20 | 1/20 |

nhan = B

*d =* |V | = 9

Nb = 11 (20 = Nb + |V |)

| d4: x4 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ân | 1/13 | 2/13 | 1/13 | 1/13 | 1/13 | 1/13 | 2/13 | 2/13 | 2/13 |

nhan = N

Nn = 4

(13 = Nn + |V |)

d5: x5 = [2,0,0,1,0,0, 0,1,0]

P(B|d5) « p(B) nk1p(xí|B)

= 3 *í*Al2 « 1.5 X 10-4

= 4 20 20 20 ~ 1-° X 10

p(N|d5) « p(N) nhi *p(xi* |N)

= 1 ( 1 ì2 12 ~ 1 75 X 10-5

4 113 J 13 13 ~ 175 X 10

p(x5 |B) > p(x5 |N) d5 € nhãn(B)

**Hình 5.1.** M inh hoạ NBC vói Multinomial naive Bayes cho bà i toán *Bắc hay Nam.*

Tổng cộng số phần tử trong từ điển là |V I = 9.

Hình 5.1 minh hoạ quá trình huấn luyện và kiểm tra cho bài toán này khi sử dụng Multinomial naive Bayes, trong đó làm mềm Laplace được sử dụng với *a* = 1. Chú ý, hai giá tri tìm được 1.5 X 1Q\_4 và 1.75 X 1Q\_5 không phải là hai xác suất cần tìm mà là hai đại lượng tỉ lệ thuận với hai xác suất đó. Đe tính cụ thể, ta có thể làm như sau:

1.5 X 1Q-4

p(B|d5) = ~~1~~~~.~~~~5 X 1Q~~~~/“~~ ~~+~~ ~~7~~~~75 X 1Q~~~~\_5~~ « Q.8955, p(N|d5) = 1 -p(B|d5) « Q.1Q45.

Như vậy xác suất để d5 có nhãn B là 89.55%, có nhãn N là 10.45%. Bạn đọc có thể tự tính với ví dụ khác: d6 = pho hutiu banhbo. Nếu tính toán đúng, ta sẽ thu được

p(B|d6) « Q.29, p(N|d6) « Q.71,

và suy ra d6 thuộc vào class N.

1. Bộ phân loại naive Bayes với thư viện scikit-learn

Để kiểm tra lại các phép tính phía trên, chúng ta cùng giải quyết bài toán này bằng scikit-learn. ở đây, dữ liệu huấn luyện và kiểm tra đã được đưa về dạng vector đặc trưng sử dụng túi từ.

**from** future **import** print\_function

**from** sklearn.naive\_bayes **import** MultinomialNB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **import** numpy as | np |  |  |  |  |  |
| *# train data* |  |  |  |  |  |  |
| d1 = [2, 1, 1, | 0, | 0, | 0, | 0, | 0, | 0] |
| d2 = [1, 1, 0, | 1, | 1, | 0, | 0, | 0, | 0] |
| d3 = [0, 1, 0, | 0, | 1, | 1, | 0, | 0, | 0] |
| d4 = [0, 1, 0, | 0, | 0, | 0, | 1, | 1, | 1] |

train\_data = np.array([d1, d2, d3, d4])

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| label = np.array([ | 'B', 'B', | 'B' | , | 'N' | ]) |  |
| *# test data* |  |  |  |  |  |  |
| d5 = np.array([[2, | 0, 0, 1, | 0, | 0, | 0, | 1, | 0]]) |
| d6 = np.array([[0, | 1, 0, 0, | 0, | 0, | 0, | 1, | 1]]) |

*## call MultinomialNB* model = MultinomialNB()

*# training* model.fit(train\_data, label)

*# test*

**print**('Predicting class of d5:', **str**(model.predict(d5)[0]))

**print**('Probability of d6 in each class:', model.predict\_proba(d6))

Kết quả:

Predicting class of d5: B

Probability of d6 in each class: [[ 0.29175335 0.70824665]]

Kết quả này nhất quán với những kết quả được tính bằng tay ở trên.

Nếu sử dụng Bernoulli naive Bayes, chúng ta cần thay đối một chút về vector đặc trưng. Lúc này, các giá tri khác không sẽ đều được đưa về một vì ta chỉ quan tâm đến việc từ đó có xuất hiện trong van bản hay không.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| d1 = | [1, 1, 1, 0, 0, 0, | 0, | 0, | 0] |  |  |
| d2 = | [1, 1, 0, 1, 1, 0, | 0, | 0, | 0] |  |  |
| d3 = | [0, 1, 0, 0, 1, 1, | 0, | 0, | 0] |  |  |
| d4 = | [0, 1, 0, 0, 0, 0, | 1, | 1, | 1] |  |  |
| train | \_data = np.array([d1, | | d2, | d3, | d4]) |  |
| label | = np.array(['B', ' | B', | 'B | *r r*  , | N' ]) | *# 0 -* |
| *# tes* | *t data* |  |  |  |  |  |
| d5 = | np.array([[1, 0, 0, | 1, | 0, | 0, | 0, 1, | 0]]) |
| d6 = | np.array([[0, 1, 0, | 0, | 0, | 0, | 0, 1, | 1]]) |

*data*

*B, 1 - N*

**from** future **import** print\_function

**from** sklearn.naive\_bayes **import** BernoulliNB **import** numpy as np

*# train*

*## call MultinomialNB*

model = BernoulliNB()

* *training*

model.fit(train\_data,

label)

* *test*

**print** ('Predicting class of d5:', **str**(model.predict(d5)[0]))

**print**('Probability of d6 in each class:', model.predict\_proba(d6))

Kết quả:

Predicting class of d5: B

Probability of d6 in each class: [[ 0.16948581 0.83051419]]

Ta thấy rằng, với bài toán nhỏ này, cả hai mô hình đều cho kết quả giống nhau (xác suất tìm được khác nhau nhưng không ảnh hưởng tới quyết đinh cuối cùng).

1. Bộ phân loại naive Bayes cho bài toán lọc email rác

Tiếp theo, chúng ta cùng làm việc với một bộ cơ sở dữ liệu lớn hơn. Trong ví dụ này, dữ liệu đã được xử lý, và là một tập con của cơ sở dữ liệu *Ling-Spam dataset* [(https://goo.gl/whHCd9)](https://goo.gl/whHCd9).

Tập dữ liệu này bao gồm 960 email tiếng Anh, được tách thành tập huấn luyện và tập kiểm tra theo tỉ lệ 700:260 với 50% trong mỗi tập là các email rác.

Dữ liệu ở đây đã được tiền xử lý. Các quy tắc xử lý như sau, sử dụng thư viện xử lý ngôn ngữ tự nhiên NLTK, [(http://www.nltk.org/)](http://www.nltk.org/):

* *Loại bỏ stop words*: Những từ xuất hiện thường xuyên như ‘and’, ‘the’, ‘of’,... được loại bỏ vì chúng xuất hiện ở cả hai nhãn.
* *Lemmatization*: Những từ có cùng gốc được đưa về cùng loại. Ví dụ, ‘include’, ‘includes’, ‘included’ đều được đưa chung về ‘include’. Tất cả các từ cũng đã được đưa về dạng ký tự thường.
* *Loại bỏ non-words*: chữ số, dấu câu và các ký tự đặc biệt đã được loại bỏ.

Dưới đây là một ví dụ của một email bình thường trước khi được xử lý:

Subject: Re: 5.1344 Native speaker intuitions

The discussion on native speaker intuitions has been extremely interesting, but I worry that my brief intervention may have muddied the waters. I take it that there are a number of separable issues. The first is the extent to which a native speaker is likely to judge a lexical string as grammatical or ungrammatical per se. The second is concerned with the relationships between syntax and interpretation (although even here the distinction may not be entirely clear cut).

và sau khi được xử lý:

re native speaker intuition discussion native speaker intuition extremely interest worry brief intervention muddy waters number separable issue first extent native speaker likely judge lexical string grammatical ungrammatical per se second concern relationship between syntax interpretation although even here distinction entirely clear cut

Dưới đây là một ví dụ về email rác sau khi được xử lý:

financial freedom follow financial freedom work ethic extraordinary desire earn least per month work home special skills experience required train personal support need ensure success legitimate homebased income opportunity put back control finance life ve try opportunity past fail live promise

Các từ ‘financial’, ‘extraordinary’, ‘earn’, ‘opportunity’,... là những từ thường thấy trong các email rác.

Trong ví dụ này, chúng ta sẽ sử dụng multinomial naive Bayes.

Để bài toán được đơn giản, chúng ta sẽ sử dụng dữ liệu đã được xử lý, có thể tải về tại [https://goo.gl/CSMxHU.](https://goo.gl/CSMxHU) Thư mục sau khi giải nén bao gồm các file: test-features.txt test-labels.txt

train-features-50.txt

train-features-100.txt

train-features-400.txt

train-features.txt

train-labels-50.txt

train-labels-100.txt

train-labels-400.txt

train-labels.txt

tương ứng với các file chứa dữ liệu của tập huấn luyện và tập kiểm tra. File train -features-50.txt chứa dữ liệu của tập huấn luyện thu gọn với chỉ tổng cộng 50 email. Mỗi file labels.txt chứa nhiều dòng, mỗi dòng là một ký tự 0 hoặc 1 thể hiện email là thường hay rác.

Mỗi file features.txt chứa nhiều dòng, mỗi dòng có ba số, chẳng hạn:

1 564 1

1 19 2

Trong đó, số đầu tiên là chỉ số của email, bắt đầu từ 1; số thứ hai là thứ tự của từ trong từ điển (tổng cộng 2500 từ); số thứ ba là tần xuất của từ đó trong email đang xét. Dòng đầu tiên nói rằng trong email thứ nhất, từ thứ 564 trong từ điển xuất hiện một lần. Cách lưu dữ liệu này giúp tiết kiệm bộ nhớ vì các email chỉ chứa một lượng nhỏ các từ trong từ điển.

Nếu biểu diễn mỗi email bằng một vector hàng có độ dài bằng độ dài từ điển (2500) thì dòng thứ nhất nói rằng đặc trưng thứ 564 của vector này bằng 1. Tương tự, đặc trưng thứ 19 của vector này bằng 2. Nếu không xuất hiện, các thành phần khác được hiểu bằng 0. Dựa trên các thông tin này, ta có thể tiến hành lập trình với thư viện sklearn.

Khai báo thư viện và đường dẫn tới files:

**from** future **import** print\_function

**import** numpy as np

**from** scipy.sparse **import** coo\_matrix *# for sparse matrix* **from** sklearn.naive\_bayes **import** MultinomialNB, BernoulliNB **from** sklearn.metrics **import** accuracy\_score *# for evaluating results # data path and file name* path = 'ex6DataPrepared/' train\_data\_fn = 'train-features.txt' test\_data\_fn = 'test-features.txt' train\_label\_fn = 'train-labels.txt' test\_label\_fn = 'test-labels.txt'

Tiếp theo ta cần viết hàm số đọc dữ liệu từ file data\_fn với nhãn tương ứng được lưu trong file label\_fn. Chú ý rằng số lượng từ trong từ điển là 2500.

Dữ liệu được lưu trong một ma trận mà mỗi hàng là một vector đặc trưng của email. Đây là một ma trận thưa nên ta sử dụng hàm [scipy.sparse.coo\_matrix](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html).

nwords = 2500

**def** read\_data(data\_fn, label\_fn):

*## read label\_fn*

with **open**(path + label\_fn) as f:

content = f.readlines()

label = [**int**(x.strip()) **for** x **in** content]

*## read data\_fn*

with **open**(path + data\_fn) as f:

content = f.readlines()

*# remove '\n' at the end of each line* content = [x.strip() **for** x **in** content] dat = np.zeros((**len**(content), 3), dtype = **int**) **for** i, line **in enumerate**(content):

a = line.split(' ')

dat[i, :] = np.array([**int**(a[0]), **int**(a[1]), **int**(a[2])])

*# remember to -1 at coordinate since we're in Python*

data = coo\_matrix((dat[:, 2], (dat[:, 0] - 1, dat[:, 1] - 1)),\ shape=(**len**(label), nwords))

**return** (data, label)

Đoạn code dưới đây giúp lấy dữ liệu huấn luyện và kiểm tra, sau đó sử dụng MultinomialNB để phân loại.

(train\_data, train\_label) = read\_data(train\_data\_fn, train\_label\_fn)

(test\_data, test\_label) = read\_data(test\_data\_fn, test\_label\_fn)

clf = MultinomialNB()

clf.fit(train\_data, train\_label)

y pred = clf.predict(test\_data)

**print**('Training size = %d, accuracy = %.2f%%' % \

(train\_data.shape[0],accuracy\_score(test\_label, y\_pred)\*100))

Kết quả:

Training size = 700, accuracy = 98.08%

Như vậy, có tới 98.08% email được phân loại đúng. Chúng ta tiếp tục thử với các tập huấn luyện nhỏ hơn:

train\_data\_fn = 'train-features-100.txt'

train\_label\_fn = 'train-labels-100.txt'

test\_data\_fn = 'test-features.txt'

test\_label\_fn = 'test-labels.txt'

(train\_data, train\_label) = read\_data(train\_data\_fn, train\_label\_fn)

(test\_data, test\_label) = read\_data(test\_data\_fn, test\_label\_fn)

clf = MultinomialNB()

clf.fit(train\_data, train\_label)

y pred = clf.predict(test\_data)

**print**('Training size = %d, accuracy = %.2f%%' % \

(train\_data.shape[0],accuracy\_score(test\_label, y\_pred)\*100))

train\_data\_fn = 'train-features-50.txt'

train\_label\_fn = 'train-labels-50.txt'

test\_data\_fn = 'test-features.txt'

test\_label\_fn = 'test-labels.txt'

(train\_data, train\_label) = read\_data(train\_data\_fn, train\_label\_fn)

(test\_data, test\_label) = read\_data(test\_data\_fn, test\_label\_fn) clf = MultinomialNB()

clf.fit(train\_data, train\_label)

y pred = clf.predict(test\_data)

**print**('Training size = %d, accuracy = %.2f%%' % \

(train\_data.shape[0],accuracy\_score(test\_label, y\_pred)[[1]](#footnote-2)100))

Kết quả:

Training size = 100, accuracy = 97.69%

Training size = 50, accuracy = 97.31%

Ta thấy rằng thậm chí khi tập huấn luyện là rất nhỏ, chỉ có tổng cộng 50 email, kết quả đạt được đã rất ấn tượng.

Tiếp tục thử mô hình BernoulliNB:

clf = BernoulliNB(binarize = .5) clf.fit(train\_data, train\_label) y pred = clf.predict(test\_data) **print**('Training size = %d, accuracy = %.2f%%' % \

(train\_data.shape[0],accuracy\_score(test\_label, y\_pred)\*100))

Kết quả:

Training size = 50, accuracy = 69.62%

Ta thấy rằng trong bài toán này, MultinomialNB hoạt động hiệu quả hơn.

1. Thảo luận

* Bộ phân loại Naive Bayes (NBC) thường được sử dụng trong các bài toán phân loại van bản.
* NBC có thời gian huấn luyện và kiểm tra rất nhanh. Điều này có được là do giả sử về tính độc lập giữa các thành phần.
* Nếu giả sử về tính độc lập được thoả mãn (dựa vào bản chất của dữ liệu), NBC được cho là sẽ có kết quả tốt hơn so với SVM và hồi quy logistic (bài [8)](#bookmark843) khi có ít dữ liệu huấn luyện.
* NBC có thể hoạt động với các vector đặc trưng mà một phần là liên tục (sử dụng Gaussian naive Bayes), phần còn lại ở dạng rời rạc (sử dụng multinomial hoặc Bernoulli). Sự độc lập giữa các đặc trưng khiến NBC có khả nang này.

Hạ Gradient

1. Giới thiệu

Xét một hàm số *f* : Rd ! R với tập xác đinh D,

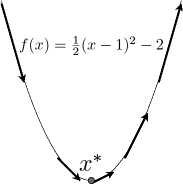
* Điểm **x**\* 2D được gọi là *cực tiểu toàn cục* (tương ứng *cực đại toàn cục*) nếu *f* (**x**) *> f* (**x**\*) (tương ứng *f* (**x**) < *f* (**x**\*)) với mọi **x** trong tập xác đinh D. Các điểm cực tiểu toàn cục và cực đại toàn cục được gọi chung là *cực trị toàn cục.*
* Điểm **x**\* 2 D được gọi là *cực tiểu địa phương* (tương ứng *cực đại địa phương*) nếu tồn tại " > 0 sao cho *f* (**x**) > *f* (**x**\*) (tương ứng *f* (**x**) < *f* (**x**\*)) với mọi **x** nằm trong lân cận V(") = {**x** : **x** 2 D,d(**x**, **x**\*) < "}. ở đây d(**x**, **x**\*) ký hiệu khoảng cách giữa hai vector **x** và **x**\*, thường là khoảng cách Euclid. Các điểm cực tiểu đia phương và cực đại đia phương được gọi chung là *cực trị địa phương.* Các điểm cực tiểu/cực đại/cực tri toàn cục cũng là các điểm cực tiểu/cực đại/cực tri đia phương.

Giả sử ta đang quan tâm đến một hàm liên tục một biến có đạo hàm mọi nơi, xác đinh trên R. Cùng nhắc lại một vài điểm cơ bản:

* Điểm cực tiểu đia phương *x\** của hàm số là điểm có đạo hàm f°(x\*) bằng không. Hơn nữa, trong lân cận của nó, đạo hàm của các điểm phía bên trái *x\** là không dương, đạo hàm của các điểm phía bên phải *x\** là không âm.
* Đường tiếp tuyến với đồ thi hàm số đó tại một điểm bất kỳ có hệ số góc bằng đạo hàm của hàm số tại điểm đó.

Hình 6.1 mô tả sự biến thiên của hàm số f (x) = 2(x — 1)2 — 2. Điểm x\* = 1 là một điểm cực tiểu toàn cục của hàm số này. Các điểm bên trái của **x**\* có đạo

| *X* | — X | 1 | + X |
| --- | --- | --- | --- |
| *f* "0) | - | 0 | + |
| f ■■■ | + X | -2 | + X |

**Hình 6.1.** Khảo sát sự biến thiên của một đa thức bậc hai.

hàm âm, các điểm bên phải có đạo hàm dương. Với hàm số này, càng xa về phía trái của **x**\* thì đạo hàm càng âm, càng xa về phía phải thì đạo hàm càng dương.

Trong học máy nói riêng và toán tối ưu nói chung, chúng ta thường xuyên phải tìm các cực tiểu toàn cục của một hàm số. Nếu chỉ xét riêng các hàm khả vi, việc giải phương trình đạo hàm bằng không có the phức tạp hoặc có vô số nghiệm. Thay vào đó, người ta thường tìm các điểm cực tiểu địa phương, và coi đó là một nghiệm cần tìm của bài toán trong những trường hợp nhất định.

Các điểm cực tiểu địa phương là nghiệm của phương trình đạo hàm bằng không (ta vẫn đang giả sử rằng các hàm này liên tục và khả vi). Nếu tìm được toàn bộ (hữu hạn) các điểm cực tiểu địa phương, ta chỉ cần thay từng điểm đó vào hàm số để suy ra điểm cực tiểu toàn cục. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng không là bất khả thi. Nguyên nhân có the đến từ sự phức tạp của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn hoặc từ việc có quá nhiều điểm dữ liệu. Thực tế cho thấy, trong nhiều bài toán machine learning, các điểm cực tiểu địa phương thường cho kết quả tốt, đặc biệt là trong các mạng neuron nhân tạo.

Một hướng tiếp cận phổ biến để giải quyết các bài toán tối ưu là dùng một phép toán. Đầu tiên, chọn một *điểm xuất phát* rồi tiến dần đến *đích* sau mỗi vòng lặp. Hạ Gradient - Gradient descent (GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

***Chú ý***: Khái niệm *nghiệm* của một bài toán tối ưu được sử dụng không hẳn để chỉ cực tiểu toàn cục. Nó được sử dụng theo nghĩa là kết quả của quá trình tối ưu. Kết quả ở một vòng lặp trung gian được gọi là *vị trí của nghiệm.* Nói cách khác, *nghiệm* có thể được hiểu là giá trị hiện tại của tham số cần tìm trong quá trình tối ưu.

1. Hạ gradient cho h àm một biến

Xét các hàm số một biến *f* : R ! R. Quay trở lại Hình 6.1 và một vài quan sát đã nêu. Giả sử *xt* là điểm tìm được sau vòng lặp thứ *t.* Ta cần tìm một thuật toán để đưa *xt* về càng gần *x\** càng tốt. Có hai quan sát sau đây:

* Nếu đạo hàm của hàm số tại *xt* là dương (f0(xt) > 0) thì *xt* nằm về bên phải so với x\*, và ngược lại. Để điểm tiếp theo *xt+1* gần với x\* hơn, ta cần di chuyển *xt* về bên trái, tức về phía *âm.* Nói các khác, *ta cần di chuyển xt ngược dấu với đạo hàm*:

*xt+1 = xt* + *A.* (6.1)

Trong đó *A* là một đại lượng ngược dấu với đạo hàm *f 0(xt).*

* *xt* càng xa x\* về bên phải thì *f 0(xt)* càng lớn (và ngược lại). Một cách tự nhiên nhất, ta chọn lượng di chuyển *A* tỉ lệ thuận với *—f0(xt*).

Từ hai nhận xét trên, ta có công thức cập nhật đơn giản là

(6.2)

Trong đó *g* là một số dương được gọi là *tốc độ học (learning rate*). Dấu trừ thể hiện việc xt cần *đi ngược* với đạo hàm *f 0(xt).* Tên gọi *Hạ gradient* xuất phát từ đây. Mặc dù các quan sát này không đúng trong mọi trường hợp, chúng vẫn là nền tảng cho rất nhiều phương pháp tối ưu.

1. Ví dụ đơn giản với Python

Xét hàm số *f* (x) = x2 + 5 sin(x) với đạo hàm f(x) = 2x + 5 cos(x). Giả sử xuất phát từ một điểm xo, quy tắc cập nhật tại vòng lặp thứ *t* là

xt+1 = xt - *g(2xt* + 5 cos(xt)).

(6.3)

Khi thực hiện trên Python, ta cần viết các hàm số

1. grad để tính đạo hàm.
2. cost để tính giá tri của hàm số. Ta không sử dụng hàm này trong thuật toán cập nhật nghiệm. Tuy nhiên, nó vẫn đóng vai trò quan trọng trong việc kiểm tra tính chính xác của đạo hàm và sự biến thiên của hàm số sau mỗi vòng lặp.
3. myGDl là phần chính thực hiện thuật toán GD. Đầu vào của hàm số này là điểm xuất phát x® và tốc độ học eta. Đầu ra là nghiệm của bài toán. Thuật toán dừng lại khi đạo hàm đủ nhỏ.

**def** grad(x):

**return** 2\*x+ 5\*np.cos(x)

**def** cost(x):

**return** x\*\*2 + 5\*np.sin(x)

**def** myGD1(x0, eta):

x = [x0]

**for** it **in range**(100):

x\_new = x[-1] - eta\*grad(x[-1])

**if abs**(grad(x\_new)) < 1e-3: *# just a small number* **break**

x.append(x\_new)

**return** (x, it)

***Điểm xuất phát khác nhau***

Sau khi đã có các hàm cần thiết, chúng ta thử tìm nghiệm với các điểm xuất phát khác nhau là *x0 =* —5 và *x0* = 5 với cùng tốc độ học = 0.1.

(x1, it1) = myGD1(-5, .1)

(x2, it2) = myGD1(5, .1)

**print**('Solution x1 = %f, cost = %f, after %d iterations'\ %(x1[-1], cost(x1[-1]), it1))

**print**('Solution x2 = %f, cost = %f, after %d iterations'\ %(x2[-1], cost(x2[-1]), it2))

Kết quả:

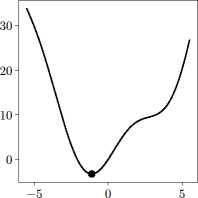
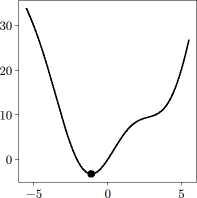
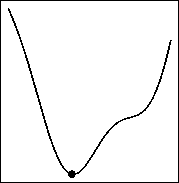
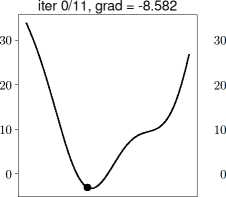
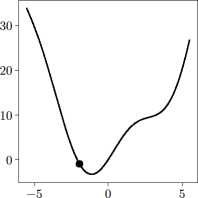
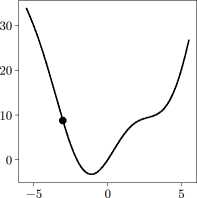
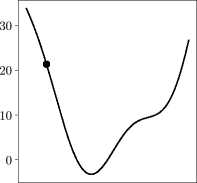
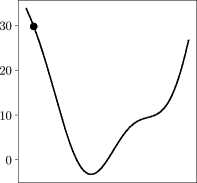
Solution x1 = -1.110667, cost = -3.246394, after 11 iterations

Solution x2 = -1.110341, cost = -3.246394, after 29 iterations

Như vậy, thuật toán trả về kết quả gần giống nhau với các điểm xuất phát khác nhau, nhưng tốc độ hội tụ khác nhau. Hình [6.2](#bookmark710) và Hình [6.3](#bookmark711) the hiện vi trí của *xt* và đạo hàm qua các vòng lặp với cùng tốc độ học = 0.1 nhưng điểm xuất phát khác nhau tại —5 và 5.

Hình [6.2](#bookmark710) tương ứng với x0 *=* —5, thuật toán hội tụ nhanh hơn. Hơn nữa, đường đi tới đích khá suôn sẻ với đạo hàm luôn âm và tri tuyệt đối của đạo hàm nhỏ dần khi xt tiến gần tới đích.

Hình [6.3](#bookmark711) tương ứng với x0 = 5, đường đi của xt chứa một khu vực có đạo hàm khá nhỏ gần điểm có hoành độ bằng 2.5. Điều này khiến thuật toán la cà ở đây khá lâu. Khi vượt qua được điểm này thì mọi việc diễn ra tốt đẹp. Các điểm không phải là điểm cực tiểu nhưng có đạo hàm gần bằng không rất dễ gây ra hiện tượng *xt* bi *bẫy* vì đạo hàm nhỏ khiến nó không thay đổi nhiều ở vòng lặp tiếp theo. Chúng ta sẽ thấy một kỹ thuật khác giúp thuật toán *thoát* những chiếc bẫy này.



-5 0 5 -5 0 5

iter2/11, grad = -11.063

iter 3/11, grad = -5.665

iter 1/11, grad = -10.984

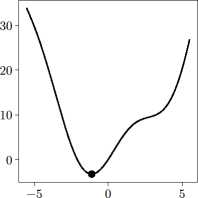
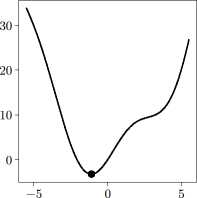
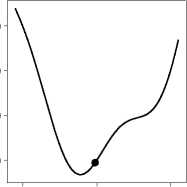
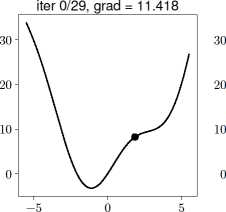
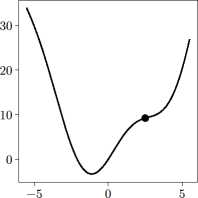
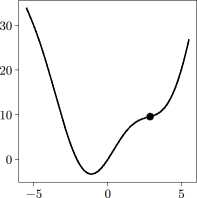
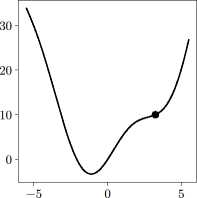
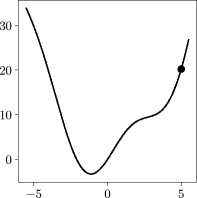
-5 0 5 -5 0 5

iter 4/11, grad = -1.747 iter 5/11, grad = -0.561

iter 7/11, grad = -0.066

iter 11/11, grad = -0.001

**Hình 6.2.** Ket quả tìm được qua các vòng lặp vói x0 = —5, = 0.1



iter 3/29, grad = 1.517

iter 6/29, grad = 0.925

iter 10/29, grad = 0.983

-5

0

5

iter 25/29, grad = 0.071

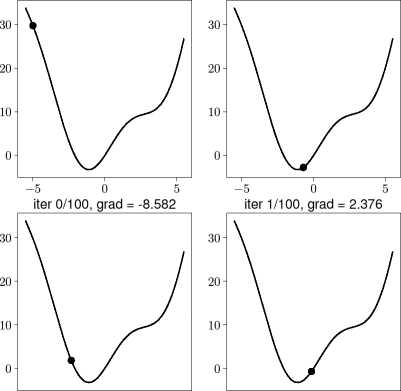
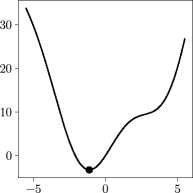
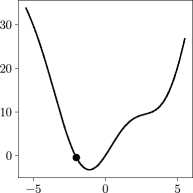
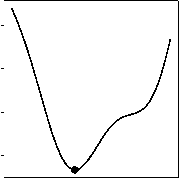
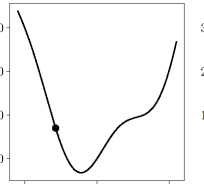
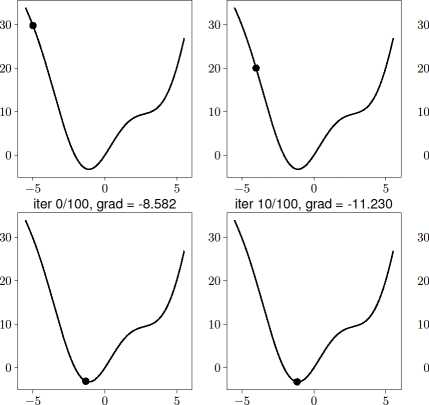
iter 29/29, grad = 0.001

iter 15/29, grad = 2.341 iter 20/29, grad = 4.739

**Hình 6.3.** Ket quả tìm được qua các vòng lặp vói x0 = 5, = 0.1

***Tốc độ học khác nhau***

Tốc độ hội tụ của GD không những phụ thuộc vào điểm xuất phát mà còn phụ thuộc vào tốc độ học. Hình 6.4 và Hình [6.5](#bookmark712) thể hiện vi trí của *xt* qua các vòng lặp với cùng điểm xuất phát x0 = —5 nhưng tốc độ học khác nhau. Ta quan sát thấy hai điều:



iter 50/100,grad =-1.476

iter 70/100, grad = -0.368

-5 0 5

iter 20/100, grad = -10.584

-5 0 5

iter 90/100, grad = -0.095

iter 30/100, grad = -6.175

iter 100/100, grad = -0.049

**Hình 6.4.** Kết quả tìm được qua các vòng lặp vói xo =

-5 0 5

iter 50/100,grad =-8.114

-5 0 5

iter 70/100, grad = 4.663

| 30- |  | 30- |  |
| --- | --- | --- | --- |
| 20- | \ / | 20- | \ / |
| 10- | *\ /* | 10- | \ G |
| 0- | V | 0- |  |
|  | -5 0 5  iter 2/100,grad = -5.398 |  | -5 0 5  iter 3/100,grad = 5.081 |
| 30- | \ / | 30- | \ / |
| 20- | \ / | 20- | \ / |
| 10- | *\ /* | 10- | \ G |
| 0- | V | 0- | V |

—5, = 0.01.

-5 0 5

iter 90/100, grad = -7.038

-5 0 5

iter 100/100, grad = 4.761

**Hình 6.5.** Kết quả tìm được qua các vòng lặp vói *x0 =* —5, = 0.5

* Với tốc độ học nhỏ *q =* 0.01 (Hình 6.4), tốc độ hội tụ rất chậm. Trong ví dụ này ta chọn tối đa 100 vòng lặp nên thuật toán dừng lại trước khi tới đích, mặc dù đã rất gần. Trong thực tế, khi việc tính toán trở nên phức tạp, tốc độ học quá thấp sẽ ảnh hưởng nhiều tới tốc độ của thuật toán. Thậm chí xt có thể không bao giờ tới được đích.
* Với tốc độ học lớn *q =* 0.5 (Hình [6.5)](#bookmark712), xt tiến nhanh tới gần đích sau vài vòng lặp. Tuy nhiên, thuật toán không hội tụ được vì sự thay đổi vi trí của *xt* sau mỗi vòng lặp là quá lớn, khiến *xt* dao động quanh đích nhưng không tới được đích.

Việc lựa chọn tốc độ học rất quan trọng. Tốc độ học thường được chọn thông qua các thí nghiệm. Ngoài ra, GD có thể làm việc hiệu quả hơn bằng cách chọn tốc độ học khác nhau ở mỗi vòng lặp. Trên thực tế, một kỹ thuật thường được sử dụng có tên là *suy giảm tốc độ học (learning rate decay*). Trong kỹ thuật này, tốc độ học được giảm đi sau một vài vòng lặp để nghiệm không bi dao động mạnh khi gần đích hơn.

1. Hạ gradient cho h àm nhiều biến

Giả sử ta cần tìm cực tiểu toàn cục cho hàm *f* (ớ) trong đó *ớ* là tập hợp các tham số cần tối ưu. Gradient của hàm số đó tại một điểm ớ bất kỳ được ký hiệu là r*ỹf* (ớ). Tương tự như hàm một biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán *ớ0*, sau đó sử dụng quy tắc cập nhật

ớt+1 = ớt - *gVof* (ớt)

(6.4)

Hoặc viết dưới dạng đơn giản hơn: ớ — ớ — *gVỹf (ớ).*

***Quay lại với bài toán hồi quy tuyến tính***

Trong mục này, chúng ta quay lại với bài toán hồi quy tuyến tính và thử tối ưu hàm mất mát của nó bằng thuật toán GD.

Nhắc lại hàm mất mát của hồi quy tuyến tính và gradient theo **w**:

L(w) = *2N*ky \_ XTwk2; r**w**L(w) = NX(X^**w - y**) (6.5)

***Ví dụ trên Python và một vài lưu ý khi lập trình***

Trước tiên, chúng ta tạo 1000 điểm dữ liệu gần đường thẳng *y* = 4 + 3x rồi dùng thư viện scikit-learn đe tìm nghiệm cho hồi quy tuyến tính:

**from** sklearn.linear\_model **import** LinearRegression

X = np.random.rand(1000)

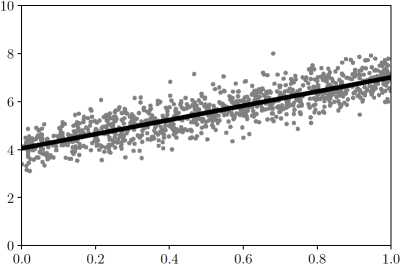
y=4+3\*X+ .5\*np.random.randn(1000) *# noise added* model = LinearRegression()

model.fit(X.reshape(-1, 1), y.reshape(-1, 1))

w, b = model.coef\_[0][0], model.intercept\_[0]

sol\_sklearn = np.array([b, w])

**print**(sol\_sklearn)

**Hình 6.6.** Ngh iệm của bài toán hồi quy tuyến tính (đường thằng màu đen) tìm được bằng thư viện scikit-learn.

Kết quả:

Solution found by sklearn: [ 3.94323245 3.12067542]

Các điểm dữ liệu và đường thẳng tìm được bằng hồi quy tuyến tính có phương trình *y* 3.94 + 3.12x được minh hoạ trong Hình 6.6. Nghiệm tìm này được rất gần với mong đợi.

Tiếp theo, ta sẽ thực hiện tìm nghiệm bằng GD. Ta cần viết hàm mất mát và gradient theo **w**. Chú ý rằng ở đây **w** đã bao gồm hệ số điều chỉnh *b.*

**def** grad(w):

N = Xbar.shape[0]

**return** 1/N \* Xbar.T.dot(Xbar.dot(w) - y)

**def** cost(w):

N = Xbar.shape[0]

**return** .5/N\*np.linalg.norm(y - Xbar.dot(w))\*\*2 **def** myGD(w\_init, grad, eta):

w = [w\_init]

**for** it **in range**(100):

w\_new = w[-1] - eta\*grad(w[-1])

**if** np.linalg.norm(grad(w\_new))/**len**(w\_new) < 1e-3:

**break**

w.append(w\_new)

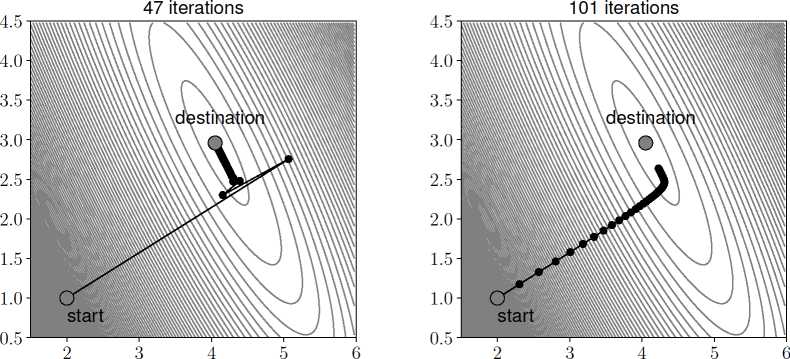
**return** w, it

one = np.ones((X.shape[0],1))

Xbar = np.concatenate((one, X.reshape(-1, 1)), axis = 1) w\_init = np.array([[2], [1]])

w1, it1 = myGD(w\_init, grad, 1)

**print**('Sol found by GD: w = ', w1[-1].T, ', after %d iterations.' %(it1+1))



**(a)** r = 1.

**(b)** r = 0.1.

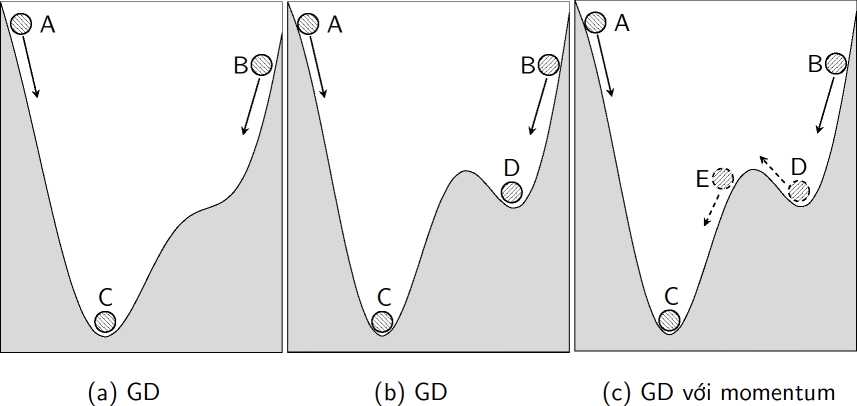
**Hình 6.7.** Dường đi nghiệm của hồi quy tuyến tính vói các tốc độ học khác nhau.

Kết quả:

Sol found by GD: w = [ 3.99026984 2.98702942] , after 49 iterations.

Thuật toán hội tụ tới kết quả khá gần với nghiệm tìm được theo scikit-learn sau 49 vòng lặp. Hình [6.7](#bookmark721) mô tả đường đi của **w** với cùng điểm xuất phát nhưng tốc độ học khác nhau. Các điểm được đánh dấu ‘start’ là các điểm xuất phát. Các điểm được đánh dấu ‘destination’ là nghiệm tìm được bằng thư viện scikit-learn. Các điểm hình tròn nhỏ màu đen là vi trí của **w** qua các vòng lặp trung gian. Ta thấy rằng khi *'q* = 1, thuật toán hội tụ tới rất gần đích theo thư viện sau 49 vòng lặp. Với tốc độ học nhỏ hìn, *q* = 0.1, nghiệm vẫn còn cách xa đích sau hìn 100 vòng lặp. Như vậy, việc chọn tốc độ học hợp lý là rất quan trọng.

ở đây, chúng ta cùng làm quen với một khái niệm quan trọng: *đường đồng mức.* Khái niệm này thường xuất hiện trong các bản đồ tự nhiên. Với các ngọn núi, đường đồng mức là các đường kín bao quanh đỉnh núi, bao gồm các điểm có cùng độ cao so với mực nước biển. Khái niệm tưìng tự cũng được sử dụng trong tối ưu. Đường đồng mức của một hàm số là tập hợp các điểm làm cho hàm số có cùng giá tri. Xét một hàm số hai biến với đồ thi là một bề mặt trong không gian ba chiều. Các đường đồng mức là giao điểm của bề mặt này với các mặt phẳng song song với đáy. Hàm mất mát của hồi quy tuyến tính với dữ liệu một chiều là một hàm bậc hai theo hai thành phần trong vector trọng số **w**. Đồ thi của nó là một bề mặt parabolic. Vì vậy, các đường đồng mức của hàm này là các đường ellipse có cùng tâm như trên Hình [6.7.](#bookmark721) Tâm này chính là đáy của parabolic và là giá tri nhỏ nhất của hàm mất mát. Các đường đồng mức càng gần tâm (’destination’) tưìng ứng với giá tri càng thấp.



**Hình 6.8.** So sánh GD vói các hiện tượng vật lý.

1. Hạ gradient với momentum

Trước hết, nhắc lại thuật toán GD để tối ưu một hàm mất mát *J(0):*

* Dự đoán một điểm xuất phát *0* = 00.
* Cập nhật 0 theo công thức

0 < 0 - ^Vớ *J*(0) (6.6)

tới khi hội tụ. ở đây, Vớ *J*(0) là gradient của hàm mất mát tại 0.

***Gradient dưới góc nhìn vật ly***

Thuật toán GD thường được ví với tác dụng của trọng lực lên một hòn bi đặt trên một mặt có dạng thung lũng như Hình 6.8a. Bất ke ta đặt hòn bi ở A hay B thì cuối cùng nó cũng sẽ lan xuống và kết thúc ở vi trí C.

Tuy nhiên, nếu bề mặt có hai đáy thung lũng như Hình 6.8b thì tùy vào việc đặt bi ở A hoặc B, vi trí cuối cùng tương ứng của bi sẽ ở C hoặc D (giả sử rằng ma sát đủ lớn và đà không mạnh để bi có thể vượt dốc). Điểm D là một điểm cực tiểu đia phương, điểm C là điểm cực tiểu toàn cục.

Vẫn trong Hình 6.8b, nếu vận tốc ban đầu của bi ở điểm B đủ lớn, nó vẫn có thể tiến tới dốc bên trái của D do có *đà.* Nếu vận tốc ban đầu lớn hơn nữa, bi có thể vượt dốc tới điểm E rồi lan xuống C như trong Hình 6.8c. Dựa trên quan sát này, một thuật toán được ra đời nhằm giúp GD thoát được các cực tiểu đia phương. Thuật toán đó có tên là *momentum* (tức *theo đà*).

***Hạ gradient với momentum***

Làm thế nào để biểu diễn *momentum* dưới dạng toán học?

Trong GD, ta cần tính lượng thay đổi ở thời điểm *t* để cập nhật vị trí mới cho nghiệm (tức *hòn bi*). Nếu ta coi đại lượng này như vận tốc *vt* trong vật lý, vị trí mới của hòn bi sẽ là ớt+1 = *ớt — vt* với giả sử rằng mỗi vòng lặp là một đìn vị thời gian. Dấu trừ thể hiện việc phải di chuyển ngược với gradient. Việc tiếp theo là tính đại lượng *vt* sao cho nó vừa mang thông tin của *độ dốc* hiện tại (tức gradient), vừa mang thông tin của đà. Thông tin của đà có thể được hiểu là vận tốc trước đó *Vt-1* (với giả sử rằng vận tốc ban đầu v0 = 0). Một cách đìn giản nhất, ta có thể lấy tổng trọng số của chúng:

(6.7)

*Vt = 7Vt-1* + *I/Vỹ J (ớ)*

Trong đó 7 là một số dưìng nhỏ hìn một. Giá trị thường được chọn là khoảng 0.9, *Vt-1* là vận tốc tại thời điểm trước đó, *Vỹ J*(ớ) chính là độ dốc tại điểm hiện tại. Từ đó, ta có công thức cập nhật nghiệm:

(6.8)

ớ — ớ - *Vt =* ớ - *'qVỹ J*(ớ) - 7Vt 1

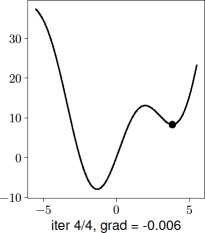
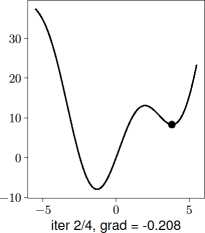
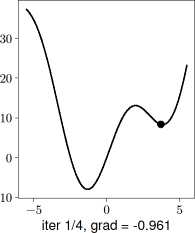
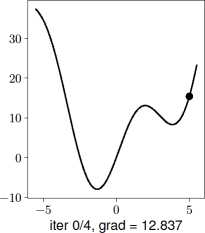
Sự khác nhau giữa GD thông thường và GD với momentem nằm ở thành phần cuối cùng trong [(6.8)](#bookmark713). Thuật toán đìn giản này mang lại hiệu quả trong các bài toán thực tế.

Xét một hàm đìn giản có hai điểm cực tiểu địa phưìng, trong đó một điểm là cực tiểu toàn cục:

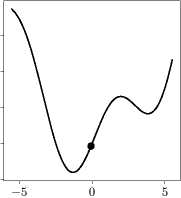
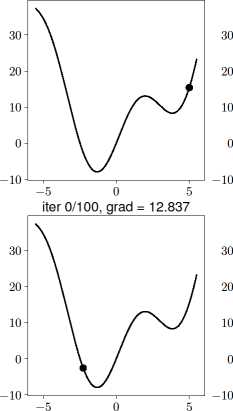
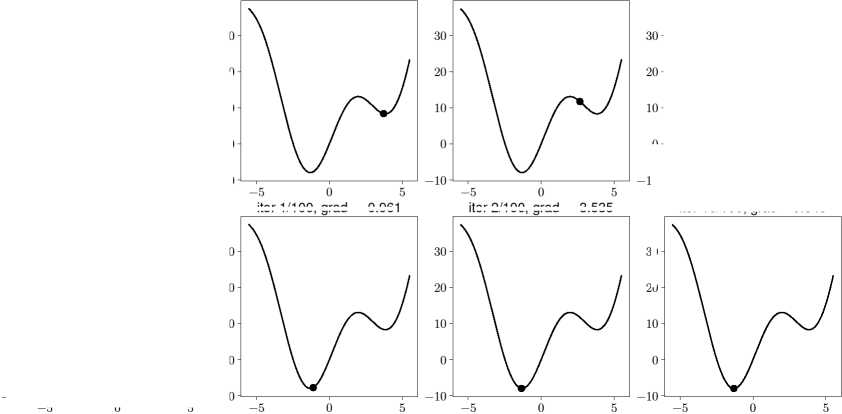
*f (x)* = *x2* + 10sin(x). (6.9)

Hàm số này có đạo hàm là *f* 0(x) = 2x + 10cos(x). Hình 6.9 thể hiện các vị trí trung gian của nghiệm khi không sử dụng momentum. Ta thấy rằng thuật toán hội tụ nhanh chóng sau chỉ bốn vòng lặp. Tuy nhiên, nghiệm đạt được không phải là cực tiểu toàn cục. Trong khi đó, Hình [6.10](#bookmark731) thể hiện các vị trí trung gian của nghiệm khi có sử dụng momentum. Chúng ta thấy rằng hòn bi vượt được dốc thứ nhất nhờ có đà, theo quán tính tiếp tục vượt qua điểm cực tiểu toàn cục, nhưng trở lại điểm này sau 50 vòng lặp rồi chuyển động chậm dần quanh đó tới khi dừng hẳn ở vòng lặp thứ 100. Ví dụ này cho thấy momentum thực sự đã giúp nghiệm thoát được khu vực cực tiểu địa phưìng.

Nếu biết trước điểm xuất phát theta, gradient của hàm mất mát tại một điểm bất kỳ grad(theta), lượng thông tin lưu trữ từ vận tốc trước đó gamma và tốc độ học eta, chúng ta có the viết hàm GD\_momentum như sau:



**Hình 6.9.** GD thông thường



20

0

iter1/100,grad =-0.961

iter 2/100, grad = -3.535

iter10/100, grad = 9.845

20

iter 50/100,grad = 2.289

iter75/100, grad = -0.462

iter 100/100, grad = -0.044

iter 20/100, grad = -10.917

10

30

**Hình 6.10.** GD vói momentum

**def** GD\_momentum(grad, theta\_init, eta, gamma):

*# Suppose we want to store history of theta*

theta = [theta\_init]

v\_old = np.zeros\_like(theta\_init)

**for** it **in range**(100):

v\_new = gamma\*v\_old + eta\*grad(theta[-1]) theta\_new = theta[-1] - v\_new

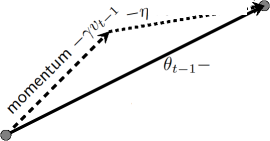
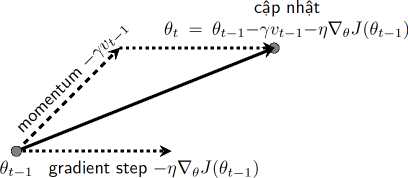
**if** np.linalg.norm(grad(theta\_new))/np.array(theta\_init).size < 1e-3:

**break**

theta.append(theta\_new)

v\_old = v\_new

**return** theta

cập nhật *ot =*

(a) Momentum gradient descent.

Toạ độ điểm tính gradient thay đổi

*ỡt-1*

<7

(b) Nesterov accelerated gradient.

*7Vt-i-n^e J*(A-1 - ?VÉ\_1)

**Hình 6.11.** Ý tưởng của Nesterov accelerated gradient

1. Nesterov accelerated gradient

Momentum giúp nghiệm vượt qua được khu vực cực tiểu địa phương. Tuy nhiên, có một hạn chế có the thấy trong ví dụ trên. Khi tới gần đích, momemtum khiến nghiệm dao động một khoảng thời gian nữa trước khi hội tụ. Một kỹ thuật có tên *Nesterov accelerated gradient* giúp cho thuật toán momentum GD hội tụ nhanh hơn.

Y tưởng trung tâm của thuật toán là dự đoán vị trí của nghiệm trước một bước. Cụ thể, nếu sử dụng số hạng momentum 7Ví\_i để cập nhật thì vị trí tiếp theo của nghiệm là *0 —* 7Vt\_1. Vậy, thay vì sử dụng gradient tại điểm hiện tại, NAG sử dụng gradient tại điểm tiếp theo *nếu sử dụng momentum.* Y tưởng này được the hiện trên Hình [6.11](#bookmark732).

1. Công thức cập nhật

Công thức cập nhật của NAG được cho như sau:

*Vt = yut-i* + *gVỹ J(ỡ -* 7Vt\_i)

*0-0 — vt*

Đoạn code dưới đây thể hiện cách cập nhật nghiệm bằng NAG:

(6.10)

(6.11)

**def** GD\_NAG(grad, theta\_init, eta, gamma):

theta = [theta\_init]

v = [np.zeros\_like(theta\_init)]

**for** it **in range**(100):

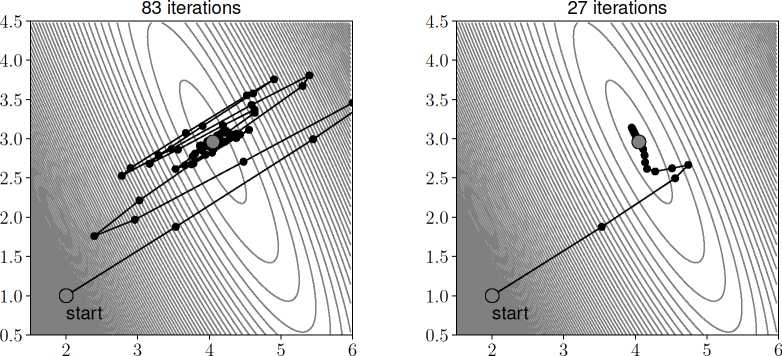
v\_new = gamma\*v[-1] + eta\*grad(theta[-1] - gamma\*v[-1]) theta\_new = theta[-1] - v\_new

**if** np.linalg.norm(grad(theta\_new))/np.array(theta\_init).size < 1e-3: **break**

theta.append(theta\_new)

v.append(v\_new)

**return** theta



**(a)** GD vói momentum.

**(b)** GD vói NAG.

**Hình 6.12.** Dường đi của nghiệm cho bài toán hồi quy tuyến tính vói hai phưong pháp Hạ gradient khác nhau. NAG cho nghiệm mượt hon và nhanh hon.

1. Ví dụ minh họa

Chúng ta cùng áp dụng cả GD với momentum và GD với NAG cho bài toán hồi quy tuyến tính. Hình 6.12a the hiện đường đi của nghiệm với phương pháp momentum. Nghiệm đi khá *zigzag* và mất nhiều vòng lặp hơn. Hình 6.12b thể hiện đường đi của nghiệm với phương pháp NAG, nghiệm hội tụ nhanh hơn và đường đi ít *zigzag* hơn.

1. Hạ gradient ngẫu nhiên
2. Hạ gradient theo Batch

Thuật toán GD được đề cập từ đầu bài còn được gọi là *batch gradient desenct.* Batch ở đây được hiểu là *tất cả,* tức sử dụng tất cả các điểm dữ liệu ***X****ị* để cập nhật bộ tham số *0.* Hạn chế của việc này là khi lượng cơ sở dữ liệu lớn, việc tính toán gradient trên toàn bộ dữ liệu tại mỗi vòng lặp tốn nhiều thời gian.

*Online learning* là khi cơ sở dữ liệu được cập nhật liên tục, mỗi lần tang thêm vài điểm dữ liệu mới. Việc này yêu cầu mô hình cũng phải được thay đổi để phù hợp với dữ liệu mới. Nếu thực hiện batch GD, tức tính lại gradient của hàm mất mát với toàn bộ dữ liệu, độ phức tạp tính toán sẽ rất cao. Lúc đó, thuật toán có the không còn mang tính *online* nữa do mất quá nhiều thời gian tính toán.

Một kỹ thuật đơn giản hơn được sử dụng là *Hạ gradient ngẫu nhiên* (SGD). Thuật toán này có thể gây ra sai số nhưng mang lại lợi ích về mặt tính toán.

1. Hạ gradient ngẫu nhiên

Trong SGD, tại một thời điểm, ta tính gradient của hàm mất mát dựa trên *chỉ một* điểm dữ liệu ***x*** rồi cập nhật *0.* Chú ý rằng hàm mất mát thường được lấy trung bình trên tất điểm dữ liệu nên gradient tương ứng với một điểm được kỳ vọng là khá gần với gradient tính theo mọi điểm dữ liệu. Sau khi duyệt qua tất cả các điểm dữ liệu, thuật toán lặp lại quá trình trên. Biến thể đơn giản này trên thực tế làm việc rất hiệu quả.

***epoch***

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một *epoch.* Với GD thông thường, mỗi epoch ứng với một lần cập nhật 0. Với SGD, mỗi epoch ứng với *N* lần cập nhật 0 với *N* là số điểm dữ liệu. Một mặt, việc cập nhật 0 theo từng điểm có the làm giảm tốc độ thực hiện một epoch. Nhưng mặt khác, với SGD, nghiệm có thể hội tụ sau vài epoch. Vì vậy, SGD phù hợp với các bài toán có lượng cơ sở dữ liệu lớn và các bài toán yêu cầu mô hình thay đổi liên tục như *học trực tuyến.* Với một mô hình đã được huấn luyện từ trước, khi có thêm dữ liệu, ta có thể chạy thêm một vài epoch nữa là đã có nghiệm hội tụ.

*Mỗi lần cập nhật nghiệm là một vòng lặp. Mỗi lần duyệt hết toàn bộ dữ liệu, là một epoch. Một epoch bao gồm nhiều vòng lặp.*

***Thứ tự lựa chọn điểm dữ liệu***

Một điểm cần lưu ý là sau mỗi epoch, thứ tự lấy các dữ liệu cần được xáo trộn để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu nang của SGD. Đây cũng chính là lý do thuật toán này có chứa từ *stochastic-ngẫu nhiên.*

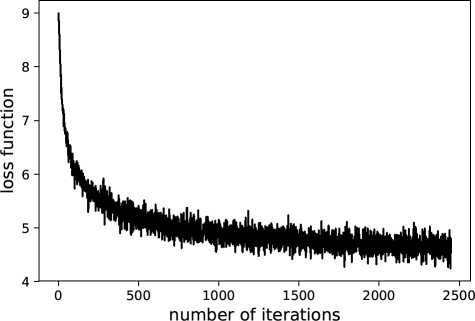
Quy tắc cập nhật của SGD là

0< 0 - *I/Mi J (0;* ***X****i,* **y**i) (6.12)

Trong đó J(0; **x**i, **y**i) , *Ji(0')* là hàm mất mát nếu chỉ có một cặp dữ liệu thứ i. Các kỹ thuật biến the của GD như momentum hay NAG hoàn toàn có the được áp dụng vào SGD.

1. Hạ gradient theo Mini-batch

Khác với SGD, mini-batch GD sử dụng 1 < *k < N* điểm dữ liệu để cập nhật ở mỗi vòng lặp. Giống với SGD, mini-batch GD bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các mini-batch, mỗi

**Hình 6.13.** Ví dụ về giá trị hàm mat mát sau mỗi vòng lặp khi sử dụng mini-batch Hạ gradient. Hàm m§t mát dao động sau mỗi lan cập nhật nhưng nhìn chung giảm dan và có xu hưóng hội tụ.

mini-batch có *k* điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu *N* không chia hết cho k). ở mỗi vòng lặp, một mini-batch được lấy ra để tính toán gradient rồi cập nhật *0.* Khi thuật toán chạy hết dữ liệu một lượt cũng là khi kết thúc một epoch. Như vậy, một epoch bao gồm xấp xỉ N/k vòng lặp. Giá tri k được gọi là *kích thước batch* (không phải *kích thước mini-batch*) được chọn trong khoảng khoảng từ vài chục đến vài trăm.

Hình 6.13 là ví dụ về giá tri của hàm mất mát của một mô hình phức tạp hơn khi sử dụng mini-batch GD. Mặc dù giá tri của hàm mất mát sau các vòng lặp không luôn luôn giảm, nhìn chung giá tri này có xu hướng giảm và hội tụ.

1. Thảo luận
2. Điều kiện dừng thuật toán

Khi nào thì nên dừng thuật toán GD?

Trong thực nghiệm, chúng ta có thể kết hợp các phương pháp sau:

1. Giới hạn số vòng lặp. Nhược điểm của cách làm này là thuật toán có thể dừng lại trước khi nghiệm đủ tốt. Tuy nhiên, đây là phương pháp phổ biến nhất và cũng đảm bảo được chương trình chạy không quá lâu.
2. So sánh gradient của hàm mất mát tại hai lần cập nhật liên tiếp, khi nào giá tri này đủ nhỏ thì dừng lại.
3. So sánh giá tri của hàm mất mát sau một vài epoch, khi nào sự sai khác đủ nhỏ thì dừng lại. Nhược điểm của phương pháp này là nếu hàm mất mát có dạng bằng phẳng tại một điểm không phải cực tiểu đia phương, thuật toán sẽ dừng lại trước khi đạt giá tri mong muốn.
4. Vừa chạy GD, vừa kiểm tra kết quả. Một kỹ thuật khác thường được sử dụng là cho thuật toán chạy với số lượng vòng lặp lớn. Trong quá trình chạy, chương trình thường xuyên kiểm tra chất lượng mô hình trên tập huấn luyện và tập xác thực. Đồng thời, mô hình sau một vài vòng lặp được lưu lại trong bộ nhớ. Nếu ta thấy chất lượng mô hình bắt đầu giảm trên tập xác thực thì dừng lại. Đây chính là kỹ thuật *early stoping* - Dừng sớm để giảm quá khớp.

***Bài* 7**

Thuật toán học perceptron

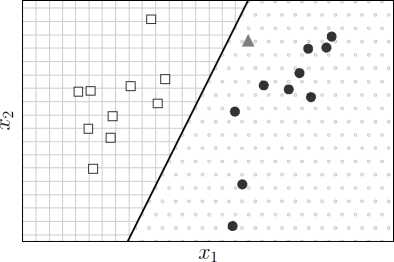
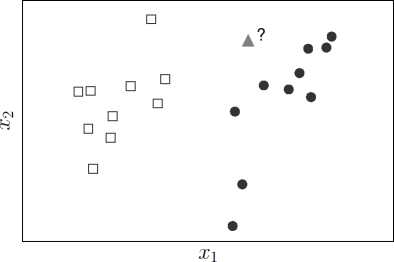
1. Giới thiệu

Trong bài này, chúng ta cùng tìm hiểu một trong các thuật toán xuất hiện đầu tiên trong lịch sử machine learning. Đây là một phưong pháp phân loại đìn giản có tên là *thuật toán học perceptron* (perceptron learning algorithm - PLA). Thuật toán này được thiết kế cho bài toán *phân loại nhị phân* khi dữ liệu chỉ thuộc một trong hai nhãn. Đây là nền tảng cho các thuật toán liên quan tới mạng neuron nhân tạo và gần đây là deep learning.

Giả sử có hai tập dữ liệu hình vuông và tròn như được minh hoạ trong Hình [7.1a](#bookmark791). Bài toán đặt ra là từ dữ liệu của hai tập được gán nhãn cho trước, hãy xây dựng một bộ phân loại có khả nang dự đoán nhãn của một điểm dữ liệu mới, chẳng hạn điểm hình tam giác màu xám.

Nếu coi mỗi vector đặc trưng là một điểm trong không gian nhiều chiều, bài toán phân loại có thể được coi như bài toán xác định nhãn của từng điểm trong không gian. Nếu coi mỗi nhãn chiếm một hoặc vài vùng trong không gian, ta cần đi tìm ranh giới giữa các vùng đó. Ranh giới đìn giản nhất trong không gian hai chiều là một đường thẳng, trong không gian ba chiều là một mặt phẳng, trong không gian nhiều chiều là một *siêu phẳng.* Những ranh giới phẳng này đìn giản vì chúng có the được biểu diễn bởi một hàm số tuyến tính. Hình [7.1b](#bookmark791) minh họa một đường thẳng phân chia hai tập dữ liệu trong không gian hai chiều. Trong trường hợp này, điểm dữ liệu mới hình tam giác rìi vào cùng tập hợp với các điểm hình tròn.

PLA là một thuật toán đìn giản giúp tìm ranh giới siêu phẳng cho bài toán phân loại nhị phân trong trường hợp tồn tại siêu phẳng đó. Nếu hai tập dữ liệu có thể được phân chia hoàn toàn bằng một siêu phẳng, ta nói rằng hai tập đó *tách biệt tuyến tính* (linearly separable).



**(a)**

**(b)**

**Hình 7.1.** Bài toán phân loại nhị phân trong không gian hai chiều. (a) Cho hai tập dữ liệu được gán nhãn vuông và tròn, hãy xác định nhãn của điếm tam giác. (b) Ví dụ về một ranh giói phang phân chia hai tập hợp. Diem tam giác được phân vào tập các điêm hình tròn.

1. Thuật toán học perceptron
2. Quy tắc phân loại

Giả sử **X** = [**x**1, **x**2,..., **x**N] 2 RdxN là ma trận chứa tập huấn luyện mà mỗi cột ***x*** là một điểm dữ liệu trong không gian *d* chiều. Các nhãn được lưu trong một vector hàng **y** = *[y1,y2,... , yN*] 2 R1xN với *yi* = 1 nếu ***x*** mang nhãn vuông và *yi =* —1 nếu **x**i mang nhãn tròn.

Tại một thời điểm, giả sử ranh giới là một siêu phẳng có phưìng trình

*f****w*** (**x**) = *W1X1* + + *wdxd* + *w0 =* ***x****T****w*** + *w0* = 0 (7.1)

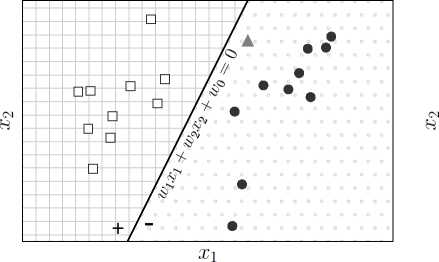
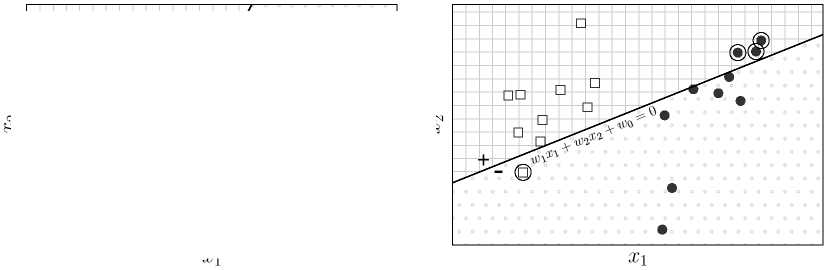
với **w** 2 là vector trọng số và w0 là hệ số điều chỉnh. Bằng cách sử dụng thủ thuật gộp hệ số điều chỉnh (xem Mục [3.2.4)](#bookmark512), ta có thể coi phưìng trình siêu phẳng là fw(**x**) = **x**T**w** = 0 với **x** ở đây được ngầm hiểu như vector đặc trưng mở rộng thêm một đặc trưng bằng một. Vector trọng số **w** chính là *vector pháp tuyến* của siêu phẳng **x**T**w** = 0.

Trong không gian hai chiều, giả sử đường thẳng W1 x1 + *w2x2* + w0 = 0 là nghiệm cần tìm như Hình 7.2a. Ta thấy rằng các điểm nằm cùng phía so với đường thẳng này làm cho hàm số *fw(****x****)* mang cùng dấu. Nếu cần thiết, ta có thể đổi dấu của **w** để các điểm trên nửa mặt phẳng nền kẻ ô vuông mang dấu dưìng (+), các điểm trên nửa mặt phẳng nền chấm mang dấu âm (-). Các dấu này tưìng đưìng với nhãn y của mỗi điểm dữ liệu. Như vậy, nếu **w** là một nghiệm của bài toán thì nhãn của một điểm dữ liệu mới **x** được xác đinh bởi

1 nếu **x**T**w >** 0

label(**x**) = (7.2)

— 1 o.w.



**(a)** Đường thẳng phân chia không gây lỗi, mọi điểm **(b)** Đường thẳng phân chia gây ra lỗi tại các điểm được phân loại đúng. được khoanh tròn.

**Hình 7.2.** Ví dụ về các đường thẳng trong không gian hai chiều: (a) một nghiệm của bài toán PLA, (b) đường thẳng không phân chia chính xác hai lóp.

Vậy, label(**x**) = sgn(**w**T**x**) với sgn là hàm xác đinh dấu. Quy ước sgn(0) = 1.

1. Xây dựng h àm mất mát

Tiếp theo, chúng ta xây dựng một hàm mất mát theo tham số **w** bất kỳ. Vẫn trong không gian hai chiều, xét đường thẳng *W1X1* + *w2x2* + *w0* = 0 được cho như Hình 7.2b. Các điểm khoanh tròn là các điểm bi *phân loại lỗi.* Tham số **w** là một nghiệm của bài toán nếu nó không gây ra điểm bi phân loại lỗi nào. Như vậy, hàm đếm số lượng điểm bi phân loại lỗi có thể coi là hàm mất mát của mô hình. Ta sẽ tìm cách tối thiểu hàm số này.

Nếu một điểm ***X****ị* với nhãn *yi* bi phân loại lỗi, ta có sgn(**x**T**w**) = *yị.* Vì hai giá tri này chỉ bằng 1 hoặc —1, ta phải có yisgn(**x**T**w**) = —1. Như vậy, hàm đếm số lượng điểm bi phân loại lỗi có thể được viết dưới dạng

Ji(w) = 52 *(-yisgn(xTw))*

(7.3)

***X****i2M*

trong đó M ký hiệu tập các điểm bi phân loại lỗi ứng với mỗi **w**. Mục đích cuối cùng là đi tìm **w** sao cho mọi điểm trong tập huấn luyện đều được phân loại đúng, tức *J1(****w****) =* 0. Một điểm quan trọng cần lưu ý là hàm mất mát *J1(****w****)* rất khó được tối ưu vì sgn là một hàm rời rạc. Chúng ta cần tìm một hàm mất mát khác để việc tối ưu khả thi hìn. Xét hàm

J(w) = 52 *(~yixi* w)-

(7.4)

**X**i2M

Trong hàm số này, hàm rời rạc sgn đã được lược bỏ. Ngoài ra, khi một điểm phân loại lỗi **x**i nằm càng xa ranh giới, giá tri —yi**x**T**w** sẽ càng lớn, khiến cho hàm mất mát cũng càng lớn. Lưu ý rằng hàm mất mát chỉ được tính trên các tập điểm biphân loại lỗi M, giá tri nhỏ nhất của hàm số này cũng bằng không nếu M là một tập rỗng. Vì vậy, *J*(**w**) được cho là tốt hìn *J1* (**w**) vì nó *trừng phạt rất nặng những điểm lấn sâu sang lãnh tho của tập còn lại.* Trong khi đó, J1(**w**) *trừng phạt* các điểm phân loại lỗi một lượng như nhau và đều bằng một, bất kể chúng gần hay xa ranh giới.

1. Tối ưu hàm mất mát

Tại một thời điểm, nếu chỉ quan tâm tới các điểm bi phân loại lỗi thì hàm số *J*(**w**) khả vi tại mọi **w**. Vậy ta có thể sử dụng GD hoặc SGD để tối ưu hàm mất mát này. Chúng ta sẽ giải quyết bài toán tối ưu hàm mất mát J(**w**) bằng SGD. Nếu chỉ một điểm dữ liệu **X** bi phân loại lỗi, hàm mất mát và gradient của nó

lần lượt là J(**w**; **X**i; *yi)* = *-yi****X****T***w**; V**w** J(**w**; **X**i; *yi) = -yi****X****ị* (7.5)

Quy tắc cập nhật **w** sử dụng SGD là

**w w -** *y(-yi****X****i) =* **w** + *yyi****X****i* (7.6)

với *y* là tốc độ học. Trong PLA, *y* được chọn bằng 1. Ta có một quy tắc cập nhật rất gọn:

**w**t+i = ***w****t* + yi**X**i (7.7)

Tiếp theo, ta thấy rằng

**X**T **w**t+1 = x' *(****w****t* + *yi****X****i) =* ***xỊ' w****t* + ydl xdl 2. (7.8)

Nếu **X**i bi phân loại lỗi và có nhãn đúng yi = 1, ta có **X**T***w****t <* 0. Cũng vì yi = 1 nên

yi^**x**ik2 = 11x42 — 1 (chú ý **X**i là một vector đặc trưng mở rộng với một phần tử bằng một). Từ đó suy ra **X**T**w**t+1 > **X**T**w**t. Nói cách khác, — yi**x**T**w**t+1 < —yi**x**T**w**t.

ii i i

Điều tưìng tự cũng xảy ra với yi = —1. Việc này chỉ ra rằng đường thẳng được mô tả bởi **w**t+1 có xu hướng khiến hàm mất mát tại điểm bi phân loại lỗi **X**i giảm đi. *Chú ý rằng việc này không đảm bảo hàm mất mát tong cộng sẽ giảm, vì rất có thể siêu thẳng mới sẽ làm cho một điểm lúc trước được phân loại đúng trở thành một điểm bị phân loại lỗi. Tuy nhiên, thuật toán này được đảm bảo sẽ hội tụ sau một số hữu hạn bước.* Thuật toán perceptron được tóm tắt dưới đây:

Thuật toán 7.1: Perceptron

1. *Tại thời điểm t* = 0, *chọn ngẫu nhiên một vector trọng số* ***w****0.*
2. *Tại thời điểm t, nếu không có điểm dữ liệu nào bị phân loại lỗi, dừng thuật toán.*
3. *Giả sử* ***X****i là một điểm bị phân loại lỗi, cập nhật*

**w**í+1 = **w**t + *yi****X****i*

1. *Thay đoi t* = t + 1 *rồi quay lại Bước 2.*
2. Chứng minh hội tụ

Gọi **w**\* là một nghiệm của bài toán phân loại nhi phân. Nghiệm này luôn tồn tại khi hai tập dữ liệu tách biệt tuyến tính. Ta sẽ chứng minh bằng phản chứng Thuật toán [7.1](#bookmark808) kết thúc sau một số hữu hạn bước.

Giả sử ngược lại, tồn tại một điểm xuất phát **w** khiến Thuật toán [7.1](#bookmark808) chạy vô hạn bước. Trước hết ta thấy rằng, nếu **w**\* là nghiệm thì a**w**\* cũng là nghiệm của bài toán với *a >* 0 bất kỳ. Xét dãy số không âm ua(t) = ||wt — aw\* 112. Theo giả thiết phản chứng, luôn tồn tại một điểm bi phân loại lỗi khi dùng nghiệm **w**t. Giả sử đó là điểm ***x*** với nhãn *yp* Ta có

*ua(t* + 1) = I|wt+1 - aw\* 112

= ||wt + *yi****X****i -* aw\*||2

= **w**t — a**w** 2 + y,2 **xj**2 + *2yi****X****T* (**w**t — a**w** )

2 i 2 i

< *ua*(t) + **llx**ill2 - 2my**x**./**w**\* (7.9)

Dấu nhỏ hìn ở dòng cuối xảy ra vì y2 = 1 và 2yi**x**T***w****t <* 0. Nếu tiếp tục đặt

ộ2 = max llxill2 *>* 1, 7 = min yi**X**T**w**\*

*i=1,2,...,N* 2 *i=1,2,...,N*

và chọn *a* = Y, ta sẽ có 0 *< ua(t* +1) < *ua(t)* + *ộ*2 — 2a7 = *ua(t) — ộ2.* Ta có thể chọn giá tri này vì [(7.9)](#bookmark815) đúng với *a* bất kỳ. Điều này chỉ ra rằng nếu luôn có điểm bi phân loại lỗi thì dãy *ua* (t) là một dãy giảm bi chặn dưới bởi 0, và phần tử sau kém phần tử trước ít nhất một lượng là ộ2 > 1. Điều vô lý này chứng tỏ giả thiết phản chứng là sai. Nói cách khác, thuật toán perceptron hội tụ sau một số hữu hạn bước.

1. Ví dụ và minh hoạ trên Python

Thuật toán [7.1](#bookmark808) có thể được triển khai như sau:

***Quy tắc phân loại***

Giả sử đã tìm được vector trọng số w, nhãn của các điểm dữ liệu X được xác đinh bằng hàm predict(w, X):

**import** numpy as np

**def** predict(w, X):

*ri ff ff*

*predict label of each row of X, given w*

*X: a 2-d numpy array of shape (N, d), each row is a datapoint*

*w: a 1-d numpy array of shape (d)*

*fl fl fl*

**return** np.sign(X.dot(w))

***Thuật toán tối ưu hàm mất mát***

Hàm perceptron(X, y, w\_init) thực hiện thuật toán PLA với tập huấn luyện X, nhãn y và nghiệm ban đầu w\_init.

**def** perceptron(X, y, w\_init):

*""" perform perceptron learning algorithm*

*X: a 2-d numpy array of shape (N, d), each row is a datapoint y: a 1-d numpy array of shape (N), label of each row of X. y[i] = 1/-1 w\_init: a 1-d numpy array of shape (d)*

*fl fl fl*

w = w\_init

**while** True:

pred = predict(w, X)

* *find indexes of misclassified points* mis\_idxs = np.where(np.equal(pred, y) == False)[0]
* *number of misclassified points* num\_mis = mis\_idxs.shape[0]

**if** num\_mis == 0: *# no more misclassified points*

**return** w

* *randomly pick one misclassified point* random\_id = np.random.choice(mis\_idxs, 1)[0]
* *update w*

w = w + y[random\_id]\*X[random\_id]

**return** w

Ắp dụng thuật toán vừa viết vào dữ liệu trong không gian hai chiều:

means = [[-1, 0], [1, 0]]

cov = [[.3, .2], [.2, .3]]

N = 10

X0 = np.random.multivariate\_normal(means[0], cov, N)

X1 = np.random.multivariate\_normal(means[1], cov, N)

X = np.concatenate((X0, X1), axis = 0)

y = np.concatenate((np.ones(N), -1\*np.ones(N)))

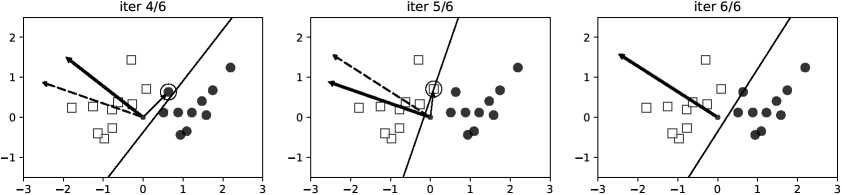
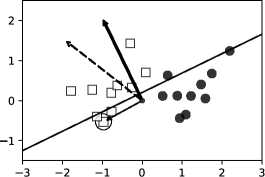
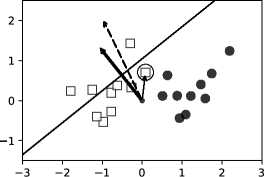
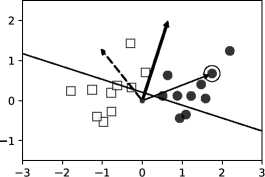
Xbar = np.concatenate((np.ones((2\*N, 1)), X), axis = 1) w\_init = np.random.randn(Xbar.shape[1])

w = perceptron(Xbar, y, w\_init)

Mỗi nhãn có 10 phần tử, là các vector ngẫu nhiên lấy theo phân phối chuẩn có ma trận hiệp phưong sai cov và vector kỳ vọng means. Hình 7.3 minh hoạ thuật toán học perceptron cho bài toán này. Nghiệm hội tụ chỉ sau sáu vòng lặp.

1. Mô hình mạng neuron đầu tiên

Hàm số dự đoán đầu ra của perceptron label(**x**) = sgn(**w**T**x**) được mô tả trên Hình [7.4a.](#bookmark835) Đây chính là dạng đìn giản nhất của một mạng neuron.

**Hình 7.3.** Minh hoạ thuật toán perceptron. Các điểm hình vuông có nhãn bằng 1, các điếm hình tròn có nhãn —1. Tại mỗi vòng lặp, đường thang là đường ranh giói. Vector pháp tuyến **w**t của đường thằng này là vector đậm nét lien. Diem được khoanh tròn là một điếm bị phân loại lỗi ***X****ị.* Vector mảnh nét lien the hiện vector ***X****ị.* Vector nét đứt the hiện **w**t+1. Nếu *yi* = 1 (một điem hình vuông), vector nét đứt bằng tong hai vector kia. Nếu *yi =* —1, vector nét đứt bằng hiệu hai vector kia.

**iter 1/6**

**iter 2/6**

**iter 3/6**

***y*** *=* sgn(**z**)

*d*

T ***W****ị****X****ị =* **w x**

***x****o*

***X****1*

**x**2

**X**3

**X**d

***W****o*

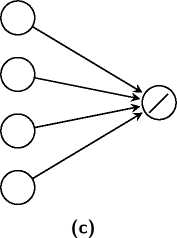
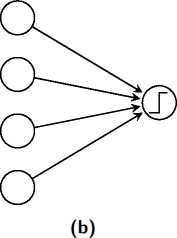
***W****1*

**W**3

*/ z*

***W****d*

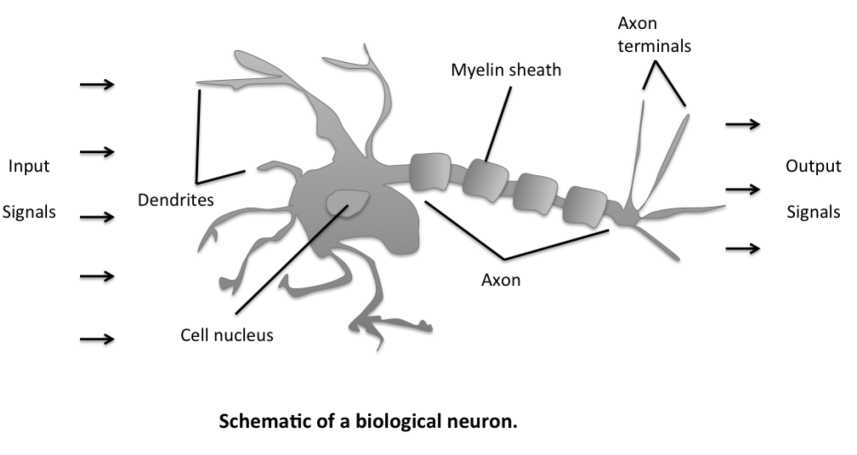
**(a)**

**Hình 7.4.** Biểu diễn perceptron và hồi quy tuyến tính dưói dạng mạng neuron. (a) perceptron đay đủ, (b) perceptron thu gọn, (c) hồi quy tuyến tính thu gọn.

Tầng đầu vào Tầng đầu ra

Tầng đầu vào Tầng đầu ra

Đầu vào **x** của mạng được minh họa bằng các hình tròn bên trái gọi là các *nút.* Tập hợp các nút này được gọi là *tầng đầu vào.* Số nút trong tầng đầu vào là *d* + 1 với nút điều chỉnh *x0* đôi khi được ẩn đi và ngầm hiểu bằng một. Các *trọng số w0,w1,... ,Wd* được gán vào các mũi tên đi tới nút *z d'oi=*0*wixi = xT***w**. Nút *y =* sgn(z) là đầu ra của mạng. Ký hiệu hình chữ Z ngược trong nút *y* thể hiện đồ thi của hàm sgn. Hàm y = sgn(z) đóng vai trò là một *hàm kích hoạt.* Có nhiều loại hàm kích hoạt khác nhau sẽ được trình bày trong các bài sau. Dữ liệu đầu



**Hình 7.5.** Cấu trúc của một neuron than kinh sinh học.

vào được đặt tại tầng đầu vào, lấy tổng có trọng số lưu vào biến *z* rồi đi qua hàm kích hoạt để có kết quả *ở y.* Đây chính là dạng đìn giản nhất của một mạng neuron nhân tạo. Perceptron cũng có the được vẽ giản lược như Hình [7.4b](#bookmark835), với ẩn ý rằng hàm tính tổng và hàm kích hoạt được gộp làm một.

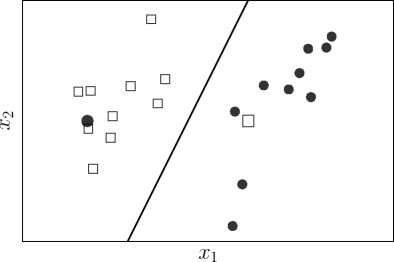
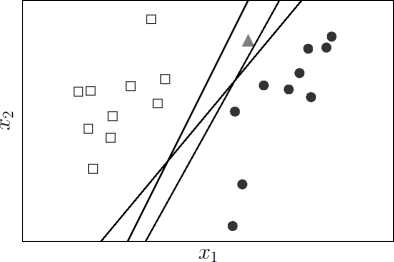
Các mạng neuron có thể có một hoặc nhiều nút ở đầu ra tạo thành một *tầng đầu ra.* Trong các mô hình phức tạp hìn, các mạng neuron có the có thêm các tầng trung gian giữa tầng đầu vào và tầng đầu ra gọi là *tầng ẩn.* Chúng ta sẽ đi sâu vào các mạng nhiều tầng ẩn ở học phần học sâu. Trước đó, chúng ta sẽ tìm hiểu các mạng neuron đìn giản hìn không có tầng ẩn nào.

Đe ý rằng nếu thay hàm kích hoạt bởi hàm đồng nhất y = z, ta sẽ có một mạng neuron mô tả mô hình hồi quy tuyến tính như Hình [7.4c](#bookmark835). Đường thẳng chéo trong nút đầu ra thể hiện đồ thi hàm số y = z. Các trục tọa độ đã được lược bỏ.

Mô hình perceptron ở trên khá giống với một thành phần nhỏ của mạng thần kinh sinh học như Hình 7.5. Dữ liệu từ nhiều dây thần kinh đầu vào đi về một nhân tế bào. Nhân tế bào tổng hợp thông tin và đưa ra quyết đinh ở tín hiệu đầu ra. Trong mạng neuron nhận tạo của perceptron, mỗi giá tri *Xi* đóng vai trò một tín hiệu đầu vào, hàm tính tổng và hàm kích hoạt có chức nang tưìng tự nhân tế bào. Tên gọi *mạng neuron nhãn tạo* được khởi nguồn từ đây.

1. Thảo Luận

*PLA có thể cho vô số nghiệm khác nhau.* Nếu hai tập dữ liệu tách biệt tuyến tính thì có vô số đường ranh giới như trong Hình 7.6a. Các đường khác nhau sẽ quyết



**(a)**

**(b)**

**Hình 7.6.** Vói bà i toán phân loại nhị phân, PLA có the (a) cho vô số nghiệm, hoặc (b) vô nghiệm thậm chí khi có nhiễu nhỏ.

đinh điểm hình tam giác có nhãn khác nhau. Trong các đường đó, đường nào là tốt nhất? Và đinh nghĩa “tốt nhất” được hiểu theo nghĩa nào? Các câu hỏi này sẽ được thảo luận kỹ hìn trong bài 10.

*PLA đòi hỏi hai tập dữ liệu phải tách biệt tuyến tính.* Hình 7.6b mô tả hai tập dữ liệu *gần* tách biệt tuyến tính. Mỗi tập có một điểm nhiễu nằm lẫn tập còn lại. Trong trường hợp này, thuật toán PLA không bao giờ dừng lại vì luôn có ít nhất hai điểm bi phân loại lỗi.

Trong một chừng mực nào đó, đường thẳng màu đen vẫn có thể coi là một nghiệm tốt vì nó đã giúp phân loại chính xác hầu hết các điểm. Việc không hội tụ với dữ liệu *gần* tách biệt tuyến tính là một nhược điểm lớn của PLA.

Nhược điểm này có thể được khắc phục bằng *thuật toán bỏ túi (pocket algorithm*). Một cách trực quan, nếu chỉ có ít nhiễu, ta sẽ đi tìm một đường ranh giới sao cho có ít điểm bi phân loại lỗi nhất. Việc này có thể được thực hiện thông qua PLA và thuật toán tìm số nhỏ nhất trong mảng một chiều:

* Giới hạn số lượng vòng lặp của PLA. Đặt nghiệm **w** sau vòng lặp đầu tiên và số điểm bi phân loại lỗi vào trong túi.
* Mỗi lần cập nhật nghiệm ***w****t* mới, ta đếm xem có bao nhiêu điểm bi phân loại lỗi. So sánh số lượng này với số điểm bi phân loại lỗi trong túi. Nếu số lượng điểm bi phân loại lỗi này nhỏ hìn, tức ta đạt được mô hình tốt hìn trên tập huấn luyện, ta thay thế nghiệm trong túi bằng nghiệm mới và số điểm bi phân loại lỗi tưìng ứng. Lặp lại bước này đến khi hết số vòng lặp.

Hồi quy logistic

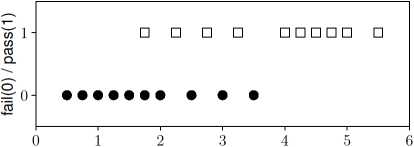
1. Giới thiệu
2. Nhắc lại hai mô hình tuyến tính

Hai mô hình tuyến tính đã học là hồi quy tuyến tính và PLA, đều có thể viết chung dưới dạng *y* = *f* (**x**T**w**) trong đó *f* (s) là một hàm kích hoạt. Trong hồi quy tuyến tính *f* (s) = *s,* tích vô hướng **x**T**w** được trực tiếp sử dụng để dự đoán đầu ra *y.* Mô hình này phù hợp nếu ta cần dự đoán một đầu ra không bị chặn. PLA có đầu ra chỉ nhận một trong hai giá trị 1 hoặc —1 với hàm kích hoạt *f* (s) = sgn(s) phù hợp với các bài toán phân loại nhị phân. Trong bài này, chúng ta sẽ thảo luận một mô hình tuyến tính với một hàm kích hoạt khác, thường được áp dụng cho các bài toán phân loại nhị phân. Trong mô hình này, đầu ra có thể được biểu diễn dưới dạng xác suất. Ví dụ, xác suất thi đỗ nếu biết thời gian ôn thi, xác suất ngày mai có mưa dựa trên những thông tin đo được trong ngày hôm nay,... Mô hình này có tên là *hồi quy logistic.* Mặc dù trong tên có chứa từ *hồi quy,* phương pháp này thường được sử dụng nhiều hơn cho các bài toán phân loại.

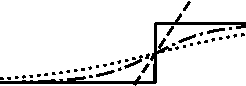
1. Một ví dụ nhỏ

Bảng 8.1: Thời gian ôn thi và kết quả thi của 20 sinh viên.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Số giờ** | **Đâu?** | **Số giờ** | **Đâu?** | **Số giờ** | **Đâu?** | **Số giờ** | **Đâu?** |
| 0.5 | 0 | 0.75 | 0 | 1 | 0 | 1.25 | 0 |
| 1.5 | 0 | 1.75 | 0 | 1.75 | 1 | 2 | 0 |
| 2.25 | 1 | 2.5 | 0 | 2.75 | 1 | 4 | 0 |
| 3.25 | 1 | 3.5 | 0 | 4 | 1 | 4.25 | 1 |
| 4.5 | 1 | 4.75 | 1 | 5 | 1 | 5.5 | 1 |

**Hình 8.1.** Ví dụ về kết quả thi dựa trên số giờ ôn tập. Trục hoành the hiện thời gian ôn tập của mỗi sinh viên, trục tung gồm hai giá trị 0/fail (các diêm hình tròn) và 1/pass (các diêm hình vuông).

hours studying

**Hình 8.2.** Một vài ví dụ về các hàm kích hoạt khác nhau.

/

hard threshold

” f(s) = ĩ+-s

*f* (s) = —

linear

Xét một ví dụ về quan hệ giữa thời gian ôn thi và kết quả của 20 sinh viên trong Bảng [8.1](#bookmark856). Bài toán đặt ra là từ dữ liệu này hãy xây dựng mô hình đánh giá khả nang đỗ của một sinh viên dựa trên thời gian ôn tập. Dữ liệu trong Bảng [8.1](#bookmark856) được mô tả trên Hình 8.1. Nhìn chung, thời gian học càng nhiều thì khả nang đỗ càng cao. Tuy nhiên, không có một ngưỡng thời gian học nào giúp phân biệt rạch ròi việc đỗ/trượt . Nói cách khác, dữ liệu của hai tập này là không tách biệt tuyến tính, và vì vậy PLA sẽ không hữu ích. Tuy nhiên, thay vì dự đoán chính xác hai giá tri đỗ/trượt, ta có thể dự đoán xác suất để một sinh viên thi đỗ dựa trên thời gian ôn thi.

1. Mô hình hồi quy logistic

Quan sát Hình 8.2 với các hàm kích hoạt *f* (s) khác nhau.

* Đường nét đứt biểu diễn một hàm kích hoạt tuyến tính không phù hợp vì đầu ra không bi chặn. Có một cách đơn giản để đưa đầu ra về dạng bi chặn: nếu đầu ra nhỏ hơn không thì thay bằng không, nếu đầu ra lớn hơn một thì thay bằng một. Điểm phân chia, còn gọi là *ngưỡng,* được chọn là điểm có tung độ 0.5 trên đường thằng này. Đây cũng không phải là một lựa chọn tốt. Giả sử có thêm một bạn sinh viên tiêu biểu ôn tập đến 20 giờ hoặc hơn thi đỗ. Lúc này ngưỡng tương ứng với mốc tung độ bằng 0.5 sẽ dich nhiều về phía phải. Kéo theo đó, rất nhiều sinh viên thi đỗ được dự đoán là trượt. Rõ ràng đây là một mô hình không tốt. Nhắc lại rằng hồi quy tuyến tính rất nhạy cảm với nhiễu, ở đây là bạn sinh viên tiêu biểu đó.
* Đường nét liền tương tự với hàm kích hoạt của PLA. Ngưỡng dự đoán đỗ/trượt tại vi trí hàm số đổi dấu còn được gọi là *ngưỡng cứng.*
* Các đường nét chấm và chấm gạch phù hợp với bài toán đang xét hơn. Chúng có một vài tính chất quan trọng:
* Là các hàm số liên tục nhận giá tri thực, bi chặn trong khoảng (0, 1).
* Nếu coi điểm có tung độ bằng 0.5 là ngưỡng, các điểm càng xa ngưỡng về bên trái có giá tri càng gần không, các điểm càng xa ngưỡng về bên phải có giá tri càng gần một. Điều này phù hợp với nhận xét rằng học càng nhiều thì xác suất đỗ càng cao và ngược lại.
* Hai hàm này có đạo hàm mọi nơi, điều này có thể có ích trong tối ưu.

***Hàm sigmoid và tanh***

Trong các hàm số có ba tính chất nói trên, hàm *sigmoid*:

f(s) = 1+1ezs - s (8-1)

được sử dụng nhiều nhất, vì nó bi chặn trong khoảng (0,1) và:

lim *ơ(s) =* 0; lim *ơ(s) =* 1. (8.2)

Thú vi hơn:

*e~s* 1 *e~s*

*ơ0(s) =* ~~e~~ = -^— = a(s)(1 - a(s)) (8.3)

(1 + *e~s)2* 1 + *e~s* 1 + *e~s*

Với đạo hàm đơn giản, hàm sigmoid được sử dụng rộng rãi trong mạng neuron. Chúng ta sẽ sớm thấy hàm sigmoid được khám phá ra như thế nào.

Ngoài ra, hàm *tanh* cũng hay được sử dụng:

tanh(s) = ~~e~~ ~~'~~ ~~e~~ = 2a(2s) — 1. (8.4)

*es* + e *s*

Hàm số này nhận giá tri trong khoảng (—1,1).

Hàm sigmoid có thể được thực hiện trên Python như sau:

**def** sigmoid(S):

*fl fl fl*

*S: an numpy array*

*return sigmoid function of each element of S*

*fl fl fl*

**return** 1/(1 + np.exp(-S))

1. Hàm mất mát và phương pháp tối ưu
2. Xây dựng hàm mất mát

Với các mô hình có hàm kích hoạt *f* (s) 2 (0,1), ta có thể giả sử rằng xác suất để một điểm dữ liệu *Xị* có nhãn thứ nhất là *f* (**x**T**w**) và nhãn còn lại là 1 — *f* (**x**T**w**):

*p(yi* = 1|**x**ì; **w**) = f í**x**'**w**) (8.5)

*p(yi =* 0|xí; w) = 1 - f í **x**'w) (8.6)

trong đó *p(yi =* 1 |**x**i; **w**) được hiểu là xác suất xảy ra sự kiện nhãn yi = 1 khi biết tham số mô hình **w** và dữ liệu đầu vào **x**i. Mục đích là tìm các hệ số **w** sao cho *f* (**x**t**w**) *yi* với mọi điểm trong tập huấn luyện.

Ký hiệu *ai = f* (**x**t**w**), hai biểu thức [(8.5)](#bookmark880) và [(8.6)](#bookmark880) có thể được viết gọn lại:

p(yi|**x**i; **w**) = ayi(1 - *ai*)1\_yi (8.7)

Biểu thức này tương đương với hai biểu thức [(8.5)](#bookmark880) và [(8.6)](#bookmark880) vì khi yi = 1, thừa số thứ hai của vế phải sẽ bằng một, khi yi = 0, thừa số thứ nhất sẽ bằng một. Đe mô hình tạo ra dự đoán khớp với dữ liệu đã cho nhất, ta cần tìm **w** để xác xuất này đạt giá tri cao nhất.

Xét toàn bộ tập huấn luyện với ma trận dữ liệu **X** = [**x**1,**x**2, ... , ***x****n*] 2 và vector nhãn tương ứng **y** = *[y1,y2,..., yN*]. Ta cần giải bài toán tối ưu

**w** = argmaxp(**y**|**X**; **w**) (8.8)

**w**

Đây chính là một bài toán MLE với tham số mô hình **w** cần được ước lượng. Ta có thể giải quyết bài toán này bằng cách giả sử các điểm dữ liệu độc lập nếu biết tham số mô hình. Đây cũng là giả sử thường được dùng khi giải các bài toán liên quan tới MLE:

*N N*

p(**y**|x; w) = np(yi|xi; w) = ỊỊ *ay* (1 - ai)1\_yi (8.9)

i=1 i=1

Lấy logarit tự nhiên, đổi dấu rồi lấy trung bình, ta thu được hàm số

1 1 X

*J*(w) = -Nlogp(**y**|x;w) = -*(yi*log*ai*+(1 \_yi)log(1 \_*ai))* (8.10) i=1

với chú ý rằng ai là một hàm số của **w** và **x**i. Hàm số này là hàm mất mát của hồi quy logistic. Vì đã đổi dấu sau khi lấy logarit, ta cần tìm **w** để *J*(**w**) đạt giá tri nhỏ nhất.

1. Tối ưu hàm mất mát

Bài toán tối ưu hàm mất mát của hồi quy logistic có thể được giải quyết bằng SGD. Tại mỗi vòng lặp, **w** được cập nhật dựa trên một điểm dữ liệu ngẫu nhiên. Hàm mất mát của hồi quy logistic với chỉ một điểm dữ liệu (**x**i,yi) và gradient của nó lần lượt là

*J*(w; xi,yi) = — (yi log *ai* + (1 - yi)log(l - Ui)) (8.11)

V

T/ „ X *( yi* 1 yi \ ~ \ ai yi /V7 ... \ *(o* 1

**w**J(w; *xi,yi)* ( *)(V****w****ai) Z(V****w****ai)* (8.12)

*ai* 1 - *ai* Ui(1 - *ai)*

ở đây ta đã sử dụng quy tắc chuỗi để tính gradient với ai = *f* (**x**T**w**). Đe cho biểu thức này đìn giản, ta sẽ tìm hàm *ai = f (****x****T***w**) sao cho mẫu số bi triệt tiêu.

Đặt *z =* ***X****T***w**, ta có

**V7** \_ da^ ~A \_ *d,a-.* /eiQA

V**w** *ai d* ( V**w** zi) *d xi* (8.13)

Tạm thời bỏ qua các chỉ số i, ta đi tìm hàm số *a = f (z)* sao cho

=a(1 \_a)

(8.14)

Nếu điều này xảy ra, mẫu số trong biểu thức [(8.12)](#bookmark885) sẽ bi triệt tiêu. Phưong trình

vi phân này không quá phức tạp. Thật vậy, [(8.14)](#bookmark887) tưìng đưìng với *dz*

*da*

a(1 — a) ị 1 ■ ' ) da

ya 1 — a/

log *a —* log(1 — a) *a* log V—

1 — a  
*a*

1*a*

*a*

*ez+C*

*a* 1 + *ez+C*

*dz*

*z* + *C*

*z* + C

ez+C

*ez+C* (1 - a)

1 + *le-z-c* = ' + C)

với C là một hằng số. Chọn C = 0, ta được *a* = f(**x**T**w**) = *ơ(z).* Đây chính là hàm sigmoid. Hồi quy logistic với hàm kích hoạt là hàm sigmoid được sử dụng phổ biến nhất. Mô hình này còn có tên là *hồi quy logistic sigmoid.* Khi nói hồi quy logistic, ta ngầm hiểu rằng đó chính là hồi quy logistic sigmoid.

Thay [(8.13)](#bookmark886) và [(8.14)](#bookmark887) vào [(8.12)](#bookmark885) ta thu được

V**w** J(**w**; **x**i,yi) = (ai - *yi)****x*** = *(ơ(****x****T***w**) - *yi)****x****i.* (8.15)

Từ đó, công thức cập nhật nghiệm cho hồi quy logistic sử dụng SGD là

**w** < **w -** *g(ai - yi)****x****i* = **w -** *g(ơ(****x****T***w**) - *yi)****x****i* (8.16)

với *g* là tốc độ học.

1. Hồi quy logistic với suy giảm trọng số

Một trong các kỹ thuật phổ biến giúp tránh quá khớp cho các mạng neuron là sử dụng *suy giảm trọng so (weight decay*). Đây là một kỹ thuật kiểm soát, trong đó một đại lượng tỉ lệ với bình phương chuẩn '2 của vector trọng số **w** được cộng vào hàm mất mát đe kiểm soát độ lớn của các hệ số. Hàm mất mát trở thành

*\~yi* log*ai -*(1 \_ yi)log(1 - *ai)*+1 Ilwll2

^(w) =

(8.17)

v i=1 k

Công thức cập nhật **w** bằng SGD trong hồi quy logistic với suy giảm trọng số là:

**w** < **w -** *g ((ơ(****x****T***w**) - yi)**x**i + A**w)**

(8.18)

1. Triển khai thuật toán trên Python

Hàm ước lượng xác suất đầu ra cho mỗi điểm dữ liệu và hàm tính giá tri hàm mất mát với weight decay có thể được thực hiện như sau trong Python.

**def** prob(w, X):

ri f! f!

*X: a 2d numpy array of shape (N, d). N datatpoint, each with size d*

*w: a 1d numpy array of shape (d)*

fl fl fl

**return** sigmoid(X.dot(w))

**def** loss(w, X, y, lam):

fl fl fl

*X, w as in prob*

*y: a 1d numpy array of shape (N). Each elem = 0 or 1*

fl fl fl

a = prob(w, X)

loss\_0 = -np.mean(y\*np.log(a) + (1-y)\*np.log(1-a))

weight\_decay = 0.5\*lam/X.shape[0]\*np.**sum**(w\*w)

**return** loss\_0 + weight\_decay

Từ công thức [(8.18)](#bookmark892), ta có thể thực hiện thuật toán tìm **w** cho hồi quy logistic như sau:

w = w\_old = w\_init

| **def** logistic\_regression(w\_init, X, y, lam, lr = 0.1,  *# lam: regulariza paramether, Ir: learning rate,* N, d = X.shape[0], X.shape[1] | nepoches = 2000): *nepoches: # epoches* |
| --- | --- |

*# store history of loss in loss\_hist*

loss\_hist = [loss(w\_init, X, y, lam)] ep = 0

**while** ep < nepoches:

ep += 1

mix\_ids = np.random.permutation(N) *# stochastic* **for** i **in** mix\_ids:

xi = X[i]

yi = y[i]

ai = sigmoid(xi.dot(w))

*# update*

w = w - lr\*((ai - yi)\*xi + lam\*w)

loss\_hist.append(loss(w, X, y, lam))

**if** np.linalg.norm(w - w\_old)/d < 1e-6:

**break**

w\_old = w

**return** w, loss\_hist

1. Hoi quy logistic cho ví dụ đầu bài

Ap dụng vào bài toán dự đoán đỗ/trượt ở đầu bài:

np.random.seed(2)

X = np.array([[0.50, 0.75, 1.00, 1.25, 1.50, 1.75, 1.75, 2.00, 2.25, 2.50,

2.75, 3.00, 3.25, 3.50, 4.00, 4.25, 4.50, 4.75, 5.00, 5.50]]).T

|  |  |
| --- | --- |
| y = np.array([0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, | 0, 1, 1, 1, 1, 1, 1]) |

*# bias trick*

Xbar = np.concatenate((X, np.ones((X.shape[0], 1))), axis = 1)

|  |  |
| --- | --- |
| w\_init = np.random.randn(Xbar.shape[1]) lam = 0.0001  w, loss\_hist = logistic\_regression(w\_init, Xbar, y, = 500) | lam, lr = 0.05, nepoches |

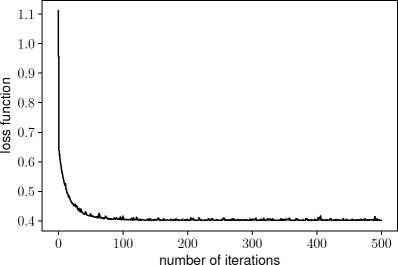
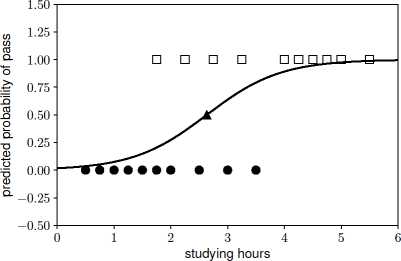
**print**('Solution of Logistic Regression:', w) **print**('Final loss:', loss(w, Xbar, y, lam))

Kết quả:

Solution of Logistic Regression: [ 1.54337021 -4.06486702] Final loss: 0.402446724975

Từ đây ta có thể rút ra xác suất thi đỗ dựa trên công thức:

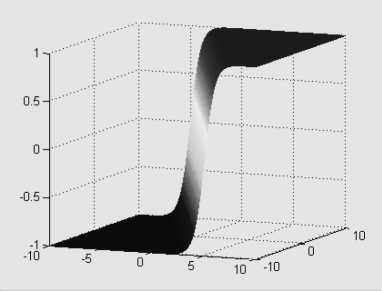
probability\_of\_pass ~ sigmoid(1.54 \* hours\_of\_studying - 4.06)



**(a)**

**(b)**

**Hình 8.3.** Nghiệm của hồi quy logistic cho bài toán dự đoán kết quả thi dựa trên thời gian học. (a) Dường nét lien the hiện xác suất thi đỗ dựa trên thời gian học. Diem tam giác the hiện ngưỡng ra quyết định đỗ/trượt. Diem này có the thay đoi tuỳ vào bài toán. (b) Giá trị của hàm mất mát qua các vòng lặp. Hàm mất mát giảm nhanh và hội tụ sóm.



**(a)** Dữ liệu cho bài toán phân loại trong không gian **(b)** Dồ thị hàm sigmoid trong không gian hai hai chiều. Chiều.

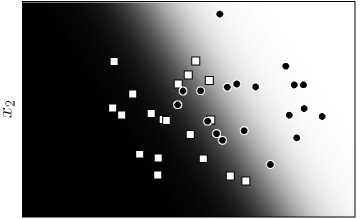
**Hình 8.4.** Ví dụ ve dữ liệu và hàm sigmoid trong không gian hai chiều.

Biểu thức này cũng chỉ ra rằng xác suất thi đỗ tang khi thời gian ôn tập tang, do sigmoid là một hàm đồng biến. Nghiệm của mô hình hồi quy logistic và giá tri hàm mất mát qua mỗi epoch được mô tả trên Hình 8.3.

1. Ví dụ với dữ liệu hai chiều

Giả sử có hai tập dữ liệu vuông và tròn phân bố trên mặt phẳng như trong Hình [8.4a](#bookmark902). Với dữ liệu đầu vào nằm trong không gian hai chiều, hàm sigmoid có dạng thác nước như trong Hình [8.4b](#bookmark902).

Kết quả dự đoán xác suất đầu ra khi áp dụng mô hình hồi quy logistic được minh họa như Hình [8.5](#bookmark907) với độ sáng của nền thể hiện xác suất điểm đó có nhãn tròn.

**Hình 8.5.** Ví dụ ve hồi quy logis­tic vói dữ liệu hai chiều. Vòng màu càng đen the hiện xác suất thuộc nhãn hình vuông càng cao. Vòng màu càng trắng the hiện xác suất thuộc nhãn hình tròn càng cao. Vòng biên giữa hai nhãn (khu vực màu xám) thê hiện các điêm thuộc vào mỗi nhãn vói xác suất thấp hơn.

X1

Màu đen đậm thể hiện giá tri gần bằng không, màu trắng thể hiện giá tri rất gần bằng một.

Nếu phải lựa chọn một ranh giới thay vì xác suất, ta thấy các đường thẳng nằm trong khu vực màu xám là các lựa chọn hợp lý. Ta sẽ chứng minh *ở* phần sau rằng tập hợp các điểm có cùng xác suất đầu ra tạo thành một siêu phẳng.

1. Tính chất của hồi quy logistic
2. *Hồi quy logistic thực ra là một thuật toán phân loại.*

Mặc dù trong tên có từ *hồi quy,* hồi quy logistic được sử dụng nhiều trong các bài toán phân loại. Sau khi tìm được mô hình, việc xác đinh nhãn *y* cho một điểm dữ liệu **x** được xác đinh bằng việc so sánh hai giá tri:

*p(y* = 1|x; **w**); *p(y* = 0|x; **w**) (8.19)

Nếu giá tri thứ nhất lớn hơn, ta kết luận điểm dữ liệu có nhãn một và ngược lại. Vì tổng hai giá tri này luôn bằng một nên ta chỉ cần xác đinh p(y = 1 |**x**; **w**) có lớn hơn 0.5 hay không.

1. *Đường ranh giới tạo bởi hồi quy logistic là một siêu phẳng.*

Thật vậy, giả sử những điểm có xác suất đầu ra lớn hơn 0.5 được gán nhãn một. Tập hợp các điểm này là nghiệm của bất phương trình:

1 T \_

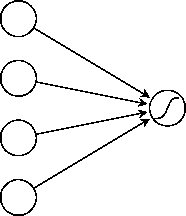
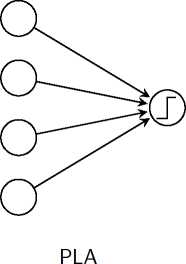
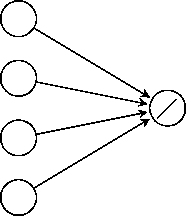
p(y = 1 |**x**; **w**) > 0.5 , ——*>* 0.5 , e“x w < 1 , **x**T**w** > 0

1 + **e~x**T**w**

Nói cách khác, tập hợp các điểm được gán nhãn một tạo thành một nửa không gian ***xTw*** *>* 0, tập hợp các điểm được gán nhãn không tạo thành nửa không gian còn lại. Ranh giới giữa hai nhãn là siêu phẳng **x**T**w** = 0.

Vì vậy, hồi quy logistic được coi như một bộ phân loại tuyến tính.

1. *Hồi quy logistic không yêu cầu giả thiết tách biệt tuyến tính.*



Hồi quy tuyến tính

Hồi quy logistic

**Hình 8.6.** Biểu diễn hồi quy tuyến tính, PLA, và hồi quy logistic dưới dạng neural network.

Một điểm cộng của hồi quy logistic so với PLA là nó không cần giả thiết dữ liệu hai tập hợp là tách biệt tuyến tính. Tuy nhiên, ranh giới tìm được vẫn có dạng tuyến tính. Vì vậy, mô hình này chỉ phù hợp với loại dữ liệu mà hai tập *gần* tách biệt tuyến tính.

1. *Ngưỡng ra quyết định có thể thay đoi.*

Hàm dự đoán đầu ra của các điểm dữ liệu mới có thể được viết như sau:

**def** predict(w, X, threshold = 0.5):

*predict output for each row of X*

*X: a numpy array of shape (N, d), threshold: 0 < threshold < 1 return a 1d numpy array, each element is 0 or 1* res = np.zeros(X.shape[0]) res[np.where(prob(w, X) > threshold)[0]] = 1 **return** res

Trong các ví dụ đã nêu, ngưỡng ra quyết đinh đều được lấy tại 0.5. Trong nhiều trường hợp, ngưỡng này có thể được thay đổi. Ví dụ, việc xác đinh các giao dich là lừa đảo của một công ty tín dụng là rất quan trọng. Việc phân loại nhầm một giao dich lừa đảo thành một giao dich thông thường gây ra hậu quả nghiêm trọng hơn chiều ngược lại. Trong bài toán đó, ngưỡng phân loại có thể giảm xuống còn 0.3. Nghĩa là các giao dich được dự đoán là lừa đảo với xác suất lớn hơn 0.3 sẽ được gán nhãn lừa đảo và cần được xử lý bằng các biện pháp khác.

1. Khi biểu diễn dưới dạng các mạng neuron, hồi quy tuyến tính, PLA và hồi quy logistic có thể được biểu diễn như trong Hình 8.6. Sự khác nhau chỉ nằm ở lựa chọn hàm kích hoạt.
2. Bài toán phân biệt hai chữ số viết tay

Xét bài toán phân biệt hai chữ số không và một trong bộ cơ sở dữ liệu MNIST. Trong mục này, class LogisticRegression trong thư viện scikit-learn sẽ được sử dụng. Trước tiên, ta cần khai báo các thư viện và tải về bộ cơ sở dữ liệu MNIST:

**import** numpy as np

**from** sklearn.datasets **import** fetch\_mldata

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

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** accuracy\_score

mnist = fetch\_mldata('MNIST original', data\_home='../../data/') N, d = mnist.data.shape

**print**('Total {:d} samples, each has {:d} pixels.'.**format**(N, d))

Kết quả:

Total 70000 samples, each has 784 pixels.

Có tổng cộng 70000 điểm dữ liệu trong tập dữ liệu MNIST, mỗi điểm là một mảng 784 phần tử tương ứng với 784 pixel. Mỗi chữ số từ không đến chín chiếm khoảng mười phần trăm. Chúng ta sẽ lấy ra tất cả các điểm ứng với chữ số không và một, sau đó chọn ngẫu nhiên 2000 điểm làm tập kiểm tra, phần còn lại đóng vai trò tập huấn luyện.

X\_all = mnist.data

y\_all = mnist.target

X0 = X\_all[np.where(y\_all == 0)[0]] *# all digit 0*

X1 = X\_all[np.where(y\_all == 1)[0]] *# all digit 1*

y0 = np.zeros(X0.shape[0]) *# class 0 label*

y1 = np.ones(X1.shape[0]) *# class 1 label*

X = np.concatenate((X0, X1), axis =0) *# all digits 0 and 1*

y = np.concatenate((y0, y1)) *# all labels*

*# split train and test*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=2000)

Tiếp theo, ta xây dựng mô hình hồi quy logistic trên tập huấn luyện và dự đoán nhãn của các điểm trong tập kiểm tra. Kết quả này được so sánh với nhãn thực sự của mỗi điểm dữ liệu để tính độ chính xác của bộ phân loại:

model = LogisticRegression(C = 1e5) *# C is inverse of lam* model.fit(X\_train, y\_train) y pred = model.predict(X\_test)

**print**("Accuracy %.2f %%" % (100\*accuracy\_score(y\_test, y pred.tolist())))

Kết quả:

DD

**Hình 8.7.** Các chữ số bị phân loại lỗi trong bài toán phân loại nhị phân vói hai chữ số không và một.

Accuracy 99.90 %

Như vậy, gần 100% các ảnh được phân loại chính xác. Điều này dễ hiểu vì hai chữ số không và một khác nhau rất nhiều.

Tiếp theo, ta cùng đi tìm những ảnh bị phân loại sai và hiển thi chúng:

mis = np.where((y\_pred - y\_test) !=0)[0]

Xmis = X\_test[mis, :]

**from** display\_network **import** \*

filename = 'mnist\_mis.pdf'

with PdfPages(filename) as pdf:

plt.axis('off')

A = display\_network(Xmis.T, 1, Xmis.shape[0])

f2 = plt.imshow(A, interpolation='nearest' ) plt.gray()

pdf.savefig(bbox\_inches='tight')

plt.show()

Chỉ có hai chữ số bị phân loại lỗi được cho trên Hình 8.7. Trong đó, chữ số không bị phân loại lỗi là dễ hiểu vì nó trông rất giống chữ số một.

1. Bài toán phân loại đa lớp

Hồi quy logistic được áp dụng cho các bài toán phân loại nhị phân. Các bài toán phân loại thực tế có thể có nhiều hơn hai nhãn dữ liệu, được gọi là bài toán *phân loại đa lớp (multi-class classification*). Hồi quy logistic cũng có thể được áp dụng vào các bài toán này bằng một vài kỹ thuật.

Có ít nhất bốn cách áp dụng các bộ phân loại nhị phân vào bài toán phân loại đa lớp.

1. one-vs-one

Ta có thể xây dựng nhiều bộ phân loại nhị phân cho từng cặp hai nhãn dữ liệu. Bộ thứ nhất phân biệt nhãn thứ nhất và nhãn thứ hai, bộ thứ hai phân biệt nhãn thứ nhất và nhãn thứ ba,... Có tổng cộng *P =* ~~c~~~~(C,-~~~~1~~~~)~~ bộ phân loại nhị phân cần xây dựng với *C* là số lượng nhãn. Cách thực hiện này được gọi là *one-vs-one.*

Với một điểm dữ liệu kiểm tra, ta dùng tất cả *P* bộ phân loại để dự đoán nhãn của nó. Kết quả cuối cùng có thể được xác đinh bằng cách xem điểm dữ liệu đó được gán nhãn nào nhiều nhất. Ngoài ra, nếu mỗi bộ phân loại có thể đưa ra xác suất giống hồi quy logistic, ta có thể tính tổng các xác suất mà điểm dữ liệu đó rơi vào mỗi nhãn. Chú ý rằng tổng các xác suất là *P* thay vì một bởi có *P* bộ phân loại khác nhau.

Cách làm này không lợi về tính toán vì số bộ phân loại phải huấn luyện tang nhanh khi số nhãn tang lên. Hơn nữa, điều không hợp lý xảy ra nếu một chữ số có nhãn bằng một được đưa vào bộ phân loại giữa hai nhãn chữ số nam và sáu.

1. Phân loại phân tầng

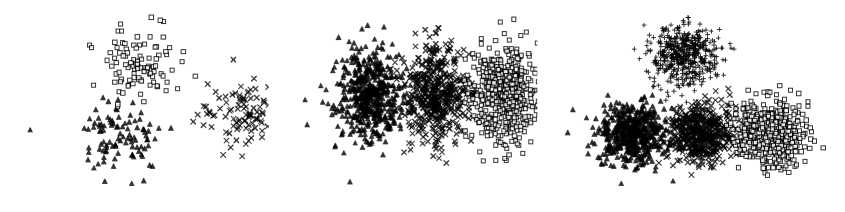
One-vs-one yêu cầu xây dựng ~~Cl;~~ bộ phân loại khác nhau. Đe giảm số bộ phân loại cần xây dựng, ta có thể dùng phương pháp *phân tầng.* Y tưởng của phương pháp này có thể được thấy qua ví dụ sau.

Xét bài toán phân loại bốn chữ số {4, 5, 6, 7} trong MNIST. Vì chữ số 4 và 7 khá giống nhau, chữ số 5 và 6 khá giống nhau nên trước tiên ta xây dựng bộ phân loại giữa {4, 7} và {5, 6}. Sau đó xây dựng thêm hai bộ phân loại để xác đinh từng chữ số trong mỗi nhóm. Tổng cộng, ta cần ba bộ phân loại nhi phân so với sáu bộ như khi sử dụng one-vs-one.

Có nhiều cách chia nhỏ tập dữ liệu ban đầu ra các cặp tập con. Cách phân tầng có ưu điểm là giảm số bộ phân loại nhi phân cần xây dựng. Tuy nhiên, cách làm này có một hạn chế lớn: nếu chỉ một bộ phân loại cho kết quả sai thì kết quả cuối cùng chắc chắn sẽ sai. Ví dụ, nếu một ảnh chứa chữ số 5 bi phân loại lỗi bởi bộ phân loại đầu tiên thì cuối cùng nó sẽ bi nhận nhầm thành 4 hoặc 7.

1. Mã hoá nhị phân

Có một cách tiếp tục giảm số bộ phân loại là *mã hoá nhị phân.* Trong phương pháp này, mỗi nhãn được mã hoá bởi một số nhi phân. Ví dụ, nếu có bốn nhãn thì chúng được mã hoá bởi 00, 01, 10, và 11. Số bộ phân loại nhi phân cần xây dựng chỉ là *m =* piog2(C)e trong đó *C* là số nhãn, I a I là số nguyên nhỏ nhất không nhỏ hơn *a.* Bộ phân loại đầu tiên giúp xác đinh bit đầu tiên của nhãn, bộ thứ hai xác đinh bit tiếp theo,.... Cách làm này sử dụng một số lượng nhỏ nhất các bộ phân loại nhi phân. Tuy nhiên, một điểm dữ liệu chỉ được phân loại đúng khi mọi bộ phân loại nhi phân dự đoán đúng bit tương ứng. Hơn nữa, nếu số nhãn không phải là lũy thừa của hai, mã nhi phân nhận được có thể không tương ứng với nhãn nào.



a) b) c)

**Hình 8.8.** Ví dụ về phân phối của các tập dữ liệu trong bài toán phân loại đa lóp.

1. one-vs-rest

Kỹ thuật được sử dụng nhiều nhất là *one-vs-rest.* Cụ thể, *C* bộ phân loại nhi phân được xây dựng tương ứng với các nhãn. Bộ thứ nhất xác đinh một điểm có nhãn thứ nhất hay không, hoặc xác suất để một điểm có nhãn đó. Tương tự, bộ thứ hai xác đinh điểm đó có nhãn thứ hai hay không hoặc xác xuất có nhãn thứ hai là bao nhiêu. Nhãn cuối cùng được xác đinh theo nhãn mà điểm đó rơi vào với xác suất cao nhất.

Hồi quy logistic trong thư viện scikit-learn có thể được áp dụng trực tiếp vào các bài toán phân loại đa lớp với kỹ thuật one-vs-rest. Với MNIST, ta có thể dùng hồi quy logistic kết hợp với one-vs-rest (mặc đinh) như sau:

X\_train, X\_test, y\_train, y\_test = \ train\_test\_split(X\_all, y\_all, test\_size=10000)

model = LogisticRegression(C = 1e5) *# C is inverse of lam*

model.fit(X\_train, y\_train)

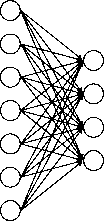
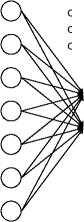
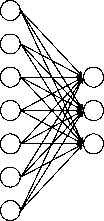
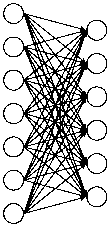
y pred = model.predict(X\_test)

**print**("Accuracy %.2f %%" % (100[[2]](#footnote-3)accuracy\_score(y\_test, y pred.tolist())))

Kết quả thu được tương đối thấp, khoảng 91.7%. Phương pháp KNN đơn giản hơn đã có độ chính xác khoảng 96%. Điều này chứng tỏ one-vs-rest không làm việc tốt trong trường hợp này.

1. Thảo luận
2. Kết hỢp các phương pháp trẽn

Trong nhiều trường hợp, ta cần kết hợp nhiều kỹ thuật trong số bốn kỹ thuật đã đề cập. Xét ba ví dụ trong Hình 8.8.

**Hình 8.9.** Mô hình neural net­work cho các kỹ thuật sử dụng các bộ phân loại nhị phân cho bài toán phân loại đa lóp.

1

2

vs

1

3

vs

1

4

vs

2

3

vs

2

4

vs

3

4

vs

one-vs-one

1,2 vs 3,4

1 vs 2

3 vs 4

Phân tầng

bit 0 vs bit 1

bit 0 vs bit 1

class 1 = '00' class 2 = '01' class 3 = '10' class 4 = '11'

1

2

3

4

vs vs vs vs

rest

rest rest rest

one-vs-rest

Mã hoá nhị phân

* Hình 8.8b: One-vs-rest không phù hợp vì tập dữ liệu ở giữa và hợp của hai tập còn lại là không (gần) tách biệt tuyến tính. Lúc này, one-vs-one hoặc phân tầng phù hợp hơn.
* Hình 8.8c: Tương tự như trên, có ba tập dữ liệu thẳng hàng nên one-vs-rest sẽ không phù hợp. Trong khi đó, one-vs-one vẫn hiệu quả vì từng cặp nhãn dữ liệu là (gần) tách biệt tuyến tính. Tương tự, phân tầng cũng làm việc nếu ta phân chia các nhãn một cách hợp lý. Ta cũng có thể kết hợp nhiều phương pháp. Ví dụ, dùng one-vs-rest để tách nhãn ở hàng trên ra khỏi ba nhãn thẳng hàng ở dưới. Ba nhãn còn lại có thể tiếp tục được phân loại bằng các phương pháp khác. Tuy nhiên, khó khan vẫn nằm ở việc phân nhóm như thế nào.

Với bài toán phân loại đa lớp, nhìn chung các kỹ thuật sử dụng các bộ phân loại nhi phân ít mang lại hiệu quả.

1. Biểu diễn dưới dạng mạng neuron

Lấy ví dụ bài toán có bốn nhãn dữ liệu {1, 2, 3, 4}; ta có thể biểu diễn các kỹ thuật đã được đề cập dưới dạng mạng neuron như trong Hình [8.9](#bookmark781). Mỗi nút ở tầng đầu ra thể hiện đầu ra của một bộ phân loại nhi phân.

Các mạng neuron này đều có nhiều nút ở tầng đầu ra, vector trọng số **w** đã trở thành *ma trận trọng so* **W**. Mỗi cột của **W** tương ứng với vector trọng số của một nút đầu ra. Các bộ phân loại nhi phân này có thể được xây dựng đồng thời.

Nếu chúng là các bộ hồi quy logistic, công thức cập nhật theo SGD:

**w** < **w -** *y(ai - yi)****x*** (8.20)

có thể được tổng quát thành

**W** < **W -** ^**X**i(**a**i - yi)T. (8.21)

Với **W**, ***y****i,* ***a****i* lần lượt là ma trận trọng số, vector đầu ra thực sự và vector đầu ra dự đoán ứng với dữ liệu **x**i. Chú ý rằng vector **y**i là một vector nhi phân, vector **a**i gồm các phần tử nằm trong khoảng (0,1).

*Chú ý*: Số hạng thứ hai trong [(8.21)](#bookmark959) không thể là (**a**i — **y**i)**x**T vì ma trận này khác chiều với **W**. Số hạng này cần là tích của hai vector: vector thứ nhất cần có cùng số hàng với **W**, tức chiều của dữ liệu **x**i; vector thứ hai cần phù hợp với số cột của **W**, tức số nút ở tầng đầu ra.

***Bài 9***

Hồi quy softmax

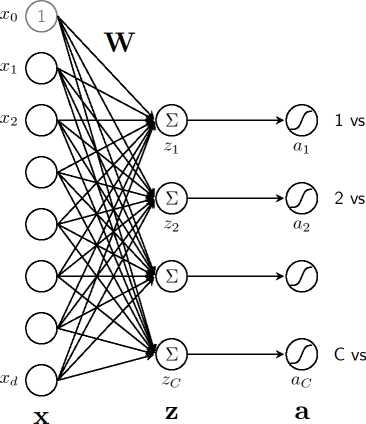
Các bài toán phân loại thực tế thường có nhiều lớp dữ liệu. Như đã thảo luận trong Bài [8](#bookmark843), các bộ phân loại nhi phân tuy có thể kết hợp với nhau để giải quyết các bài toán phân loại đa lớp nhưng chúng vẫn có những hạn chế nhất đinh. Trong bài này, một phương pháp mở rộng của hồi quy logistic có tên là *hồi quy softmax* sẽ được giới thiệu nhằm khắc phục những hạn chế đã đề cập. Một lần nữa, mặc dù trong tên có chứa từ “hồi quy”, hồi quy softmax được sử dụng cho các bài toán phân loại. Hồi quy softmax là một trong những thành phần phổ biến nhất trong các bộ phân loại hiện nay.

1. Giới thiệu

Với bài toán phân loại nhi phân sử dụng hồi quy logistic, đầu ra của mạng neural là một số thực trong khoảng (0,1), có ý nghĩa như xác suất để đầu vào thuộc một trong hai lớp. Y tưởng này cũng có thể mở rộng cho bài toán phân loại đa lớp, ở đó có *C* nút ở tầng đầu ra và giá tri mỗi nút đóng vai trò như xác suất để đầu vào rơi vào lớp tương ứng. Như vậy, các đầu ra này liên kết với nhau qua việc chúng đều là các số dương và có tổng bằng một. Mô hình hồi quy softmax thảo luận trong bài này đảm bảo tính chất đó.

Nhắc lại biểu diễn dưới dạng mạng neural của kỹ thuật *one-vs-rest* như trong Hình 9.1. Tầng đầu ra có the tách thành hai *tầng con* **z** và **a**. Mỗi thành phần của tầng con thứ hai *ai* chỉ phụ thuộc vào thành phần tương ứng ở tầng con thứ nhất *zi* thông qua hàm sigmoid *ai =* a(zi). Các giá tri đầu ra *ai* đều là các số dương nhưng vì không có ràng buộc giữa chúng, tổng các xác suất này không đảm bảo bằng một.

Các mô hình hồi quy tuyến tính, PLA, và hồi quy logistic chỉ có một nút ở tầng đầu ra. Trong các trường hợp đó, tham số mô hình chỉ là một vector **w**.



rest

rest

rest

*x*

*wij*

*z*

*w0j*: hệ số điều chỉnh *d:* số chiều dữ liệu *C:* số lớp dữ liệu

**x** e Rd+1

**W** e R(d+1)xơ

Zi = **w**T **x**

**z** = **W**T **x** e RC

*ai =* sigmoid(zi) e R

0 < ai < 1

**Hình 9.1.** Phân loại đa lóp vói hồi quy logistic và one-vs-rest.

Trong trường hợp tầng đầu ra có nhiều hơn một nút, tham số mô hình sẽ là tập hợp tham số ***W****ị* ứng với từng nút. Lúc này, ta có một *ma trận trọng số* **W** = [**w**i, **w**2,... , **w**C], mỗi cột ứng với một nút ở tầng đầu ra.

9.2. Hàm softmax

9.2.1. Hàm softmax

Chúng ta cần một mô hình xác suất sao cho với mỗi đầu vào **x**, *ai* thể hiện xác suất để đầu vào đó rơi vào lớp thứ i. Vậy điều kiện cần là các *ai* phải dương và tổng của chúng bằng một. Ngoài ra, ta sẽ thêm điều kiện giá tri *zi =* ***xTw****i* càng lớn thì xác suất dữ liệu rơi vào lớp thứ *i* càng cao. Điều kiện cuối này chỉ ra rằng ta cần một quan hệ đồng biến.

Chú ý rằng *zi* có thể nhận giá tri cả âm và dương vì nó là một tổ hợp tuyến tính các thành phần của vector đặc trưng **x**. Một hàm số khả vi đồng biến đơn giản có thể biến *zi* thành một giá tri dương là hàm exp(zi) = *ezi*. Hàm số này không những khả vi mà còn có đạo hàm bằng chính nó, việc này mang lại nhiều lợi ích khi tối ưu. Điều kiện tổng các ai bằng một có thể được đảm bảo nếu

\_ exp(zi)

EC=1 exP(zj) ’

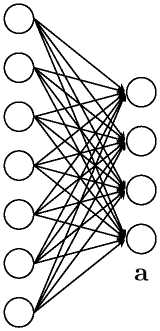
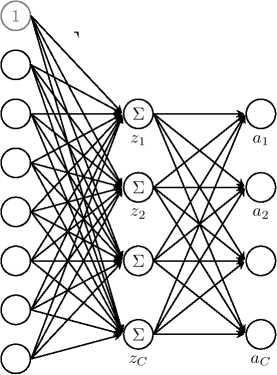
Vz = 1,2,...,C.

(9.1)

Mối quan hệ này thoả mãn tất cả các điều kiện đã xét: các đầu ra ai dương, có tổng bằng một và giữ được thứ tự của zi. Hàm số này được gọi là *hàm softmax.* Lúc này, ta có thể coi rằng

*p(yk = i\****x****k;* ***W****) = ai*

(9.2)



Xo

X1

X2

**W**

X

X

Z2

**x**

**z a**

*w0j*: hệ số điều chỉnh *d:* số chiều dữ liệu

*C*: số lóp dữ liệu

**x** e Rd+1

**W** e R(d+1)xC

\_\_.T

*Zi* ***=* w**T **x**

**z** = ***W****T****x*** *E* RC

**a** = softmax(**z**) e RC

*C*

*ai >* 0, ^2 *ai* = 1

Xi

*wij*

*zj*

**z** = softmax(**W**T **x**)

dạng đơn giản

**—►**

**x**

**Hình 9.2.** Mô hình hồi quy softmax dưới dạng neural network.

Trong đó, *p(y =* i|**x**; **W**) được hiểu là xác suất để một điểm dữ liệu **x** rơi vào lớp thứ *i* nếu biết tham số mô hình là ma trận trọng số **W**. Hình [9.2](#bookmark967) the hiện mô hình hồi quy softmax dưới dạng mạng neural. Mô hình này khác one-vs-rest nằm ở chỗ nó có các liên kết giữa mọi nút của hai tầng con **z** và **a**.

1. Xây dựng hàm softmax trong Python

Dưới đây là một đoạn code thực hiện hàm softmax. Đầu vào là một ma trận với mỗi *hàng* là một vector **z**, đầu ra cũng là một ma trận mà mỗi hàng có giá tri là **a** = softmax(**z**). Các giá tri của **z** còn được gọi là *score*:

**import** numpy as np

**def** softmax(Z):

fl fl fl

*Compute softmax values for each sets of scores in Z. each column of Z is a set of scores.*

*Z: a numpy array of shape (N, C) return a numpy array of shape (N, C) fl fl fl*

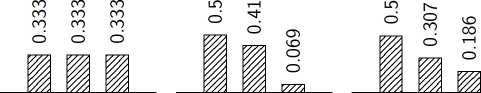
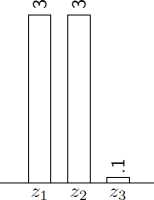
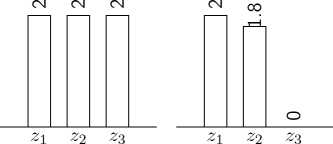
e\_Z = np.exp(Z)

A = e\_Z / e\_Z.**sum**(axis = 1, keepdims = True)

**return** A

1. Một vài ví dụ

Hình 9.3 mô tả một vài ví dụ về mối quan hệ giữa đầu vào và đầu ra của hàm softmax. Hàng trên thể hiện các score *zi* với giả sử rằng số lớp dữ liệu là ba. Hàng dưới thể hiện các giá tri đầu ra *ai* của hàm softmax.

*Ũ1 Ũ2 Ũ3*

softmax

09

ai *Ũ2 Ũ3 Ũ1 Ũ2 Ũ3 Ũ1 Ũ2 Ũ3*

**Hình 9.3.** Một số ví dụ về đầu vào và đầu ra của hàm softmax.

Có một vài quan sát như sau:

* Cột 1: Nếu các *zi* bằng nhau (bằng 2 hoặc một số bất kỳ) thì các *ai* cũng bằng nhau và bằng 1/3.
* Cột 2: Nếu giá tri lớn nhất trong các *zi* là *Z1* vẫn bằng 2, thì mặc dù xác suất tương ứng *a1* vẫn là lớn nhất, nó đã tang lên hơn 0.5. Sự chênh lệch ở đầu ra là đáng kể, nhưng thứ tự tương ứng không thay đổi.
* Cột 3: Khi các giá tri zi là âm thì các giá tri *ai* vẫn là dương và thứ tự vẫn được đảm bảo.
* Cột 4: Nếu *Z1* = *Z2* thì a1 = *a2.*

1. Phiên bản ổn định hơn của hàm softmax

Khi một trong các zi quá lớn, việc tính toán exp(zi) có thể gây ra hiện tượng tràn số, ảnh hưởng lớn tới kết quả của hàm softmax. Có một cách khắc phục hiện tượng này dựa trên quan sát

exp(zi) exp(—c) exp(zi) exp(zi — c)

(9.3)

EC.1 exp(zj) exp(-c)£C i exp(z;) EC\_1 exp(z, - c)

với *c* là một hằng số bất kỳ. Từ đây, một kỹ thuật đơn giản giúp khắc phục hiện tượng tràn số là trừ tất cả các zi đi một giá tri đủ lớn. Trong thực nghiệm, giá tri đủ lớn này thường được chọn là *c =* maxi *zi.* Ta có thể cải tiến đoạn code cho hàm softmax phía trên bằng cách trừ mỗi hàng của ma trận đầu vào Z đi giá tri lớn nhất trong hàng đó. Ta có phiên bản ổn đinh hơn là softmax\_stable:

**def** softmax\_stable(Z):

fl fl fl

*Compute softmax values for each set of scores in Z.*

*each row of Z is a set of scores.*

fl fl fl

e\_Z = np.exp(Z - np.**max**(Z, axis = 1, keepdims = True))

A = e\_Z / e\_Z.**sum**(axis = 1, keepdims = True)

**return** A

1. Hàm mất mát và phương pháp tối ưu
2. Entropy chéo

Đầu ra của mạng softmax, **a** = softmax(**W**T**x**), là một vector có số phần tử bằng số lớp dữ liệu. Các phần tử của vector này là các số dương có tổng bằng một, thể hiện xác suất để điểm đầu vào rơi vào từng lớp dữ liệu. Với một điểm dữ liệu huấn luyện thuộc lớp thứ c, chúng ta mong muốn xác suất tương ứng với lớp này càng cao càng tốt, tức càng gần một càng tốt. Việc này kéo theo các phần tử còn lại gần với không. Một cách tự nhiên, đầu ra thực sự **y** là một vector có tất cả các phần tử bằng không trừ phần từ ở vi trí thứ c bằng một. Cách biểu diễn nhãn dưới dạng vector này được gọi là mã hoá *one-hot.*

Hàm mất mát của hồi quy softmax được xây dựng dựa trên bài toán tối thiểu sự khác nhau giữa đầu ra dự đoán **a** và đầu ra thực sự **y** ở dạng one-hot. Khi cả hai là các vector thể hiện xác suất, khoảng cách giữa chúng thường được đo bằng một hàm số được gọi là *entropy chéo H*(**y**, **a**). Đặc điểm nổi bật của hàm số

Một đặc điểm nổi bật là nếu cố đinh **y**, hàm số sẽ đạt giá tri nhỏ nhất khi **a** = **y**, và càng lớn nếu **a** càng khác **y**.

Entropy chéo giữa hai vector phân phối **p** và **q** rời rạc được đinh nghĩa bởi

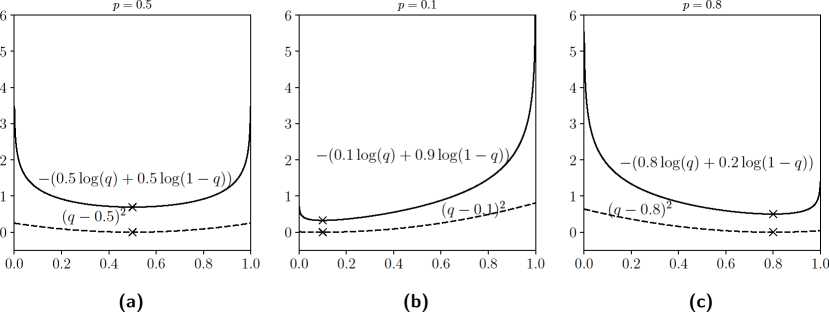
C

H (**p**; **q**) = *-^2 Pi* log q (9.4)

*i=1*

Hình 9.4 thể hiện ưu điểm của hàm entropy chéo so với hàm bình phương khoảng cách Euclid. Đây là ví dụ trong trường hợp *C* =2 và *p1* lần lượt nhận các giá tri 0.5,0.1 và 0.8 và p2 = 1 — p1. Có hai nhận xét quan trọng:

• Giá tri nhỏ nhất của cả hai hàm số đạt được khi *q = p* tại hoành độ các điểm được đánh dấu.



**Hình 9.4.** So sánh hàm entropy chéo (đường nét liền) và hàm bình phưong khoảng cách (đường nét đứt). Các điem được đánh dau the hiện điem cực tiêu toàn cục của mỗi hàm. Càng xa điếm cực tiếu toàn cục, khoảng cách giữa hai hàm số càng lớn.

• Nhận thấy rằng hàm entropy chéo nhận giá tri rất cao, tức mất mát rất cao, khi *q* ở xa *p.* Sự chênh lệch giữa các mất mát ở gần hay xa nghiệm của hàm bình phương khoảng cách *(q —* p)2 là ít đáng ke hơn. Về mặt tối ưu, hàm entropy chéo sẽ cho nghiệm gần với p hơn vì những nghiệm ở xa gây ra mất mát lớn.

Hai tính chất trên đây khiến hàm entropy chéo được sử dụng rộng rãi khi tính khoảng cách giữa hai phân phối xác suất. Tiếp theo, chúng ta sẽ chứng minh nhận đinh sau.

*Cho* **p** 2 R++ *là một vector với các thành phần dương có tểng bằng một. Bài toán tối ưu*

**q** = arg min *H* (**p**; **q**)

**q**

c

thoả mãn: *qi* = 1; *qi >* 0

*i=1*

*có nghiệm* **q** = **p**.

Bài toán này có thể giải quyết bằng phương pháp nhân tử Lagrange. Lagrangian của bài toán tối ưu này là

*C C*

*C(qi;q2;...;qc,* A) = *- ^2 Pi* log(qi) + A(J2 qi \_ 1)

*i=1 i=1*

Ta cần giải hệ phương trình

V7 *,r(„* A) \_ n Ợí + A = 0; \* *=1,---,C*

*r91,...,gc ,xL(qi, ■■■,qc; A) = 0 Qi*

( qi + q2 + • • • + qc = 1

Từ phương trình thứ nhất ta có *pi = Aqi.* Vì vậy, 1 VC 1 *pi = C*1 *qi* = A )

A = 1. Điều này tương đương với *qi = pi, Vi.*

Chú ý

1. *Hàm entropy chéo không có tính đối xứng H*(**p**, **q**) = *H*(**q**, **p**). *Diều này có thể nhận ra từ việc các thành phần của* **p** *trong công thức* [(9.4)](#bookmark1007) *có thể nhận giá trị bằng không, trong khi các thành phần của* **q** *phải là dương vì* log(0) *không xác định. Chính vì vậy, khi sử dụng entropy chéo trong các bài toán phân loại,* **p** *là đầu ra thực sự ở dạng one-hot,* **q** *là đầu ra dự đoán. Trong các thành phần thể hiện xác suất của* **q**, *không có thành phần nào tuyệt đối bằng một hoặc tuyệt đối bằng không do hàm* exp *luôn trả về một giá trị dương.*
2. *Khi* **p** *là một vector ở dạng one-hot, giả sử chỉ có pc =* 1, *biểu thức entropy chéo trở thành —* log(qc). *Biểu thức này đạt giá trị nhỏ nhất nếu qc* =1, *điều này không xảy ra vì nghiệm này không thuộc miền xác định của bài toán. Tuy nhiên, giá trị entropy chéo tiệm cận tới không khi qc tiến đến một, tức zc rất rất lớn so với các zi còn lại.*
3. Xây dựng hàm mất mát

Trong trường hợp có *C* lớp dữ liệu, mất mát giữa đầu ra dự đoán và đầu ra thực sự của một điểm dữ liệu **x**i với nhãn **y**i được tính bởi

C

Ji(W) = *J*(W; xi, ***y****ì) = yji log(aji)* (9-5)

j=i

với *yji* và *ơỷi* lần lượt là phần tử thứ *j* của vector xác suất **y**i và **a**i. Nhắc lại rằng đầu ra **a**i phụ thuộc vào đầu vào **x**i và ma trận trọng số **W**. Tới đây, nếu để ý rằng chỉ có đúng một j sao cho *yji =* 1, Vi, biểu thức [(9.5)](#bookmark1017) chỉ còn lại một số hạng tương ứng với giá tri j đó. Đe tránh việc sử dụng quá nhiều ký hiệu, chúng ta giả sử rằng yi là nhãn của điểm dữ liệu **x**i (các nhãn là các số tự nhiên từ 1 tới C), khi đó j chính bằng *yi.* Sau khi có ký hiệu này, ta có thể viết lại

Ji(W) = *- log(ayi,i)* (9.6)

với *ayi;i* là phần tử thứ yi của vector **a**i.

Khi sử dụng toàn bộ tập huấn luyện **x**i, ***y****i,i =* 1,2,..., *N*, hàm mất mát của hồi quy softmax được xác đinh bởi

1 A . .

J(**W**; **X**, **Y**) = *- NY,* log(a,i,i) (9.7)

i=1

ở đây, ma trận trọng số **W** là biến cần tối ưu. Hàm mất mát này có gradient khá gọn, kỹ thuật tính gradient gần giống với hồi quy logistic. Đe tránh quá khớp, ta cũng có thể sử dụng cơ chế kiểm soát suy giảm trọng số:

*JI.W-.* **X**. **Y**) = -N E'og^..\*) + *1* II**WI**IF (9.8)

\i=1 /

Trong các mục tiếp theo, chúng ta sẽ làm việc với hàm mất mát [(9.7)](#bookmark1018). Việc mở rộng cho hàm mất mát với cơ chế kiểm soát [(9.8)](#bookmark1019) không phức tạp vì gradient của số hạng kiểm soát I**||W||**F đơn giản là A**W**. Hàm mất mát [(9.7)](#bookmark1018) có thể được thực hiện trên Python như sau:

**def** softmax\_loss(X, y, W):

*W: 2d numpy array of shape (d, C), each column correspoding to one output node*

*X: 2d numpy array of shape (N, d), each row is one data point*

*y: 1d numpy array — label of each row of X*

fl fl fl

A = softmax\_stable(X.dot(W))

id0 = **range**(X.shape[0]) *# indexes in axis 0, indexes in axis 1 are in y* **return** -np.mean(np.log(A[id0, y]))

Chú ý

1. *Khi biểu diễn dưới dạng toán học, mỗi điểm dữ liệu là một cột của ma trận* ***X****; nhưng khi làm việc với numpy, mỗi điểm dữ liệu được đọc theo* axis = 0 *của mảng hai chiều* X. *Việc này thống nhất với các thư viện scikit-learn hay tensorflow ở việc* X[i] *được dùng để chỉ điểm dữ liệu thứ* i, *tính từ 0. Tức là, nếu có N điểm dữ liệu trong không gian d chiều thì* ***X*** *2 RdxN, nhưng* X.shape == (N, d).
2. **W** 2 RdxC*,* W.shape == (d, C).
3. ***WTX*** *sẽ được biểu diễn bởi* X.dot(W), *và có* shape == (N, C).
4. *Khi làm việc với phép nhãn ma trận hay mảng nhiều chiều trong numpy, cần chú ý tới kích thước của các ma trận sao cho các phép nhãn thực hiện được.*
5. Tối ưu hàm mất mát

Hàm mất mát sẽ được tối ưu bằng gradient descent, cụ thể là mini-batch gradient descent. Mỗi lần cập nhật của mini-batch gradient descent được thực hiện trên một batch có số phần tử 1 < *k* N .Để tính được gradient của hàm mất mát theo tập con này, trước hết ta xem xét gradient của hàm mất mát tại một điểm dữ liệu.

Với chỉ một cặp dữ liệu (**x**^, **y**»), ta dùng [(9.5)](#bookmark1017)



C C

Ji(w) = -22 *yji log(aji) =* -52 yjilog

j=i j=i

exp(**x**T wj)  
EC=i exP(xT *wk*)

*C / / C*

*=* ~ 52 I yjixT *w ~* yji log (52 exp(xT ***w***) j=i \ \k=i

C / *C* !

= “52 yjixT***w***+log Eexp(xTwk)) (9-9)

*j=i k=i*

Tiếp theo ta sử dụng công thức

*V****w****Ji* (**W**) = [V**W**1 *Ji(w),* V**W**2 *Ji(w),...,* V**W**C *Ji* (w)] . (9.10)

Trong đó, gradient theo từng cột của **W**j có thể tính được dựa theo [(9.9)](#bookmark968) và quy tắc chuỗi:

V**W**j Ji(w) *yjixi* I

exP(xT wj)  
EC=i exP(xT *wk*)

**x**i

yjixi + ajixi xi(aji yji)

ejixi (với eji aji yji)

(9.11)

Giá tri *eji* = *aji — yji* chính là sự sai khác giữa đầu ra dự đoán và đầu ra thực sự tại thành phần thứ *j*. Kết hợp [(9.10)](#bookmark969) và [(9.11)](#bookmark970) với **e**i = **ai — y**i, ta có

Vw *Ji*(**W**) = **x**i[eii, e2i,..., eCi] = **x**i**e**T

(9.12)

(9.13)

) VwJ(**W**) = ***1^2* x**i**e**T = N**XE**t

i=i

với **E** = **A — Y**. Công thức đơn giản này giúp cả batch gradient descent và mini­batch gradient descent có the dễ dàng được áp dụng. Trong trường hợp mini-batch gradient, giả sử kích thước batch là *k,* ký hiệu **X**b 2 Rdxk, **Y**b 2 {0,1}Cxk, **A**b 2 RCxk là dữ liệu ứng với một batch, công thức cập nhật cho một batch sẽ là

**W**^**W -** -E **X**b**E**T (9.14)

với *Nb* là kích thước của mỗi batch và *y* là tốc độ học.

Hàm số tính gradient theo **w** trong Python có thể được thực hiện như sau: **def** softmax\_grad(X, y, W):

fl ff ff

*W: 2d numpy array of shape (d, C),*

*each column correspoding to one output node*

*X: 2d numpy array of shape (N, d), each row is one data point*

*y: 1d numpy array — label of each row of X*

A = softmax\_stable(X.dot(W)) *# shape of (N, C)* id0 = **range**(X.shape[0])

A[id0, y] -= 1 *# A - Y, shape of (N, C)*

**return** X.T.dot(A)/X.shape[0]

Từ đó, ta có the viết hàm số huấn luyện hồi quy softmax như sau:

| **def** softmax\_fit(X, y, W, lr = 0.01, nepochs = 100, tol = 1e-5, 10): | batch\_size = |
| --- | --- |

W\_old = W.copy()

ep = 0

loss\_hist = [loss(X, y, W)] *# store history of loss*

N = X.shape[0]

nbatches = **int** (np.ceil(**float**(N)/batch\_size)) **while** ep < nepochs:

ep += 1

mix\_ids = np.random.permutation(N) *# stochastic* **for** i **in range**(nbatches):

|  |  |
| --- | --- |
| *# get the i-th batch*  batch\_ids = mix\_ids[batch\_size\*i:**min**(batch\_size\*(i+ X\_batch, y\_batch = X[batch\_ids], y[batch\_ids] | 1), N)] |

W -= lr\*softmax\_grad(X\_batch, y\_batch, W) *# gradient descent* loss\_hist.append(softmax\_loss(X, y, W))

**if** np.linalg.norm(W - W\_old)/W.size < tol:

**break**

W\_old = W.copy()

**return** W, loss\_hist

Cuối cùng là hàm dự đoán nhãn của các điếm dữ liệu mới. Nhãn của một điếm dữ liệu mới được xác đinh bằng chỉ số của lớp dữ liệu có xác suất rơi vào cao nhất, và cũng chính là chỉ số của score cao nhất.

**def** pred(W, X):

ri ff ff

*predict output of each columns of X . Class of each x\_i is determined by location of the max probability. Note that classes are indexed from 0.*

fl fl fl

**return** np.argmax(X.dot(W), axis =1)

1. Ví dụ trên Python

Để minh hoạ ranh giới của các lớp dữ liệu khi sử dụng hồi quy softmax, chúng ta cùng làm một ví dụ nhỏ trong không gian hai chiều với nam lớp dữ liệu:

C, N = 5, 500 *# number of classes and number of points per class*

means = [[2, 2], [8, 3], [3, 6], [14, 2], [12, 8]]

cov = [[1, 0], [0, 1]]

X0 = np.random.multivariate\_normal(means[0], cov, N)

X1 = np.random.multivariate\_normal(means[1], cov, N)

X2 = np.random.multivariate\_normal(means[2], cov, N)

X3 = np.random.multivariate\_normal(means[3], cov, N)

161

14­

12­

10 -

*ự) ữ.*

8 -

g

6 -

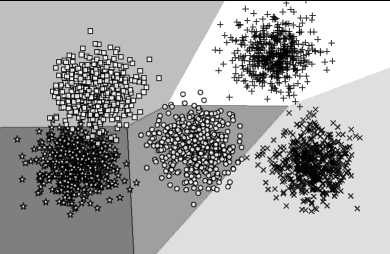
4­

2­

0-

0 20 40 60 80 10o

number of epoches

**(a)**

**(b)**

**Hình 9.5.** Ví dụ ve sử dụng hồi quy softmax cho năm lóp dữ liệu. (a) Giá trị hàm mat mát qua các epoch. (b) Ket quả phân loại cuối còng.

X4 = np.random.multivariate\_normal(means[4], cov, N)

X = np.concatenate((X0, X1, X2, X3, X4), axis =0) *# each row is a datapoint* Xbar = np.concatenate((X, np.ones((X.shape[0], 1))), axis =1) *# bias trick*

y = np.asarray([0]\*N + [1]\*N + [2]\*N+ [3]\*N + [4]\*N) *# label*

W\_init = np.random.randn(Xbar.shape[1], C)

W, loss\_hist = softmax\_fit(Xbar, y, W\_init, lr = 0.05)

Giá tri của hàm mất mát qua các epoch được cho trên Hình [9.5a](#bookmark1036). Ta thấy rằng hàm mất mát giảm rất nhanh sau đó hội tụ. Các điểm dữ liệu huấn luyện của mỗi lớp là các điểm có hình dạng khác nhau trong Hình [9.5b](#bookmark1036). Các phần có nền khác nhau thể hiện vùng của mỗi lớp dữ liệu tìm được bằng hồi quy softmax. Ta thấy rằng các đường ranh giới có dạng đường thẳng. Kết quả phân chia vùng cũng khá tốt, chỉ có một số ít điểm trong tập huấn luyện bi phân loại sai. Ta cũng thấy hồi quy softmax tốt hơn rất nhiều so với phương pháp kết hợp các bộ phân loại nhi phân.

MNIST với hồi quy softmax trong scikit-learn

Trong scikit-learn, hồi quy softmax được tích hợp trong class sklearn.linear\_model .LogisticRegression. Như sẽ thấy trong phần thảo luận, hồi quy logistic chính là hồi quy softmax cho bài toán phân loại nhi phân. Với bài toán phân loại đa lớp, thư viện này mặc đinh sử dụng kỹ thuật one-vs-rest. Đe sử dụng hồi quy softmax, ta thay đổi thuộc tính multi\_class = 'multinomial' và solver = 'lbfgs'. ở đây, 'lbfgs' là một phương pháp tối ưu rất mạnh cũng dựa trên gradient.

Quay lại với bài toán phân loại chữ số viết tay trong cơ sở dữ liệu MNIST. Đoạn code dưới đây thực hiện việc lấy ra 10000 điểm dữ liệu trong số 70000 điểm làm tập kiểm tra, còn lại là tập huấn luyện. Bộ phân loại được sử dụng là hồi quy softmax.

**import** numpy as np

**from** sklearn.datasets **import** fetch\_mldata

**from** sklearn.linear\_model **import** LogisticRegression

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

**from** sklearn.metrics **import** accuracy\_score

mnist = fetch\_mldata('MNIST original', data\_home='../../data/')

X = mnist.data

y = mnist.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=10000) model = LogisticRegression(C = 1e5,

solver = 'lbfgs', multi\_class = 'multinomial') *# C is inverse of lam* model.fit(X\_train, y\_train) y pred = model.predict(X\_test)

**print**("Accuracy %.2f %%" % (100\*accuracy\_score(y\_test, y pred.tolist())))

Kết quả:

Accuracy: 92.19 %

So với kết quả hơn 91.7% của one-vs-rest hồi quy logistic, kết quả của hồi quy softmax đã được cải thiện. Kết quả thấp này hoàn toàn có thể dự đoán được vì thực ra hồi quy softmax chỉ tạo ra các đường ranh giới tuyến tính. Kết quả tốt nhất của bài toán phân loại chữ số trong MNIST hiện nay vào khoảng hơn 99.7%, đạt được bằng một mạng neuron tích chập với rất nhiều tầng ẩn và tầng cuối cùng là một hồi quy softmax.

1. Thảo luận
2. Hồi quy logistic là trường hợp đặc biệt của hồi quy softmax

Khi *C* =2, hồi quy softmax và hồi quy logistic là hai mô hình giống nhau. Thật vậy, với C = 2, đầu ra của hàm softmax cho một đầu vào **x** là:

a = exp(xT**W**Ị) = *t;* (9 \_.

a1 ..... .\ 1 I */\'T Í-T.T* w ; a2 1 a1 (9.15)

exp(**x**T **W**1) + exp(**x**T **w**2) 1 + exp(**x**T (**w**2 — **W**1))

Từ đây ta thấy rằng, *a1* có dạng là một hàm sigmoid với vector trọng số có dạng **w** = — (**w**2 — **W**1). Khi C =2, bạn đọc cũng có thể thấy rằng hàm mất mát của hồi quy logistic và hồi quy softmax là như nhau. Hơn nữa, mặc dù có hai đầu ra, hồi quy softmax có thể biểu diễn bởi một đầu ra vì tổng của chúng bằng một.

Giống như hồi quy logistic, hồi quy softmax được sử dụng trong các bài toán phân loại. Các tên gọi này được giữ lại vì vấn đề lịch sử.

1. Ranh giới tạo bởi hồi quy softmax là các mặt tuyến tính

Thật vậy, dựa vào hàm softmax thì một điểm dữ liệu **x** được dự đoán là rơi vào class *j* nếu *aj > ak, = j* .Ta có thể chứng minh được rằng:

*aj > ak , Zj > zk ,* ***xTw****j >* ***xT*w**k , **x**T(**w**j — **w**k) > 0. (9.16)

Như vậy, một điểm thuộc lớp thứ j nếu và chỉ nếu **x**T (**w**j — **w**k) > 0, = j. Như vậy, mỗi lớp dữ liệu chiếm một vùng là giao của các nửa không gian. Nói cách khác, đường ranh giới giữa các lớp là các mặt tuyến tính.

1. Hồi quy softmax là một trong hai bộ phân loại phổ biến nhất

Hồi quy softmax là bộ phân loại phổ biến nhất được dùng hiện nay. Hồi quy softmax đặc biệt được sử dụng nhiều trong các mạng neuron sâu với rất nhiều tầng an. Những tầng phía trước có the được coi như một bộ trích chọn vector đặc trưng, tầng cuối cùng thường là một hồi quy softmax.

Máy vector hỗ trỢ

1. Giới thiệu

*Mấy vector hễ trợ* (support vector machine, SVM) là một trong những thuật toán phân loại phổ biến và hiệu quả. Y tưởng đứng sau SVM khá đơn giản, nhưng để giải bài toán tối ưu SVM, chúng ta cần kiến thức về tối ưu và đối ngẫu.

Trước khi đi vào phần ý tưởng chính của SVM, chúng ta cùng ôn lại kiến thức về hình học giải tích trong chương trình phổ thông.

1. Khoảng cách từ một điểm tới một siêu mặt phẳng

Trong không gian hai chiều, khoảng cách từ một điểm có toạ độ (x0, y0) tới đường thẳng có phương trình *w1x* + *w2y* + *b* = 0 được xác đinh bởi

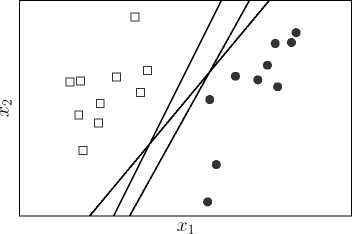
|wiXo + *w2yo* + *b\*

*\fwị* + w2

Trong không gian ba chiều, khoảng cách từ một điểm có toạ độ (x0,y0,z0) tới một mặt phẳng có phương trình w1x + *w2y* + *w3z* + b = 0 được xác đinh bởi

|wiXo + *w2yo* + W3Z0 + b\  
*ỵ/wị* + w2 + w2

Hơn nữa, nếu bỏ dấu tri tuyệt đối ở tử số, ta có thể xác đinh được điểm đó nằm về phía nào của đường thẳng hay mặt phẳng đang xét. Những điểm làm cho biểu thức trong dấu tri tuyệt đối mang dấu dương nằm về cùng một phía (tạm gọi là *phía dương*), những điểm làm cho giá tri này mang dấu âm nằm về phía còn lại (gọi là *phía âm*). Những điểm làm cho tử số bằng không sẽ nằm trên đường thẳng/mặt phẳng phân chia.

**Hình 10.1.** Hai lóp dữ liệu vuông và tròn là tách biệt tuyến tính. Có vô số đường thằng có the phân loại chính xác hai lóp dữ liệu này (xem thêm Bài 7).

Các công thức này có thể được tổng quát lên cho trường hợp không gian *d* chiều. Khoảng cách từ một điểm (vector) có toạ độ (xio,x2o,... ,Xdo) tói siêu phẳng *W1X1* + *W2X2* + • • • + *WdXd* + *b* = 0 được xác đinh bởi

|W1Xio + W2X20 + + *WdXdo* + *b\ \_* \**w**T**X**o + *b\*

*ỵ/w1* + W2 + + wd IIw II2

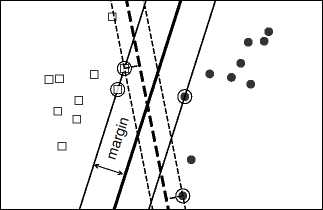
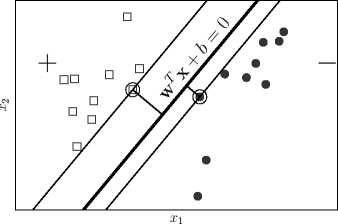
vói **X**o = [X10;X20; . . . ;Xdo]T; **w** = [W1;W2; . . . ;Wd]T.

1. Nhắc lại bài toán phân loại hai lớp dữ liệu

Xin nhắc lại bài toán phân loại đã đề cập trong Bài 7 (PLA). Giả sử có hai lóp dữ liệu được mô tả bởi các vector đặc trưng trong không gian nhiều chiều. Hơn nữa, hai lóp dữ liệu này là tách biệt tuyến tính, tức tồn tại một siêu phẳng phân chia chính xác hai lóp đó. Hãy tìm một siêu phẳng sao cho tất cả các điểm thuộc một lóp nằm về cùng một phía của siêu phẳng đó và ngược phía vói toàn bộ các điểm thuộc lóp còn lại. Chúng ta đã biết rằng, thuật toán PLA có thể thực hiện được việc này nhưng PLA có the cho vô số nghiệm như Hình [10.1](#bookmark1069).

Có một câu hỏi được đặt ra: Trong vô số các mặt phân chia đó, đâu là mặt tốt nhất? Trong ba đường thẳng minh họa trong Hình [10.1](#bookmark1069), có hai đường thẳng khá lệch về phía lóp hình tròn. Điều này có thể khiến nhiều điểm hình tròn chưa nhìn thấy bi phân loại lỗi thành điểm hình vuông. Liệu có cách nào tìm được đường phân chia sao cho đường này không lệch về một lóp không?

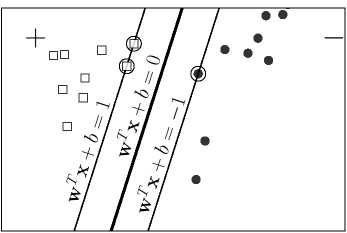
Để trả lời câu hỏi này, chúng ta cần tìm một tiêu chuẩn để đo sự *lệch* về mỗi lóp của đường phân chia. Gọi khoảng cách nhỏ nhất từ một điểm thuộc một lóp tói đường phân chia là *lề* (margin). Ta cần tìm một đường phân chia sao cho lề của hai lóp là như nhau đối vói đường phân chia đó. Hơn nữa, độ rộng của lề càng lón thì khả nang xảy ra phân loại lỗi càng thấp. Bài toán tối ưu trong SVM chính là bài toán đi tìm đường phân chia sao cho lề rộng nhất. Đây cũng là lý do SVM còn được gọi là *bộ phân loại lề lớn nhất* (maximum margin classifier). Nguồn gốc tên gọi *máy vector hễ trợ* sẽ sóm được làm sáng tỏ.



**(a)**

**(b)**

**Hình 10.2.** Ý tưởng của SVM. Le của một lóp được định nghĩa là khoảng cách từ các điem gan nhất của lóp đó tói mặt phân chia. Le của hai lóp phải bằng nhau và lón nhất có the.

**Hình 10.3.** Giả sử mặt phân chia có phương trình **w**T**x** + *b* = 0. Không mất tính tong quát, bằng cách nhân các hệ so **w** và b vói các hằng sè phù hợp, ta có the giả sử rằng điem gan nhất của lóp vuông tói mặt này thoả mãn **w**T**x**+b = 1. Khi đó, điếm gan nhất của lóp tròn thoả mãn **w**T**x** + b = —1.

1. Xây dựng bài toán tối ưu cho máy vector hỗ trỢ

Giả sử dữ liệu trong tập huấn luyện là các cặp (vector đặc trưng, nhãn): (**x**i ,y1), (**x**2,y2),..., (**x**N *,yN*) nhãn bằng +1 hoặc -1 và *N* là số điểm dữ liệu. Không mất tính tổng quát, giả sử các điểm vuông có nhãn là 1, các điểm tròn có nhãn là -1 và siêu phang ***wTx*** + b = 0 là mặt phân chia hai lớp (Hình [10.3)](#bookmark1075). Ngoài ra, lớp hình vuông nằm về phía dương, lớp hình tròn nằm về phía âm của mặt phân chia. Nếu xảy ra điều ngược lại, ta chỉ cần đổi dấu của **w** và b. Bài toán tối ưu trong SVM sẽ là bài toán đi tìm các tham số mô hình **w** và b.

Với cặp dữ liệu (**x**„, yn) bất kỳ, khoảng cách từ tới mặt phân chia là .

Điều này xảy ra ta đã giả sử *yn* cùng dấu với phía của **x**„. Từ đó suy ra *yn* cùng dấu với (**w**T***x****n* + b) và tử số luôn là một đại lượng không âm. Với mặt phân chia này, lề được tính là khoảng cách gần nhất từ một điểm (trong cả hai lớp, vì cuối cùng lề của hai lớp bằng nhau) tới mặt phân chia:

*. yn****(w****T* ***x*** + b)

lề

min ***-—77-***

n Ii**w|i**2

Bài toán tối ưu của SVM đi tìm **w** và *b* sao cho lề đạt giá tri lớn nhất:

(**w**, *b) =* argmax < min ***—77-*** > = argmax < -—— min *y****n****(****w xn*** + *b)*

w,b I n **||w||**2 I **w**,b I **||w||**2 n

J10.1) Nếu ta thay vector trọng số **w** bởi *k****w*** và b bởi *kb* trong đó *k* là một hằng số dương bất kỳ thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức lề không đổi. Vì vậy, ta có thể giả sử:

*ym(****w****T* ***x****m* + b) = 1

với những điểm nằm gần mặt phân chia nhất (được khoanh tròn trong Hình [10.3)](#bookmark1075).

Như vậy, với mọi *n* ta luôn có

*yn* (**w**T**X**n + b) *>* 1

Bài toán tối ưu [(10.1)](#bookmark1080) có thể được đưa về bài toán tối ưu ràng buộc có dạng

(**w**, b) = arg max -——

6 Wb **||w||**2 (10.2)

thoả mãn: yn(**w**T**x**n + b) > 1,8n = 1,2,..., *N*

Bằng một biến đổi đơn giản, ta có thể tiếp tục đưa bài toán này về dạng

(w,b) = argmin 1 Iiwii2

W;b 2 (10.3)

thoả mãn: 1 — yn(**w**T**x**n + b) < 0,= 1, 2,..., N

ở đây, ta đã lấy nghịch đảo hàm mục tiêu, bình phương nó để được một hàm khả vi, và nhân với 2 để biểu thức đạo hàm đẹp hơn.

Trong bài toán [(10.3)](#bookmark1081), hàm mục tiêu là một chuẩn - có dạng toàn phương. Các hàm bất phương trình ràng buộc là affine. Vậy bài toán [(10.3)](#bookmark1081) là một bài toán quy hoạch toàn phương. Hơn nữa, hàm mục tiêu là lồi chặt vì **||w||**2 = **wTIw** và **I** là ma trận đơn vị - một ma trận xác định dương. Từ đây có the suy ra nghiệm của SVM là duy nhất.

Tới đây, bài toán này có thể giải được bằng các công cụ hỗ trợ giải quy hoạch toàn phương, ví dụ CVXOPT. Tuy nhiên, việc giải bài toán này trở nên phức tạp khi số chiều *d* của không gian dữ liệu và số điểm dữ liệu N lớn. Thay vào đó, người ta thường giải bài toán đối ngẫu của bài toán này. Thứ nhất, bài toán đối ngẫu có những tính chất thú vị khiến nó được giải một cách hiệu quả hơn. Thứ hai, trong quá trình xây dựng bài toán đối ngẫu, người ta thấy rằng SVM có thể được áp dụng cho những bài toán mà dữ liệu không nhất thiết tách biệt tuyến tính, như chúng ta sẽ thấy sau này.

***Xác định lớp cho một điểm dữ liệu mới***

Sau khi đã tìm được mặt phân chia **w**T**x** + *b* = 0, nhãn của một điểm bất kỳ sẽ được xác đinh đơn giản bằng

class(**x**) = sgn(**w**T **x** + b)

1. Bài toán đối ngẫu của máy vector hỗ trỢ

Bài toán tối ưu [(10.3)](#bookmark1081) là một bài toán lồi. Chúng ta biết rằng nếu một bài toán lồi thoả mãn tiêu chuẩn Slater thì đối ngẫu mạnh xảy ra. Ngoài ra, nếu đối ngẫu mạnh thoả mãn thì nghiệm của bài toán chính là nghiệm của hệ điều kiện KKT.

1. Kiểm tra tiêu chuẩn Slater

Trong bước này, chúng ta sẽ chứng minh bài toán tối ưu [(10.3)](#bookmark1081) thoả mãn điều kiện Slater. Điều kiện Slater nói rằng, nếu tồn tại **w**, b thoả mãn

1 - *yn(****wT x*** + b) < 0, = 1,2,...,N

thì đối ngẫu mạnh cũng thoả mãn. Việc kiểm tra điều kiện này không quá phức tạp. Vì luôn có một siêu phẳng phân chia hai lớp dữ liệu tách biệt tuyến tính nên tập khả thi của bài toán tối ưu [(10.3)](#bookmark1081) khác rỗng. Điều này cũng có nghĩa là luôn tồn tại cặp (**w**0,b0) sao cho:

1 - *yn* (**w**T ***x****n* + bo) < 0, = 1, 2,...,N (10.4)

, 2 - y„(2**w**T ***x*** + 2bo) < 0, = 1, 2,...,N (10.5)

Vậy chỉ cần chọn **w**1 = 2**w**0 và *b1 =* 2b0, ta sẽ có:

1 - *yn(****w****T* ***x****n* + bi) <-1 < 0, 8n = 1, 2,...,N

Điều này chỉ ra rằng (**w**1, b1) là một điểm khả thi chặt. Từ đó suy ra điều kiện Slater thoả mãn.

1. Hàm Lagrange của bài toán tối ưu

Hàm Lagrange của bài toán [(10.3)](#bookmark1081) là

1 \_ X

L(**w**,b, **A**) = 2**||w||**2 + **22** An(1 - y™(**wTx**„ + b)) (10.6)

*n=1*

với **A** = [A1, A2,..., A V]t và An *>* 0, =1,2,..., N.

1. Hàm đối ngẫu Lagrange

Theo đinh nghĩa, hàm đối ngẫu Lagrange là

g(**A**) = min L(**w**, *b,* **A**)

**W**;b

với **A** N 0. Việc tìm giá tri nhỏ nhất của hàm này theo **w** và b có thể đựợc thực hiện bằng cách giải hệ phương trình đạo hàm của L(**w**, b, **A**) theo **w** và b bằng 0:

N N

VwL(**w**, b, **A**) = **w** ^2 *Xn'.hXn =* **0** ) **w ^2** (10.7)

*n=1 n=1*

N

*rbC(****w****,b,* **A**) = 22 *^nVn* = 0 (10.8)

*n=1*

Thay [(10.7)](#bookmark1100) và [(10.8)](#bookmark1100) vào [(10.6)](#bookmark1095) ta thu được g(**A**)[[[3]](#footnote-4)](#bookmark0):

N 1 N N

g(A) — 2^3 ^n 2 2 y 2 y*^n^mynyni****^****n xm* (10.9)

*n=1 n=1 m=1*

Hàm *g(****A****)* trong [(10.9)](#bookmark1101) là hàm số quan trọng nhất của SVM.

Ta có thể viết lại *g(****A****)* dưới dạn[g[[4]](#footnote-5)](#bookmark1)

*g(****A****) =* -1 *At* ***V****t****V****A* + ***1****T* **A**. (10.10)

với **V** = [y1**X**1,y2**X**2,... ,yN**xn]** và **1** = [1,1,..., 1]T.

Nếu đặt **K** = **V**T**V** thì **K** là một ma trận nửa xác đinh dương. Thật vậy, với mọi vector **A** ta có **A**T**KA** = ***A****T****V****T****VA*** *=* II**VA**k2 > 0. Vậy *g(****A****) = -1 At****K****a* + **1**T**A** là một hàm lồi.

1. Bài toán đối ngẫu Lagrange

Từ đó, kết hợp hàm đối ngẫu Lagrange và các điều kiện ràng buộc của **A**, ta sẽ thu được bài toán đối ngẫu Lagrange của bài toán [(10.3)](#bookmark1081):

**A** = arg max *g(****A****)*

thoả mãn: **A** 0 z„ „ „ „ X

V í10.11)

22*^nVn* 0

n=1

Ràng buộc thứ hai được lấy từ [(10.8)](#bookmark1100). Đây là một bài toán lồi vì ta đang đi tìm giá tri lớn nhất của một hàm mục tiêu lõm trên một đa diện. Hơn nữa, đây là một bài toán quy hoạch toàn phương và cũng có thể được giải bằng các thư viện như CVXOPT.

Biến tối ưu trong bài toán tối ngẫu là **A**, là một vector *N* chiều tương ứng với số điểm dữ liệu. Trong khi đó, số tham số phải tìm trong bài toán tối ưu chính [(10.3)](#bookmark1081) là *d* +1, chính là tổng số chiều của **w** và *b,* tức số chiều của mỗi điểm dữ liệu cộng một. Trong rất nhiều trường hợp, số điểm dữ liệu trong tập huấn luyện lớn hơn số chiều dữ liệu. Nếu giải trực tiếp bằng các công cụ giải quy hoạch toàn phương, bài toán đối ngẫu có thể phức tạp hơn bài toán gốc. Tuy nhiên, điểm hấp dẫn của bài toán đối ngẫu này đến từ cấu trúc đặc biệt của hệ điều kiện KKT.

1. Điều kiện KKT

Quay trở lại bài toán, vì đây là một bài toán tối ưu lồi và đối ngẫu mạnh xảy ra, nghiệm của bài toán thoả mãn hệ điều kiện KKT sau đây với biến số **w**, b và **A**:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 - y„(**w**T**X**n + b) < 0, =1,2,.. | | .,N | (10.12) |
|  | An *>* 0, =1,2,.. | .,N | (10.13) |
| An(1 - y„(**w** | T**X**n + b)) = 0, =1,2,.. | .,N | (10.14) |
|  | *N* |  |  |
|  | *w —* Anynxn |  | (10.15) |
|  | XN |  |  |
|  | Anyn 0  *n=1* |  | (10.16) |

Trong những điều kiện trên, điều kiện lỏng lẻo bù trừ [(10.14)](#bookmark1111) là thú vi nhất. Từ đó ta có thể suy ra *An* = 0 hoặc 1 — *yn(****w****T****x****n* + b) = 0 với *n* bất kỳ. Trường hợp thứ hai tương đương với

**w**T **X**n + b = *yn.* (10.17)

Những điểm thoả mãn [(10.17)](#bookmark1112) chính là những điểm nằm gần mặt phân chia nhất (những điểm được khoanh tròn trong Hình [10.3)](#bookmark1075). Hai đường thẳng **w**T**x**n+b = ±1 *tựa* lên các vector thoả mãn [(10.17)](#bookmark1112). Những vector thoả mãn [(10.17)](#bookmark1112) được gọi là *vector hễ trợ* (support vector). Tên gọi *máy vector hễ trợ* xuất phát từ đây.

Số lượng điểm thoả mãn [(10.17)](#bookmark1112) thường chiếm một lượng nhỏ trong số *N* điểm dữ liệu huấn luyện. Chỉ cần dựa trên những vector hỗ trợ này, chúng ta hoàn toàn có thể xác đinh được mặt phân cách cần tìm. Nói cách khác, hầu hết các *An* bằng không, tức A là một vector thưa. Máy vector hỗ trợ vì vậy cũng được coi là một *mô hình thưa* (sparse model). Các mô hình thưa thường có cách giải quyết hiệu quả hơn các mô hình tương tự với nghiệm *dày đặc* (dense, hầu hết các phần tử khác không). Đây là lý do thứ hai của việc bài toán đối ngẫu SVM được quan tâm nhiều hơn là bài toán chính.

Tiếp tục phân tích, với những bài toán với số điểm dữ liệu *N* nhỏ, ta có thể giải hệ điều kiện KKT phía trên bằng cách xét các trường hợp *Xn* = 0 hoặc *Xn* = 0. Tổng số trường hợp phải xét là *2N*. Thông thường, *N >* 50 và *2N* là một con số rất lớn. Việc thử 2N trường hợp là bất khả thi. Phương pháp thường được dùng để giải hệ này là *sequential minimal optimization* (SMO). Trong phạm vi cuốn sách, chúng ta sẽ không đi sâu tiếp vào việc giải hệ KKT như thế nào.

Trong phần tiếp theo chúng ta sẽ giải bài toán tối ưu [(10.11)](#bookmark1106) qua một ví dụ nhỏ bằng CVXOPT, và trực tiếp sử dụng thư viện sklearn đe huấn luyện mô hình SVM. Sau khi tìm được **A** từ bài toán [(10.11)](#bookmark1106), ta có thể suy ra **w** dựa vào [(10.15)](#bookmark1111) và *b* dựa vào [(10.14)](#bookmark1111) và [(10.16)](#bookmark1111). Rõ ràng ta chỉ cần quan tâm tới *Xn =* 0.

Đặt S = *{n* : *Xn* = 0} và *NS* là số phần tử của S. Theo [(10.15)](#bookmark1111), **w** được tính bằng

**w** = *Xmym****x****m.* (10.18)

*m2S*

Với mỗi *n 2 S*, ta có

1 = *yn(****wTX****n* + b) , b = *y****n - WTX****n.*

Mặc dù hoàn toàn có thể suy ra b từ một cặp (**x**n, yn) nếu đã biết **w**, một phiên bản tính b khác thường được sử dụng và có phần ổn đinh hơn trong tính toán là trung bình cộn[g[[5]](#footnote-6)](#bookmark2) của các b tính được theo mỗi *n* 2 S

b = *N^Ĩ2(yn ~ wT xn) =* N *Ĩ2* ( yn *~Ĩ2* (10.19)

S *n2S S n2S \ m2S /*

Để xác đinh một điểm **x** thuộc vào lớp nào, ta cần tìm dấu của biểu thức

wT x + b *=^2 Xm ymxLx* + N. ( yn *Xmymxỉnxn} .*

*m2S S n2S \ m2S /*

Biểu thức này phụ thuộc vào cách tính tích vô hướng giữa **x** và từng ***x****m* 2 S.

1. Lập trình tìm nghiệm cho máy vector hỗ trỢ

Trong mục này, ta sẽ tìm nghiệm của SVM bằng hai cách khác nhau. Cách thứ nhất dựa trên bài toán [(10.11)](#bookmark1106) với nghiệm tìm được theo các công thức [(10.19)](#bookmark1114) và [(10.18)](#bookmark1113). Cách làm này giúp chứng minh tính đúng đắn của các công thức đã xây dựng. Cách thứ hai sử dụng trực tiếp thư viện sklearn, giúp sinh viên làm quen với việc áp dụng SVM vào dữ liệu thực tế.

1. Tìm nghiệm theo công thức

Trước tiên ta khai báo các thư viện và tạo dữ liệu giả (dữ liệu này được sử dụng trong các hình từ đầu bài. Ta thấy rằng hai lớp dữ liệu tách biệt tuyến tính):

**from** future **import** print\_function

**import** numpy as np

np.random.seed(22)

* *simulated samples*

means = [[2, 2], [4, 2]]

cov = [[.3, .2], [.2, .3]]

N = 10

X0 = np.random.multivariate\_normal(means[0], cov, N) *# blue class data*

X1 = np.random.multivariate\_normal(means[1], cov, N) *# red class data*

X = np.concatenate((X0, X1), axis =0) *# all data*

y = np.concatenate((np.ones(N), -np.ones(N)), axis =0) *# label*

* *solving the dual problem (variable: lambda)*

**from** cvxopt **import** matrix, solvers

V = np.concatenate((X0, -X1), axis =0) *# V in the book*

Q = matrix(V.dot(V.T))

p = matrix(-np.ones((2\*N, 1))) *# objective function 1/2 lambda^T\*Q\*lambda -*

*1'T\*lambda*

* *build A, b, G, h*

G = matrix(-np.eye(2\*N))

h = matrix(np.zeros((2\*N, 1)))

A = matrix(y.reshape(1, -1))

b = matrix(np.zeros((1, 1)))

solvers.options['show\_progress'] = False

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sol  l= | | = solvers.qp(Q, p, np.array(sol['x']) | G,  *#* | h, A, b)  *solution lambda* | |
| *#* | *calculate w and b* | |  |  |  |
| w | = | Xbar.T.dot(l) |  |  |  |
| S | = | np.where(l > 1e-8) | [0] | *# support set,* | *1e-8 to avoid small value of l* |
| b | = | np.mean(y[S].reshape( | | -1, 1) - X[S,:]. | dot(w)) |

**print**('Number of suport vectors = ', S.size) **print**('w = ', w.T)

**print** ('b = ', b)

Kết quả:

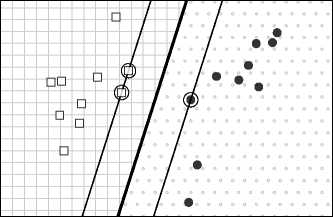
Number of suport vectors = 3

w = [[-2.00984382 0.64068336]]

b = 4.66856068329

Như vậy trong số 20 điểm dữ liệu của cả hai lớp, chỉ có ba điểm đóng vai trò vector hỗ trợ. Ba điểm này giúp tính w và b. Đường thẳng phân chia tìm được có màu đen đậm và được minh hoạ trong Hình [10.4](#bookmark1127). Hai đường đen mảnh thể hiện đường thẳng tựa lên các vector hỗ trợ được khoanh tròn.

Hình vẽ và mã nguồn trong bài có thể được tìm thấy tại<https://goo.gl/VKBgVG>.

**Hình 10.4.** Minh hoạ nghiệm tìm được bởi SVM. Tat cả các điếm nằm trong vòng có nền kẻ ô sẽ được phân vào còng lóp vói các điem vuông. Diều tương tự xảy ra vói các điếm tròn nằm trên nen d§u ch§m.

1. Tìm nghiệm theo thư viện

Chúng ta sẽ sử dụng [sklearn.svm.SV](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)[C[[6]](#footnote-7)](#bookmark3). Sinh viên có thể tham khảo thêm thư viện [libsvm](https://www.csie.ntu.edu.tw/~cjlin/libsvm/) được viết trên ngôn ngữ C, có API cho Python và Matlab:

*# solution by sklearn*

**from** sklearn.svm **import** SVC

model = SVC(kernel = 'linear', C = 1e5) *# just a big number* model.fit(X, y)

w = model.coef\_

b = model.intercept\_

**print**('w = ', w)

**print**('b = ', b)

Kết quả:

w= [[-2.00971102 0.64194082]]

b = [ 4.66595309]

Kết quả này thống nhất với kết quả tìm được ở mục trước. Có rất nhiều tuỳ chọn cho SVC, trong đó có thuộc tính kernel, chúng ta sẽ tìm hiểu sau.

1. Tóm tắt

• Nếu hai lớp dữ liệu tách biệt tuyến tính, có vô số các siêu phẳng phân chia hai lớp đó. Khoảng cách gần nhất từ một điểm dữ liệu tới siêu phẳng này được gọi là lề.

* SVM là bài toán đi tìm mặt phân cách sao cho lề của hai lớp bằng nhau và lớn nhất, đồng nghĩa với việc các điểm dữ liệu có một khoảng cách an toàn tới mặt phân chia.
* Bài toán tối ưu trong SVM là một bài toán quy hoạch toàn phương với hàm mục tiêu lồi chặt. Vì vậy, cực tiểu địa phương cũng là cực tiểu toàn cục của bài toán.
* Mặc dù có thể trực tiếp giải SVM qua bài toán chính, người ta thường giải bài toán đối ngẫu. Bài toán đối ngẫu cũng là một bài toán quy hoạch toàn phương nhưng nghiệm là các vector thưa nên có những phương pháp giải hiệu quả hơn. Ngoài ra, bài toán đối ngẫu có những tính chất thú vị sẽ được thảo luận sau.

1. BÀI TẬP CUỐI CHƯƠNG

**Bài 1.** Xây dựng một bộ phân loại cho tập dữ liệu MNIST có độ chính xác trên 97% trên tập kiểm tra.

**Bài 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.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 trên tập dữ liệu đã mở rộng này và đo độ chính xác của nó trên tập kiểm tra. (Làm quen với kỹ thuật tang số lượng mẫu trong tập huấn luyện (data augmentation) hay mở rộng tập huấn luyện (training set expansion)).

**Bài 3.** Hãy làm việc với tập dữ liệu Titanic trên Kaggle.

**Bài 4.** Hãy xây dựng một bộ phân loại thư rác.

**Bài 5.** Nên sử dụng thuật toán nào để huấn luyện mô hình Hồi quy Tuyến tính, nếu tập huấn luyện có hàng triệu đặc trưng?

**Bài 6.** 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?

**Bài 7.** Thuật toán Hạ Gradient có thể bị kẹt trong điểm cực tiểu địa phương khi huấn luyện mô hình Hồi quy Logistic hay không?

**Bài 8.** 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?

**Bài 9.** Giả sử ta sử dụng thuật toán Hạ Gradient theo Batch và theo dõi lỗi kiểm đinh trong từng epoch, và phát hiện ra lỗi kiểm đinh tang lên liên tục. Vấn đề này có thể là gì? Ta sẽ giải quyết vấn đề này bằng cách nào?

**Bài 10.** Liệu ta có nên dừng thuật toán Hạ Gradient theo Mini-batch ngay khi lỗi kiểm đinh tang lên?

**Bài 11.** 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?

**Bài 12.** Giả sử ta đang sử dụng Hồi quy Đa thức. Khi theo dõi đồ thi quá trình học, ta nhận ra có một khoảng cách lớn giữa lỗi huấn luyện và lỗi kiểm đinh. Vấn đề gì đang xảy ra? Hãy nêu ra ba giải pháp cho vấn đề này.

**Bài 13.** Giả sử ta đang sử dụng Hồi quy Ridge. Ta nhận ra rằng lỗi huấn luyện và lỗi kiểm đinh 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 tang hay giảm giá tri siêu tham số điều chuẩn *a?*

**Bài 14.** Tại sao ta nên sử dụng:

1. Hồi quy Ridge thay vì Hồi quy Tuyến tính (không sử dụng điều chuẩn)?
2. Hồi quy Lasso thay vì Ridge?
3. Elastic Net thay vì Lasso?

**Bài 15.** 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?

**Bài 16.** 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.

**Bài 17.** Y tưởng can bản của Máy Vector Hỗ trợ là gì?

**Bài 18.** Vector hỗ trợ là gì?

**Bài 19.** Tại sao việc co giãn giá tri đầu vào lại quan trọng khi sử dụng SVM?

**Bài 20.** Bộ phân loại SVM có thể trả về điểm số tin cậy và xác suất khi phân loại một mẫu không?

**Bài 21.** 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 tram đặc trưng?

**Bài 22.** 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. Ta nên tang hay giảm các tham số 7 (gamma) và *C*?

**Bài 23.** Nên thiết lập các tham số quy hoạch toàn phương (H, f, A, và b) như thế nào để giải bài toán cho bộ phân loại SVM biên mềm với bộ giải quy hoạch toàn phương có sẵn?

**Bài 24.** 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.

**Bài 25.** Hãy huấn luyện bộ phân loại SVM trên tập dữ liệu MNIST. Do SVM là bộ phân loại nhi phân, sẽ cần sử dụng phương pháp một-còn lại để phân loại 10 chữ số. Có the sẽ cần tinh chỉnh siêu tham số với các tập kiểm đinh nhỏ đe tang tốc quá trình này. Độ chính xác cao nhất có thể đạt được là bao nhiêu?

**Bài 26.** Hãy huấn luyện bộ hồi quy SVM trên tập dữ liệu nhà ở California.

Chương III

CÁC THUẬT TOÁN HỌC KHÔNG GIÁM

SÁT

Chương 3 trang bị kiến thức về học không giám sát. Thuật toán phân cụm K mean và thuật toán giảm chiều PCA được lựa chọn giới thiệu đến các em sinh viên. Thời lượng học tập 2 buổi, mỗi buổi 2 tiết.

Phân cụm K-means

1. Giới thiệu

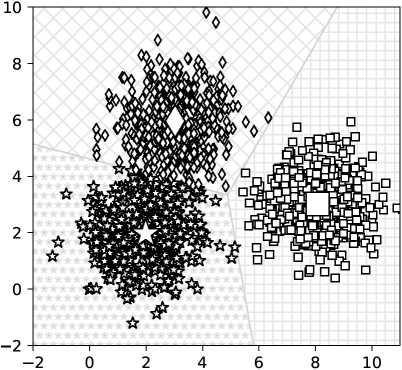
Trong các bài trước, chúng ta đã làm quen các thuật toán học có giám sát. Trong bài này, một thuật toán đơn giản của học không giám sát sẽ được trình bày. Thuật toán này có tên là *phân cụm K-means (K-means clustering*).

Trong phân cụm K-means, ta không biết nhãn của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm khác nhau sao cho dữ liệu trong cùng một cụm có những tính chất giống nhau.

**Ví dụ:** Một công ty muốn tạo ra một chính sách ưu đãi cho những nhóm khách hàng khác nhau dựa trên sự tương tác giữa mỗi khách hàng với công ty đó (số nam là khách hàng, số tiền khách hàng đã chi trả cho công ty, độ tuổi, giới tính, thành phố, nghề nghiệp,...). Giả sử công ty có dữ liệu của khách hàng nhưng phân cụm. Phân cụm K-means là một thuật toán có the giúp thực hiện công việc này. Sau khi phân cụm, nhân viên công ty có thể quyết đinh mỗi nhóm tương ứng với nhóm khách hàng nào. Phần việc cuối cùng này cần sự can thiệp của con người, nhưng lượng công việc đã được rút gọn đi đáng kể.

Một nhóm/cụm có thể được đinh nghĩa là tập hợp các điểm có vector đặc trưng gần nhau. Việc tính toán khoảng cách có thể phụ thuộc vào từng loại dữ liệu, trong đó khoảng cách Euclid được sử dụng phổ biến nhất. Trong bài này, các tính toán được thực hiện dựa trên khoảng cách Euclid. Tuy nhiên, quy trình thực hiện thuật toán có thể được áp dụng cho các loại khoảng cách khác.

Hình [11.1](#bookmark1178) là một ví dụ về dữ liệu được phân vào ba cụm. Giả sử mỗi cụm có một điểm đại diện được gọi là *tâm cụm,* được minh hoạ bởi các điểm màu trắng lớn. Mỗi điểm thuộc vào cụm có tâm gần nó nhất. Tới đây, chúng ta có một bài toán thú vi: Trên vùng biển hình chữ nhật có ba đảo hình thoi, hình vuông và sao

**Hình 11.1.** Ví dụ vói ba cụm dữ liệu trong không gian hai chiều.

nam cánh lớn màu trắng như Hình [11.1](#bookmark1178). Một điểm trên biển được gọi là thuộc lãnh hải của một đảo nếu nó nằm gần đảo này hơn so với hai đảo còn lại. Hãy xác đinh ranh giới lãnh hải giữa các đảo.

Cũng trên Hình [11.1](#bookmark1178), các vùng với nền khác nhau biểu thi lãnh hải của mỗi đảo. Có thể thấy rằng đường phân đinh giữa các lãnh hải có dạng đường thẳng. Chính xác hơn, chúng là đường trung trực của các cặp đảo gần nhau. Vì vậy, lãnh hải của một đảo sẽ là một hình đa giác. Cách phân chia dựa trên khoảng cách tới điểm gần nhất này trong toán học được gọi là Voronoi diagram. Trong không gian ba chiều, lấy ví dụ là các hành tinh, *lãnh không* của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, chúng ta sẽ có những *siêu đa diện.*

1. Phân tích toán học

Mục đích cuối cùng của thuật toán phân cụm K-means là từ dữ liệu đầu vào và số lượng cụm cần tìm, hãy xác đinh tâm mỗi cụm và phân các điểm dữ liệu vào cụm tương ứng. Giả sử thêm rằng mỗi điểm dữ liệu chỉ thuộc đúng một cụm.

Giả sử *N* điểm dữ liệu trong tập huấn luyện được ghép lại thành ma trận **X** = [**x**1, **x**2,..., **x**N] 2 RdxN và K < N là số cụm được xác đinh trước. Ta cần tìm các tâm cụm **m**1, **m**2,... , **m**K 2 Rdx1 và nhãn của mỗi điểm dữ liệu. ở đây, mỗi cụm được đại diên bởi một nhãn, thường là một số tự nhiên từ 1 đến K. Nhắc lại rằng các điểm dữ liệu trong bài toán phân cụm K-means ban đầu không có nhãn cụ the.

Với mỗi điểm dữ liệu ***X***, ta cần tìm nhãn *yi = k* của nó, ở đây *k* 2 {1,2,... , Kg. Nhãn của một điểm cũng thường được biểu diễn dưới dạng một vector hàng K phần tử ***y****i* 2 R1xK , trong đó tất cả các phần tử của ***y****i* bằng 0 trừ phần tử ở vi trí thứ k bằng 1. Cách biểu diễn này còn được gọi là mã hoá *one-hot.* Cụ thể, *yij =* 0, 8j = *k,yik =* 1. Khi chồng các vector ***y****i* lên nhau, ta được một ma trận nhãn **Y** 2 RN xK. Nhắc lại rằng *yij* là phần tử hàng thứ *i,* cột thứ *j* của ma trận **Y**, và cũng là phần tử thứ j của vector **y**i. Ví dụ, nếu một điểm dữ liệu có vector nhãn là [1; 0; 0;... ; 0] thì nó thuộc vào cụm thứ nhất, là [0,1,0,... , 0] thì nó thuộc vào cụm thứ hai,... Điều kiện của **y**i có thể viết dưới dạng toán học:

K

*yij 2* {0; 1g; *8i;j*; **52**yij = 1; (11-1)

j=i

1. H àm mất mát và b ài toán tối ưu

Gọi **111;,.** 2 là tâm của cụm thứ k. Giả sử một điểm dữ liệu **xi** được phân vào cụm k. Vector sai số nếu thay **x**i bằng **m**k là (**x**i — **m**k). Ta muốn vector sai số này gần với vector không, tức **x**i gần với **m**k. Việc này có thể được thực hiện thông qua việc tối thiểu bình phưìng khoảng cách Euclid **||x**i — **m**k 112. Hìn nữa, vì **x**i được phân vào cụm *k* nên yik = *1,yij =* 0; 8j = *k.* Khi đó, biểu thức khoảng cách Euclid có thể được viết lại thành

K

llxi - **111;,.** 112 = yik\xi - **111;,.** 112 = **22** yij llxi - mj Il2 (n-2)

j=i

Như vậy, sai số trung bình cho toàn bộ dữ liệu sẽ là:

1 N K

L(Y; M) = *N^^yij* ||xi - mj ||2 (n.3)

i=i j=i

Trong đó **M** = [**m**1; **m2**;...; **m**K] 2 RdxK là ma trận tạo bởi K tâm cụm. Hàm mất mát trong bài toán phân cụm K-means là L(**Y**; **M**) với ràng buộc như được nêu trong [(11.1)](#bookmark1184). Đe tìm các tâm cụm và cụm tưìng ứng của mỗi điểm dữ liệu, ta cần giải bài toán tối ưu có ràng buộc

1 N K

Y; M = argmin -J- **52 V** yij ||xi - mj II2

**Y**;**M** N

j=1 K (11.4)

thoả mãn: yij 2 {0; 1g; 8i; j; **52** yij = 1; 8i

j=i

1. Thuật toán tối ưu h àm mất mát

Bài toán [(11.4)](#bookmark1189) là một bài toán khó tìm điểm tối ưu vì có thêm các điều kiện ràng buộc. *Bài toán này thuộc loại quy hoạch nguyên - là loại rất khó tìm nghiệm tối ưu toàn cục.* Tuy nhiên, trong một số trường hợp chúng ta vẫn có phưìng pháp để tìm nghiệm gần đúng. Một kỹ thuật đìn giản và phổ biến để giải bài toán [(11.4)](#bookmark1189) là xen kẽ giải **Y** và **M** khi biến còn lại được cố đinh cho tới khi hàm mất mát hội tụ. Chúng ta sẽ lần lượt giải quyết hai bài toán sau.

***Cố định* M, *tìm* Y**

*Giả sử đã tìm được các tăm cụm, hãy tìm các vector nhãn để hàm mất mát đạt giá trị nhỏ nhất.*

Khi các tâm cụm là cố đinh, bài toán tìm vector nhãn cho toàn bộ dữ liệu có thể được chia nhỏ thành bài toán tìm vector nhãn cho từng điểm dữ liệu **x**i như sau:

1 A

yi = argmin yij llxi - mj 112

yi *j=i*

(11.5)

K

thoảmãn: *yij 2* {0,1g, Vi,J; **52***yij* = 1, Vi

j=i

Vì chỉ có một phần tử của vector nhãn **y**j bằng 1 nên bài toán [(11.5)](#bookmark1194) chính là bài toán đi tìm tâm cụm gần điểm **x** nhất:

*j =* argmin **||xi - mj ||2.**

(11.6)

j

Vì **||x**j — **m**j 112 là bình phương khoảng cách Euclid từ điểm ***X****ị* tới tâm cụm **m**j, ta có thể kết luận rằng *mỗi điểm* ***X****i thuộc vào cụm có tăm gần nó nhất.* Từ đó có thể suy ra vector nhãn của từng điểm dữ liệu.

**Cố định Y, tìm M**

*Giả sử đã biết cụm của từng điểm, hãy tìm các tăm cụm mới để hàm mất mát đạt giá trị nhỏ nhất.*

Khi vector nhãn cho từng điểm dữ liệu đã được xác đinh, bài toán tìm tâm cụm được rút gọn thành

mj m-gnmi **V** yij||xi - mj||2\_ (n.7)

***m*** *N i=i*

Đe ý rằng hàm mục tiêu là một hàm liên tục và có đạo hàm xác đinh tại mọi điểm **m**j. Vì vậy, ta có thể tìm nghiệm bằng phương pháp giải phương trình đạo hàm bằng không. Đặt *l(****m****j*) là hàm mục tiêu bên trong dấu argmin của [(11.7)](#bookmark1195), ta cần giải phương trình sau đây:

2 A z

r**m** l(**m**j ) = *N^yij* (**m**j - *xi)* = **0** (11-8)

i=i

JV JV

, **m**j **52** yij *= Ĩ2 yij xi, mj* = ~~V n'~~*~~yji~~* (11-9)

i=i i=i i=i yij

Để ý rằng mẫu số chính là tổng số điểm dữ liệu trong cụm *j*, tử số là tổng các điểm dữ liệu trong cụm j. Nói cách khác, **m**j là trung bình cộng của các điểm trong cụm *j* .

1. Tóm tắt thuật toán

Tới đây, ta có thể tóm tắt thuật toán K-means cụming như sau.

Thuật toán 11.1: phân cụm K-means

***Đầu vào:*** *Ma trận dữ liệu* **X** 2 RdxN *và số lượng cụm cần tìm K < N*. ***Đầu ra:*** *Ma trận tâm cụm* **M** 2 RdxK *và ma trận nhãn* **Y** 2 xK.

1. *Chọn K điểm bất kỳ trong tập huấn luyện làm các tâm cụm ban đầu.*
2. *Phân mỗi điểm dữ liệu vào cụm có tâm gần nó nhất.*
3. *Nếu việc phân cụm dữ liệu vào từng cụm ở bước 2 không thay đoi so với vòng lặp trước nó thì dừng thuật toán.*
4. *Cập nhật tâm cụm bằng cách lấy trung bình cộng của các điểm đã được gán vào cụm đó sau bước 2.*
5. *Quay lại bước 2.*

Thuật toán này sẽ hội tụ sau một số hữu hạn vòng lặp. Thật vậy, dãy số biểu diễn giá tri của hàm mất mát sau mỗi bước là một đại lượng không tang và bi chặn dưới. Điều này chỉ ra rằng dãy số này phải hội tụ. Đe ý thêm nữa, số lượng cách phân cụm cho toàn bộ dữ liệu là hữu hạn (khi số cụm K là cố đinh) nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

Nếu tồn tại một cụm không chứa điểm nào, mẫu số trong [(11.8)](#bookmark1196) sẽ bằng không, và phép chia sẽ không thực hiện được. Vì vậy, K điểm bất kỳ trong tập huấn luyện được chọn làm các tâm cụm ban đầu ở bước 1 để đảm bảo mỗi cụm có ít nhất một điểm. Trong quá trình huấn luyện, nếu tồn tại một cụm không chứa điểm nào, có hai cách giải quyết. Cách thứ nhất là bỏ cụm đó và giảm K đi một. Cách thứ hai là thay tâm của cụm đó bằng một điểm bất kỳ trong tập huấn luyện, chẳng hạn như điểm xa tâm cụm hiện tại của nó nhất.

1. Ví dụ trên Python
2. Giới thiệu bài toán

Chúng ta sẽ làm một ví dụ đơn giản. Trước hết, ta tạo tâm cụm và dữ liệu cho từng cụm bằng cách lấy mẫu theo phân phối chuẩn có kỳ vọng là tâm của cụm đó và ma trận hiệp phương sai là ma trận đơn vi. ở đây, hàm cdist trong scipy. spatial.distance được dùng để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả.

Dữ liệu được tạo bằng cách lấy ngẫu nhiên 500 điểm cho mỗi cụm theo phân phối chuẩn có kỳ vọng lần lượt là (2, 2), (8, 3) và (3, 6); ma trận hiệp phưìng sai giống nhau và là ma trận đìn vi.

**from** future **import** print\_function

**import** numpy as np

**import** matplotlib.pyplot as plt

**from** scipy.spatial.distance **import** cdist

**import** random

np.random.seed(18)

means = [[2, 2], [8, 3], [3, 6]]

cov = [[1, 0], [0, 1]]

N = 500

X0 = np.random.multivariate\_normal(means[0], cov, N)

X1 = np.random.multivariate\_normal(means[1], cov, N)

X2 = np.random.multivariate\_normal(means[2], cov, N)

X = np.concatenate((X0, X1, X2), axis = 0)

K = 3 *# 3 clusters*

original\_label = np.asarray([0]\*N + [1]\*N + [2]\*N).T

1. Các hàm số cần thiết cho phân cụm *K*-means

Trước khi viết thuật toán chính phân cụm K-means, ta cần một số hàm phụ trợ:

1. kmeans\_init\_centroids khởi tạo các tâm cụm.
2. kmeans\_asign\_labels tìm nhãn mới cho các điểm khi biết các tâm cụm.
3. kmeans\_update\_centroids cập nhật các tâm cụm khi biết nhãn của từng điểm.
4. has\_converged kiểm tra điều kiện dừng của thuật toán.

**def** kmeans\_init\_centroids(X, k):

*# randomly pick k rows of X as initial centroids*

**return** X[np.random.choice(X.shape[0], k, replace=False)]

|  |  |
| --- | --- |
| **def** | kmeans\_assign\_labels(X, centroids):   * *calculate pairwise distances btw data and centroids* D = cdist(X, centroids) * *return index of the closest centroid*   **return** np.argmin(D, axis = 1) |
| **def** | has\_converged(centroids, new\_centroids):  *# return True if two sets of centroids are the same* **return** (**set** ([**tuple**(a) **for** a **in** centroids]) == **set**([**tuple**(a) **for** a **in** new\_centroids])) |

**def** kmeans\_update\_centroids(X, labels, K): centroids = np.zeros((K, X.shape[1])) **for** k **in range**(K):

*# collect all points that are assigned to the k-th cluster* Xk = X[labels == k, :]

centroids[k,:] = np.mean(Xk, axis =0) *# take average* **return** centroids

Phần chính của phân cụm K-means:

**def** kmeans(X, K):

centroids = [kmeans\_init\_centroids(X, K)] labels = []

it = 0

**while** True:

labels.append(kmeans\_assign\_labels(X, centroids[-1])) new\_centroids = kmeans\_update\_centroids(X, labels[-1], K) **if** has\_converged(centroids[-1], new\_centroids):

**break**

centroids.append(new\_centroids)

it += 1

**return** (centroids, labels, it)

Ắp dụng thuật toán vừa viết vào dữ liệu ban đầu và hiển thi kết quả cuối cùng:

centroids, labels, it = kmeans(X, K)

**print**('Centers found by our algorithm:\n', centroids[-1]) kmeans\_display(X, labels[-1])

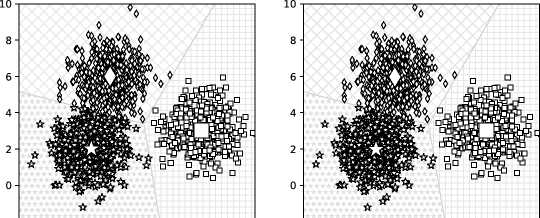
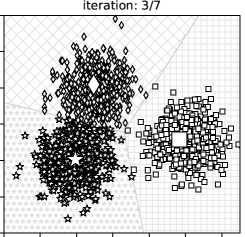
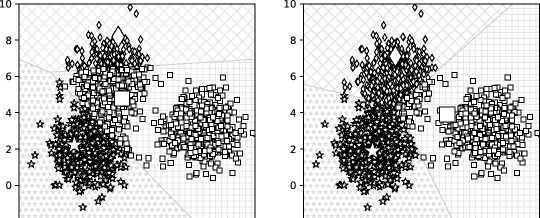
Kết quả:

Centers found by our algorithm: [[ 1.9834967 1.96588127]

[ 3.02702878 5.95686115]

[ 8.07476866 3.01494931]]

Hình [11.2](#bookmark1156) minh hoạ thuật toán phân cụm K-means trên tập dữ liệu này sau một số vòng lặp. Nhận thấy rằng tâm cụm và các vùng *lãnh thể* của chúng thay đổi qua các vòng lặp và hội tụ chỉ sau sáu vòng lặp. Từ kết quả này ta thấy rằng thuật toán phân cụm K-means làm việc khá thành công, các tâm cụm tìm được gần với các tâm cụm ban đầu và các nhóm dữ liệu được phân ra gần như hoàn hảo (một vài điểm gần ranh giới giữa hai cụm hình thoi và hình sao có the lẫn vào nhau).

**10**

**iteration: 1/7 iteration: 2/7**

**—2 H I I I 1 1 1 1 —2 H 1 1 1 1 1 1—**

**-2 0 2 4 6 8 10 -2 0 2 4 6 8 10**

**10**

**8**

**6**

**4**

**2**

**0**

**-2**

**-2 0 2 4 6 8 10**

**iteration: 4/7 iteration: 5/7**

—24 I I I 1 1 1 1 **—2 H** 1 1 1 1 1 **1—**

**-2 0 2 4 6 8 10 -2 0 2 4 6 8 10**

**8**

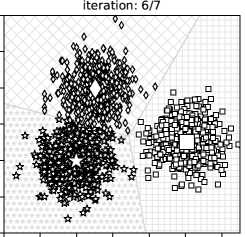
**6**

**4**

**2**

**0**

**-2**

**Hình 11.2.** Thuật toán phân cụm K-means qua các vòng lặp.

**-2 0 2 4 6 8 10**

1. Kết quả tìm được bằng thư viện scikit-learn

Để kiểm tra thêm, chúng ta hãy so sánh kết quả trên với kết quả thu được bằng cách sử dụng thư viện [scikit—learn](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html).

**from** sklearn.cluster **import** KMeans

model = KMeans(n\_clusters=3, random\_state=0).fit(X)

**print**('Centers found by scikit-learn:')

**print**(model.cluster\_centers\_) pred\_label = model.predict(X) kmeans\_display(X, pred\_label)

Kết quả:

Centroids found by scikit-learn: [[ 8.0410628 3.02094748]

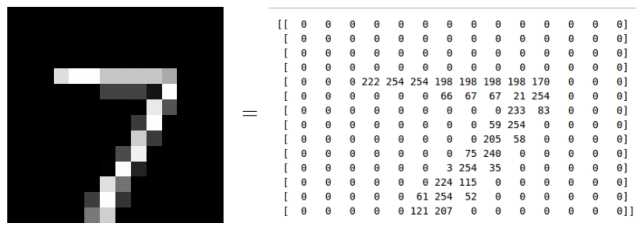
[ 2.99357611 6.03605255]

[ 1.97634981 2.01123694]]

Ta nhận thấy rằng các tâm cụm tìm được rất gần với kết quả kỳ vọng.

Tiếp theo, chúng ta cùng xem xét ba ứng dụng đơn giản của phân cụm K-means.

**EEanmanEEHBiiEiBEEpsa**

**BHBSSHBHBSHHBHBBBEE1E BnaaHEHHBnHESSHHHnnB BHEISaSHHBHaEEBBBnDHS EBBSHHHBEEE3B1E1BSHBHHB HBBBBEBBBnHHaHEEBMBS EHBHEBHHElEBBBBHBBnHEl BEHElBEEBSSBSBnsnHEinB BBBBHBHBEBHnBBHHEEiBS EBBBBBBBBSDEBHBBBBISB**

**Hình 11.3.** 200 mẫu ngẫu nhiên trong bộ cơ sở dữ liệu MNIST.

**Hình 11.4.** Ví dụ về chữ số 7 và giá trị các pixel của nó.

1. Phân cụm chữ số viết tay
2. Bộ cơ sở dữ liệu MNIST

MNIST là bộ cơ sở dữ liệu lớn nhất về chữ số viết tay và được sử dụng trong hầu hết các thuật toán phân loại hình ảnh. MNIST bao gồm hai tập con: tập huấn luyện có 60 nghìn mẫu và tập kiểm tra có 10 nghìn mẫu. Tất cả đều đã được gán nhãn. Hình 11.3 hiển thị 200 mẫu được trích ra từ MNIST.

Mỗi bức ảnh là một ảnh xám (chỉ có một kênh), có kích thước 28 X 28 điểm ảnh (tức 784 điểm ảnh). Mỗi điểm ảnh mang giá trị là một số tự nhiên từ 0 đến 255. Các điểm ảnh màu đen có giá trị bằng không, các điểm ảnh càng trắng thì có giá trị càng cao. Hình [11.4](#bookmark1231) là một ví dụ về chữ số 7 và giá trị các điểm ảnh của nó.

1. Bài toán phân cụm giả định

***Bài toán***: Giả sử ta không biết nhãn của các bức ảnh, hãy phân các bức ảnh gần giống nhau về một cụm.

Bài toán này có thể được giải quyết bằng phân cụm K-means. Mỗi bức ảnh có thể được coi là một điểm dữ liệu với vector đặc trưng là vector cột 784 chiều. Vector này nhận được bằng cách chồng các cột của ma trận điểm ảnh lên nhau.

1. Làm việc trên Python

Để tải về MNIST, chúng ta có thể dùng trực tiếp một hàm số trong scikit-learn:

**from** future **import** print\_function

**import** numpy as np

**from** sklearn.datasets **import** fetch\_mldata

data\_dir = '../../data' *# path to your data folder*

mnist = fetch\_mldata('MNIST original', data\_home=data\_dir) **print**("Shape of minst data:", mnist.data.shape)

Kết quả:

Shape of minst data: (70000, 784)

shape của ma trận dữ liệu mnist.data là (7®®0®, 784) tức có 70000 mẫu, mỗi mẫu có kích thước 784. Chú ý rằng trong scikit-learn, mỗi điểm dữ liệu thường được lưu dưới dạng một vector hàng. Tiếp theo, chúng ta lấy ra ngẫu nhiên 10000 mẫu và thực hiện phân cụm K-means trên tập con này:

**from** sklearn.cluster **import** KMeans

**from** sklearn.neighbors **import** NearestNeighbors

K = 10 *# number of clusters*

N = 10000

X = mnist.data[np.random.choice(mnist.data.shape[0], N)]

kmeans = KMeans(n\_clusters=K).fit(X)

pred\_label = kmeans.predict(X)

Sau khi thực hiện đoạn code trên, các tâm cụm được lưu trong biến kmeans. cluster\_centers\_, nhãn của mỗi điểm dữ liệu được lưu trong biến pred\_label. Hình 11.5 hiển thi các tâm cụm tìm được và 20 mẫu ngẫu nhiên được phân vào cụm tương ứng. Mỗi hàng tương ứng với một cụm, cột đầu tiên bên trái là các tâm cụm tìm được. Ta thấy rằng các tâm cụm đều giống với một chữ số hoặc là kết hợp của hai/ba chữ số nào đó. Ví dụ, tâm cụm ở hàng thứ tư là sự kết hợp của các chữ số 4, 7, 9; ở hàng thứ bảy là kết hợp của các chữ số 7, 8 và 9.

Nhận thấy rằng các bức ảnh lấy ra ngẫu nhiên từ mỗi cụm không thực sự giống nhau. Lý do có thể vì những bức ảnh này ở xa các tâm cụm mặc dù tâm cụm đó đã là gần nhất. Như vậy phân cụm K-means làm việc chưa thực sự tốt trong trường hợp này. Tuy nhiên, chúng ta vẫn có the khai thác một số thông tin hữu ích sau khi thực hiện thuật toán. Thay vì chọn ngẫu nhiên các bức ảnh trong mỗi cụm, ta chọn 20 bức ảnh gần tâm của mỗi cụm nhất, vì càng gần tâm thì độ tin cậy càng cao. Quan sát Hình [11.6](#bookmark1169), có thể thấy dữ liệu trong mỗi hàng khá giống

ã •ĩ

arararaEraraarararaESHraraErap]

**BBnSnnnnnainnnnnBiH ãBHHHBEBElBHSBBBElEBB**

fe £

*& q* **ẵ**

*(ữ*

*(0*

***Cữ***

**QQDSĐDEĐE9DE1E1B innnnnnnnnnnnn IBBiaBriHriBBBESEB inraEHEnnHasnsH**

aHEHHraaiaraBiErara

**3C1EE1BHE1E1HBHSE1**

| *Q* | *0* | 0 | *0* | 0 | 0 | ***Ũ*** | *0* | ố | 0 | **0** | 0 | 0 | 0 | ớ | 0 | D | 0 | 0 | *ữ* | *0* |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *ỉ* | ***i*** | / | *1* | / | *ĩ* | X | / | / |  | / | / | / | / | *í* | / | *!* | *1* |  | / | *1* |
| *2* | 2 | 2 | z | 2 |  |  | <2 | 2 | 7- | X | X | 2 | 2 | *ă* | X | **2-** | 2 | X | 2 | *ă* |
| •ĩ | 7 | X | X | ***±*** | X | *ĩ* | ì | X | *X* | X | X | X | *X* | X | 2 | X | X | X | X | *X* |
| i | 1 | X | X | X | 1 | **(** | X | X | 1 | X | X | 1 | I | X | X | **t** | 1 | X | X | í |
|  | ó | 6 | G | ***(ữ*** | G | **X** | (o | *k* | £ | *Cữ* | *Cữ* | X | Ờ | X | X | **X** | & | 6 | X | X |
| *i* | ***9*** | X | X | *2* | X | **X** | ? | X | 7 | X | *~ỹ* | X | X | X | **X** | X | X | ? | X | **X** |
| *&* | ***Ế*** | ó | X | ố | *6* | *c* | 6 | **ế** | 6 | é | ó | 6 | é | £ | Ò | ế | é | *0* | ể | X |
| *q* | H | 4 |  |  | *4* | *X* | 4 | 4 | X | 4 | X | 4 | X | X | 4 | **H** | X | 4 | X | X |
| 5 | s | *3* | s | 3 | *s* | *s* | 3 | **3** | 3 | 3 | 3 | 3 | 5 | 3 | 5 | **3** | ì | *5* | *ỉ* | 3 |

5

**Hình 11.6.** Tâm và

20 điếm gần tâm nhất của mỗi cụm.

**Hình 11.5.** Các tâm

cụm (cột đầu) và 20 điếm ngẫu nhiên trong mỗi cụm. Các chữ số trên mỗi hàng thuộc vào còng một cụm.

nhau và giống với tâm cụm ở cột đầu tiên bên trái. Từ đây có thể rút ra một vài quan sát thú vi:

1. Có hai kiểu viết chữ số 1 - thẳng và chéo. Phân cụm K-means nghĩ rằng đó là hai chữ số khác nhau. Điều này là dễ hiểu vì phân cụm K-means là một thuật toán học không giám sát. Nếu có sự can thiệp của con người, chúng có the được nhóm lại thành một.
2. ở hàng thứ chín, chữ số 4 và 9 được phân vào cùng một cụm. Sự thật là hai chữ số này khá giống nhau. Điều tương tự xảy ra đối với hàng thứ bảy với các chữ số 7, 8, 9. Phân cụm K-means có thể được áp dụng để tiếp tục phân nhỏ các cụm đó.

Một kỹ thuật phân cụm thường được sử dụng là *phân cụm theo tầng (hierarchical clustering*). Có hai loại phân cụm theo tầng:

*• Agglomerative* tức “đi từ dưới lên”. Ban đầu ta chọn K là một số lớn gần bằng số điểm dữ liệu. Sau khi thực hiện phân cụm K-means lần đầu, các cụm gần nhau được ghép lại thành một cụm. Khoảng cách giữa các cụm có thể được xác đinh bằng khoảng cách giữa các tâm cụm. Sau bước này, ta thu được một số lượng cụm nhỏ hơn. Tiếp tục phân cụm K-means với điểm khởi tạo là tâm của cụm lớn vừa thu được. Lặp lại quá trình này đến khi nhận được kết quả chấp nhận được.

**Hình 11.7.** Tách vật the trong ảnh.

*• Divisive* tức “đi từ trên xuống”. Ban đầu, thực hiện phân cụm K-means với K nhỏ để được các cụm lớn. Sau đó tiếp tục áp dụng phân cụm K-means vào mỗi cụm lớn đến khi kết quả chấp nhận được.

1. Tách vật thể trong ảnh

Phân cụm K-means cũng được áp dụng vào bài toán *tách vật thể trong ảnh (object segmentation*). Cho bức ảnh như trong Hình [11.7](#bookmark1250), hãy xây dựng một thuật toán tự động nhận diện và tách rời vùng khuôn mặt.

Bức ảnh có ba màu chủ đạo: hồng ở khan và môi; đen ở mắt, tóc, và hậu cảnh; màu da ở vùng còn lại của khuôn mặt. Ảnh này khá rõ nét và các vùng được phân biệt rõ ràng bởi màu sắc nên chúng ta có thể áp dụng thuật toán phân cụm K-means. Thuật toán này sẽ phân các điểm ảnh thành ba cụm, cụm chứa phần khuôn mặt có thể được chọn tự động hoặc bằng tay.

Đây là một bức ảnh màu, mỗi điểm ảnh được biểu diễn bởi ba giá tri tương ứng với màu đỏ, lục, và lam (RGB). Nếu coi mỗi điểm ảnh là một điểm dữ liệu được mô tả bởi một vector ba chiều chứa các giá tri này, sau đó áp dụng phân cụm K-means, chúng ta có thể đạt được kết quả như mong muốn.

1. Làm việc trẽn Python

Khai báo thư viện và hiển thi bức ảnh:

**import** matplotlib.image as mpimg **import** matplotlib.pyplot as plt **import** numpy as np

**from** sklearn.cluster **import** KMeans img = mpimg.imread('girl3.jpg') plt.imshow(img)

imgplot = plt.imshow(img) plt.axis('off')

plt.show()

**Hình 11.8.** Kết quả nhận được sau khi thực hiện phân cụm K- means. Có ba cụm tương ứng vói ba màu đỏ, hồng, đen.

Biến đổi bức ảnh thành một ma trận mà mỗi hàng là ba giá tri màu của một điểm ảnh:

X = img.reshape((img.shape[0]\*img.shape[1], img.shape[2]))

Sau khi tìm được các cụm, giá tri của mỗi pixel được thay bằng giá tri của tâm tương ứng. Kết quả được minh hoạ trên Hình 11.8. Ba màu đỏ, đen, và màu da đã được phân nhóm khá thành công. Khuôn mặt có the được tách ra từ phần có màu da và vùng bên trong nó. Như vậy, phân cụm K-means tạo ra một kết quả chấp nhận được cho bài toán này.

1. Nén ảnh

Trước hết, xét đoạn code dưới đây:

**for** K **in** [5, 10, 15, 20]:

kmeans = KMeans(n\_clusters=K).fit(X) label = kmeans.predict(X)

img4 = np.zeros\_like(X)

* *replace each pixel by its centroid*

***for*** k **in range**(K):

img4[label == k] = kmeans.cluster\_centroids\_[k]

* *reshape and display output image*

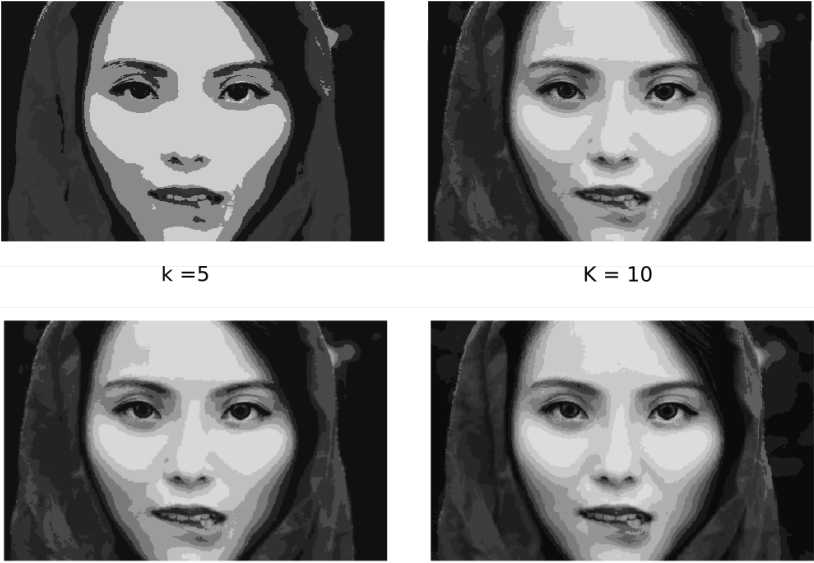
img5 = img4.reshape((img.shape[0], img.shape[1], img.shape[2]))

plt.imshow(img5, interpolation^ nearest')

plt.axis('off')

plt.show()

Nhận thấy rằng mỗi điểm ảnh có thể nhận một trong số 2563 ~ 16 triệu màu. Đây là một số rất lớn (tương đương với 24 bit cho một điểm ảnh). Phân cụm K-means có thể được áp dụng để nén ảnh với số bit ít hơn. Phép nén ảnh này làm mất dữ liệu nhưng kết quả vẫn chấp nhận được. Quay trở lại bài toán tách vật thể trong mục trước, nếu thay mỗi điểm ảnh bằng tâm cụm tương ứng, ta



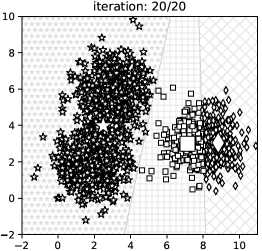
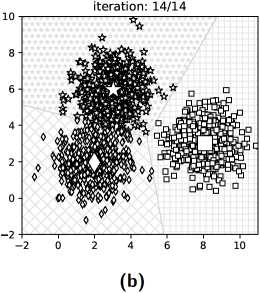
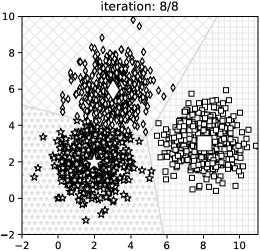
K = 15 K = 20

**Hình 11.9.** Chat lượng nén ảnh vói số lượng cluster khác nhau.

thu được một bức ảnh nén. Tuy nhiên, chất lượng bức ảnh rõ ràng đã giảm đi nhiều. Trong đoạn code trên đây, ta đã làm một thí nghiệm nhỏ với số lượng cụm tang lên 5, 10,15, 20. Sau khi tìm được tâm cho mỗi cụm, giá tri của một điểm ảnh được thay bằng giá tri của tâm tương ứng. Kết quả được hiển thi trên Hình 11.9. Có thể thấy rằng khi số lượng cụm tang lên, chất lượng bức ảnh đã được cải thiện. Đe nén bức ảnh này, ta chỉ cần lưu K tâm cụm tìm được và nhãn của mỗi điểm ảnh.

1. Thảo luận
2. Hạn chế của phân cụm *K*-means

* *Số cụm K cần được xác định trước.* Trong trường hợp, chúng ta không biết trước giá tri này.
* *Nghiệm cuối còng phụ thuộc vào các tâm cụm được khởi tạo ban đầu.* Thuật toán phân cụm K-means không đảm bảo tìm được nghiệm tối ưu toàn cục, nghiệm cuối cùng phụ thuộc vào các tâm cụm được khởi tạo ban đầu. Hình 11.10 the hiện các kết quả khác nhau khi các tâm cụm được khởi tạo khác nhau. Ta cũng thấy rằng trường hợp (a) và (b) cho kết quả tốt, trong khi



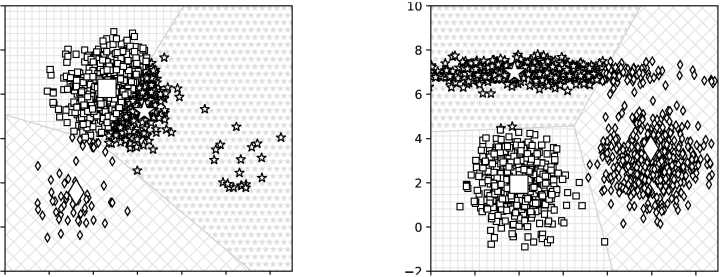
**(a)**

**(c)**

**Hình 11.10.** Các giá trị khởi tạo ban đầu khác nhau dẫn đến các nghiệm khác nhau.

kết quả thu được ở trường hợp (c) không thực sự tốt. Một điểm nữa có thể rút ra là số lượng vòng lặp tới khi thuật toán hội tụ cũng khác nhau. Trường hợp (a) và (b) cùng cho kết quả tốt nhưng (b) chạy trong thời gian gần gấp đôi. Một kỹ thuật giúp hạn chế nghiệm xấu như trường hợp (c) là chạy thuật toán phân cụm K-means nhiều lần với các tâm cụm được khởi tạo khác nhau và chọn ra lần chạy cho giá tri hàm mất mát thấp nhất. Ngoài ra, có một vài thuật toán giúp chọn các tâm cụm ban đầu.

* *Các cụm cần có số lượng điểm gần bằng nhau.* Hình [11.11a](#bookmark1281) minh hoạ kết quả khi các cụm có số điểm chênh lệch. Trong trường hợp này, nhiều điểm lẽ ra thuộc cụm hình vuông đã bi phân nhầm vào cụm hình sao.
* *Các cụm cần có dạng hình tròn (cầu).* Khi các cụm vẫn tuân theo phân phối chuẩn nhưng ma trận hiệp phương sai không tỉ lệ với ma trận đơn vi, các cụm sẽ không có dạng tròn (hoặc cầu trong không gian nhiều chiều). Khi đó, phân cụm K-means không hoạt động hiệu quả. Lý do chính là vì phân cụm K-means quyết đinh nhãn của một điểm dữ liệu dựa trên khoảng cách Euclid của nó tới các tâm. Trong trường hợp này, *Gaussian mixture models* (GMM). Trong GMM, mỗi cụm được giả sử tuân theo một phân phối chuẩn với ma trận hiệp phương sai không nhất thiết tỉ lệ với ma trận đơn vi. Ngoài các tâm cụm, các ma trận hiệp phương sai cũng là các biến cần tối ưu trong GMM.
* *Khi một cụm nằm trong cụm khác.* Hình [11.12](#bookmark1282) là một ví dụ kinh điển về việc phân cụm K-means không làm việc. Một cách tự nhiên, chúng ta sẽ phân dữ liệu ra thành bốn cụm: mắt trái, mắt phải, miệng, xung quanh mặt. Nhưng vì mắt và miệng nằm trong khuôn mặt nên phân cụm K-means cho kết quả không chính xác. Với dữ liệu như trong ví dụ này, *phân cụm spectral* sẽ cho kết quả tốt hơn. Phân cụm spectral cũng coi các điểm gần nhau tạo thành một cụm, nhưng không giả sử về một tâm chung cho cả cụm. Phân cụm spectral được thực hiện dựa trên một đồ thi vô hướng với đỉnh là các điểm dữ liệu và

cạnh được nối giữa các điểm gần nhau, môi cạnh được đánh trọng số là một hàm của khoảng cách giữa hai điểm.

**iteration: 15/15**

**-2 0 2 4 6 8 10 -2 0 2 4 6 8 10**

**(a) (b)**

**iteration: 12/12**

**10**

**8**

**6**

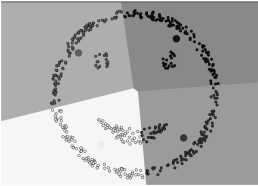
**4**

**2**

**0**

**-2**

**Hình 11.11.** Phân cụm K-means hoạt động không thực sự tốt trong trường hợp các cụm có số lượng phan tử chênh lệch hoặc các cụm không có dạng hình tròn.

**Hình 11.12.** Một ví dụ về việc phân cụm K-means không hoạt động hiệu quả.

1. Các ứng dụng khác của phân cụm *K*-means

Mặc dù có những hạn chế, phân cụm K-means vẫn cực kỳ quan trọng trong học máy và là nền tảng cho nhiều thuật toán phức tạp khác. Dưới đây là một vài ứng dụng khác của phân cụm K-means.

Cách thay một điểm dữ liệu bằng tâm cụm tương ứng là một trong số các kỹ thuật có tên chung là *vector quantization* - *VQ.* Không chỉ được áp dụng trong nén dữ liệu, VQ còn được kết hợp với Bag-of-Words áp dụng rộng rãi trong các thuật toán xây dựng vector đặc trưng.

Ngoài ra, VQ cũng được áp dụng vào các bài toán tìm kiếm trong cơ sỏ dữ liệu lớn. Khi số điểm dữ liệu là rất lớn, việc tìm kiếm trỏ nên cực kỳ quan trọng. Khó khan chính của việc này là làm thế nào có thể tìm kiếm một cách nhanh chóng trong lượng dữ liệu khổng lồ đó. Ý tưỏng cơ bản là sử dụng các thuật toán phân cụm để phân các điểm dữ liệu thành nhiều cụm nhỏ. Đe tìm các điểm gần nhất của một điểm *truy vấn,* ta có thể tính khoảng cách giữa điểm này và các tâm cụm thay vì toàn bộ các điểm trong cơ sở dữ liệu.

***B ài 12***

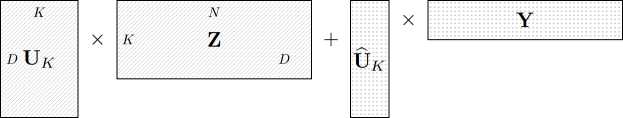
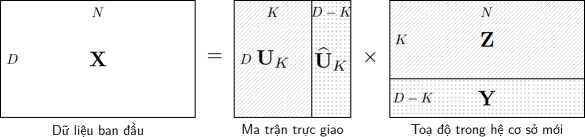
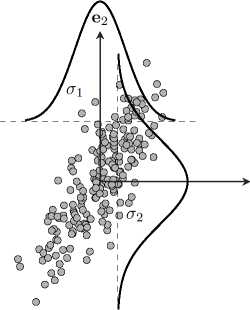
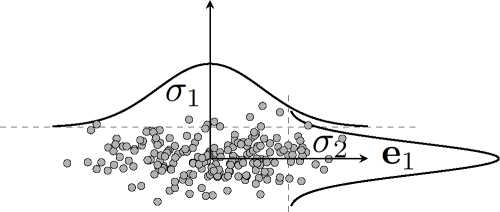
Phân tích thành phần chính

1. Phân tích th ành phần chính
2. Ý tưởng

Giả sử vector dữ liệu ban đầu **x** 2 RD được giảm chiều trở thành **z** 2 RK với *K < D.* Một cách đơn giản để giảm chiều dữ liệu từ *D* về *K < D* là chỉ giữ lại *K* phần tử quan trọng nhất. Có hai câu hỏi lập tức được đặt ra. Thứ nhất, làm thế nào để xác đinh tầm quan trọng của mỗi phần tử? Thứ hai, nếu tầm quan trọng của các phần tử là như nhau, ta cần bỏ đi những phần tử nào?

Để trả lời câu hỏi thứ nhất, hãy quan sát Hình 12.1a. Giả sử các điểm dữ liệu có thành phần thứ hai (phương đứng) giống hệt nhau hoặc sai khác nhau không đáng kể (phương sai nhỏ). Khi đó, thành phần này hoàn toàn có thể được lược bỏ, và ta ngầm hiểu rằng nó sẽ được xấp xỉ bằng kỳ vọng của thành phần đó trên toàn bộ dữ liệu. Ngược lại, nếu áp dụng phương pháp này lên chiều thứ nhất (phương ngang), lượng thông tin bi mất đi đáng kể do sai số xấp xỉ quá lớn. Vì vậy, lượng thông tin theo mỗi thành phần có thể được đo bằng phương sai của dữ liệu trên thành phần đó. Tổng lượng thông tin là tổng phương sai trên toàn bộ các thành phần. Lấy một ví dụ về việc có hai camera được đặt dùng để chụp cùng một người, một camera phía trước và một camera đặt trên đầu. Rõ ràng, hình ảnh thu được từ camera đặt phía trước mang nhiều thông tin hơn so với hình ảnh nhìn từ phía trên đầu. Vì vậy, bức ảnh chụp từ phía trên đầu có thể được bỏ qua mà không làm mất đi quá nhiều thông tin về hình dáng của người đó.

Câu hỏi thứ hai tương ứng với trường hợp Hình 12.1b. Trong cả hai chiều, phương sai của dữ liệu đều lớn; việc bỏ đi một trong hai chiều đều khiến một lượng thông tin đáng kể bi mất đi. Tuy nhiên, nếu xoay trục toạ độ đi một góc phù hợp, một



e2

**(a)**

**(b)**

ei

**Hình 12.1.** Ví dụ ve phương sai của dữ liệu trong không gian hai chiều. (a) Phương sai của chiều thử hai (tỉ lệ vói độ rộng của đường hình chuông) nhỏ hơn phương sai của chiều thứ nhất. (b) Cả hai chiều có phương sai đáng ke. Phương sai của mỗi chiều là phương sai của thành phan tương ứng được lấy trên toàn bộ dữ liệu. Phương sai tỉ lệ thuận vói độ phân tán của dữ liệu.

**Hình 12.2.** Ý tưởng chính của PCA: Tìm một hệ trực chuẩn mói sao cho trong hệ này, các thành phan quan trọng nhất nằm trong *K* thành phan đau tiên.

trong hai chiều dữ liệu có thể được lược bở vì dữ liệu có xu hướng phân bố xung quanh một đường thẳng.

*Phân tích thành phần chính* (principle component analysis, PCA) là phương pháp đi tìm một phép xoay trục toạ độ để được một hệ trục toạ độ mới sao cho trong hệ mới này, thông tin của dữ liệu chủ yếu tập trung ở một vài thành phần. Phần còn lại chứa ít thông tin hơn có thể được lược bỏ.

Phép xoay trục toạ độ có liên hệ chặt chẽ tới hệ trực chuẩn và ma trận trực giao. Giả sử hệ cơ sở trực chuẩn mới là **U** (mỗi cột của **U** là một vector đơn vi cho một chiều) và ta muốn giữ lại K toạ độ trong hệ cơ sở mới này. Không mất tính tổng quát, giả sử đó là K thành phần đầu tiên. Quan sát Hình [12.2](#bookmark1287) với cơ sở mới **U** = [**U**K, **U***K*] là một hệ trực chuẩn với **U**K là ma trận con tạo bởi K cột đầutiên của **U**. Trong hệ cơ sở mới này, ma trận dữ liệu có thể được viết thành

**X** = **U**k **Z** + **U** *K* **Y**

(12.1)

Từ đây ta cũng suy ra

\_UK’

\_UK\_

**Z** = **U**K **X  
Y** = **u**K **X**

(12.2)

Mục đích của PCA là đi tìm ma trận trực giao **U** sao cho phần lớn thông tin nằm ở **U**K**Z**, phần nhỏ thông tin nằm ở **U**K**Y**. Phần nhỏ này sẽ được lược bỏ và xấp xỉ bằng một ma trận có các cột như nhau. Gọi mỗi cột đó là **b**, khi đó, ta sẽ xấp xỉ **Y b1**T với **1**T 2 R1xN là một vector hàng có toàn bộ các phần tử bằng một. Giả sử đã tìm được **U**, ta cần tìm **b** thoả mãn:

**b** = argminb|**Y - b1**TkF = argminb||UK**X - b1**TkF (12.3)

Giải phương trình đạo hàm theo **b** của hàm mục tiêu bằng **0**:

(**b1**T **- U**K**X**)**1** = 0 ) N**b** = **U**K**X1** ) **b** = **U**K**x**. (12.4)

ở đây ta đã sử dụng **1**T**1** = *N* và **x** = N**X1** là vector trung bình các cột của **X**. Với giá tri **b** tìm được này, dữ liệu ban đầu sẽ được xấp xỉ bởi

**X** = **U**k**Z** + **U***k***Y** « **U**k**Z** + **U***k***b1**T = **U**k**Z** + **U**K**U**K**x 1**T - ***X*** (12.5)

1. H àm mất mát

Hàm mất mát của PCA được coi như sai số của phép xấp xỉ, được đinh nghĩa là

1 IIX **X** 112 1 11U , - Y U, -U'/' Ỹ1 TII2 1 I IU T .U'T' X U, -U'/' Ỹ1 T II2

N llX \_ XIIF *= N* k UK Y \_ UK UKx1 IIF *= N* k U K UKX UK UKx1 IIF

*=* Nll**Gáy**(**X - x1**T)||F - *J*(**U**) (12.6)

Chú ý rằng, nếu các cột của một ma trận **V** tạo thành một hệ trực chuẩn thì với một ma trận **W** bất kỳ, ta luôn có

**IIVWII**F = trace(**W**T**V**T**VW**) = trace(**W**T**W**) = **||W||**F (12.7)

Đặt **X** = **X — x1**T. Ma trận này có được bằng cách trừ mỗi cột của **X** đi trung bình các cột của nó. Ta gọi **X** là ma trận dữ liệu đã được chuẩn hoá. Có thể thấy ***x****n* = ***x****n* - x, =1, 2,..., N.

Vì vậy hàm mất mát trong [(12.6)](#bookmark1310) có thể được viết lại thành:

1 „ 1 „ 1 DÁ

*T(****U****)*  1 I IttT 'VI I2 1 I IxT U.d I2 1 \ " IIxT,,. II2 (128)

J(U) — NllUKXIIF — N X UKIIF — *N /* X uill2 (12-8)

*i=K*+1

1 *D D*

— N Ĩ2 uT*XXT*ui lí uTSui (12-9)

i=K+1 i=K +1

với **S** — N **X X**T là ma trận hiệp phương sai của dữ liệu và luôn là một ma trận nửa xác đinh dương.

Công việc còn lại là tìm các **u**i để mất mát là nhỏ nhất.

Với ma trận **U** trực giao bất kỳ, thay *K* — 0 vào [(12.9)](#bookmark1311) ta có

L = ***u****T***Su**i — N **V UI**IF — Ntrace(**X**T**UU**T**X**)

(12.10)

(12.11)

i=1

11 *X*D

— Ntrace(**X**T**X**) — Ntrace(**XX**T) — trace(**S**) — Ai

Với A1 *>* A2 *> ■ ■ ■ > AD >* 0 là các tri riêng của ma trận nửa xác đinh dương **S**. Chú ý rằng các tri riêng này là thực và không â[m[[7]](#footnote-8)](#bookmark4).

*Như vậy L không phụ thuộc vào cách chọn ma trận trực giao* **U** và bằng tổng các phần tử trên đường chéo của **S**. Nói cách khác, L chính là tổng các phương sai theo từng thành phần của dữ liệu ban đầ[u[[8]](#footnote-9)](#bookmark5).

Vì vậy, việc tối thiểu hàm mất mát *J* được cho bởi [(12.9)](#bookmark1311) tương đương với việc tối đa biểu thức

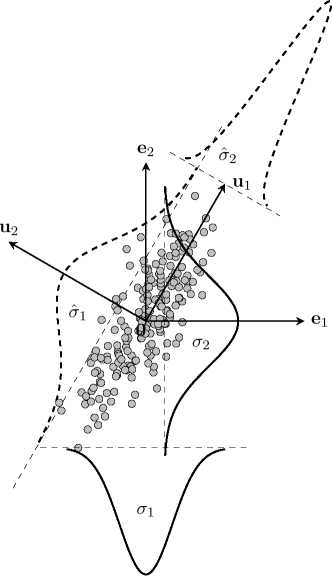
*X*K

*F* — L - J — 22 ***u****i****Su****T* (12.12)

i=1

1. Tối ưu h àm mất mát

Nghiệm của bài toán tối ưu hàm mất mát PCA được tìm dựa trên khẳng đinh sau đây:

**Hình 12.3.** PCA có the được coi là phương pháp đi tìm một hệ cơ sở trực chuan đóng vai trò một phép xoay, sao cho trong hệ cơ sở mói này, phương sai theo một số chiều nào đó là không đáng ke và có the lược bỏ. Trong hệ cơ sở ban đau **Oe**i**e**2, phương sai theo mỗi chiều (độ rộng của các đường hình chuông nét lien) đều lón. Trong không gian mói vói hệ cơ sở **Ou**1**u**2, phương sai theo hai chiều (độ rộng của các đường hình chuông nét đứt) chênh lệch nhau đáng ke. Chieu dữ liệu có phương sai nhỏ có the được lược bỏ vì dữ liệu theo chiều này ít phân tán.

*Nếu* **S** *là một ma trận nửa xác định dương, bài toán tối ưu*

K

max > **u**T**Su-**

Uk \*

*i=1*

*có nghiệm* **u**1;...; **u**K *là các vector riêng ứng với K trị riêng (kể cả lặp) lớn nhất của* **S**. *Khi đó, giá trị lớn nhất của hàm mục tiêu là ^2K=* 1 *Xi, với* A1 *>* A2 *> ■ ■ ■ > XD là các trị riêng của* **S**.

(12.13)

(12.14)

*thoả mãn:* **U**K**U**K = **I**

Khẳng đinh này có thể được chứng minh bằng quy nạp.

Tri riêng lớn nhất A1 của ma trận hiệp phương sai **S** còn được gọi là *thành phần chính thứ nhất* (the first principal component), tri riêng thứ hai A2 được gọi là *thành phần chính thứ hai,...* Tên gọi *phân tích thành phần chính* (principal component analysis) bắt nguồn từ đây. Ta chỉ giữ lại K thành phần chính đầu tiên khi giảm chiều dữ liệu dùng PCA.

Hình [12.3](#bookmark1316) minh hoạ các thành phần chính với dữ liệu hai chiều. Trong không gian ban đầu với các vector cơ sở **e**1, **e**2, phương sai theo mỗi chiều dữ liệu (tỉ lệ với độ rộng của các hình chuông nét liền) đều lớn. Trong hệ cơ sở mới **Ou**1**u**2, phương sai theo chiều thứ hai (72 nhỏ so với (72. Điều này chỉ ra rằng khi chiếu dữ liệu lên **u**2, ta được các điểm rất gần nhau và gần với giá tri trung bình theo chiều đó. Trong trường hợp này, vì giá tri trung bình theo mọi chiều bằng 0, ta có thể thay thế toạ độ theo chiều **u**2 bằng 0. Rõ ràng là nếu dữ liệu có phương sai càng nhỏ theo một chiều nào đó thì khi xấp xỉ chiều đó bằng một hằng số, sai số xấp xỉ càng nhỏ. PCA thực chất là đi tìm một phép xoay tương ứng với một ma trận trực giao sao cho trong hệ toạ độ mới, tồn tại các chiều có phương sai nhỏ có thể được bỏ qua; ta chỉ cần giữ lại các chiều/thành phần khác quan trọng hơn. Như đã khẳng đinh ở trên, tổng phương sai theo toàn bộ các chiều chiều trong một hệ cơ sở bất kỳ là như nhau và bằng tổng các tri riêng của ma trận hiệp phương sai. Vì vậy, PCA còn được coi là phương pháp giảm số chiều dữ liệu sao tổng phương sai còn lại là lớn nhất.

1. Các bước thực hiện phân tích th ành phần chính

Từ các suy luận trên, ta có thể tóm tắt lại các bước trong PCA như sau:

1. Tính vector trung bình của toàn bộ dữ liệu: **x** = *~N^2n=*1 *xn.*
2. Trừ mỗi điểm dữ liệu đi vector trung bình của toàn bộ dữ liệu để được dữ liệu chuẩn hoá:

***x*** *=* ***x****n* ***- x*** (12.15)

1. Đặt **X** = [**X**1, **x**2*;* ***X****D*] là ma trận dữ liệu chuẩn hoá, tính ma trận hiệp

phương sai:

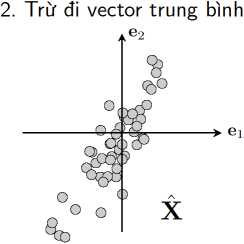
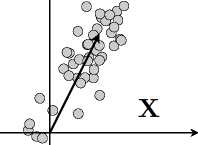
(12.16)

1. Tính các tri riêng và vector riêng tương ứng có '2 norm bằng 1 của ma trận này, sắp xếp chúng theo thứ tự giảm dần của tri riêng.
2. Chọn *K* vector riêng ứng với *K* tri riêng lớn nhất để xây dựng ma trận **U**K có các cột tạo thành một hệ trực giao. K vector này được gọi là các thành phần chính, tạo thành một không gian con gần với phân bố của dữ liệu ban đầu đã chuẩn hoá.
3. Chiếu dữ liệu ban đầu đã chuẩn hoá **X** xuống không gian con tìm được.
4. Dữ liệu mới là toạ độ của các điểm dữ liệu trên không gian mới: **Z** = **U**K**X**.

*Như vậy, PCA là kết hợp của phép tịnh tiến, xoay trục toạ độ và chiếu dữ liệu lên hệ toạ độ mới.*

Dữ liệu ban đầu có thể tính được xấp xỉ theo dữ liệu mới bởi **x** « **U**K**Z** + **x**.

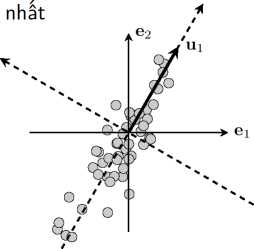
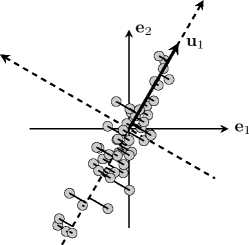
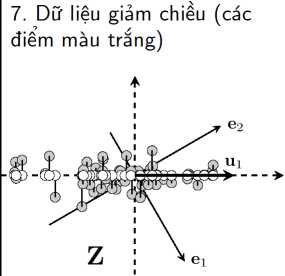
Quy trình thực hiện PCA

3. Tính ma trận hiệp phương sai:

1. Tìm vector trung bình

S = *NxX*

4. Tìm trị riêng và  
vector riêng của S:  
(A1, (Ap, UD).

Các vector riêng được  
chọn phải tạo thành  
một hệ trực chuẩn.

6. Chiếu dữ liệu ban đầu xuống các vector riêng đó

5. Chọn *K* vector riêng ứng vổi các trị riêng lổn

**Hình 12.4.** Các bưóc thực hiện PCA.

Một điểm dữ liệu mới **v** 2 RD sẽ được giảm chiều bằng PCA theo công thức **w** = **U**K(**v — x**) 2 RK. Ngược lại, nếu biết **w**, ta có thể xấp xỉ **v** bởi **U**K**w** + **X**. Các bước thực hiện PCA được minh hoạ trong Hình [12.4.](#bookmark1290)

1. Liên hệ với phân tích giá trị suy biến

PCA và SVD có mối quan hệ đặc biệt với nhau.

1. SVD cho bài toán xấp xỉ hạng thấp tốt nhất

Nghiệm của bài toán xấp xỉ một ma trận bởi một ma trận có hạng không vượt quá k:

min||**X —** A||f

A (12.17)

thoả mãn: rank(**A**) = *K*

chính là SVD cắt ngọn của **A**.

Cụ thể, nếu SVD của **X** 2 RDxN là

**X** = **UXV**t (12.18)

với **U** 2 RDxD và **V** 2 RN*xN* là các ma trận trực giao và **X** 2 RDxN là ma trận đường chéo (không nhất thiết vuông) với các phần tử trên đường chéo không âm giảm dần. Nghiệm của bài toán [(12.17)](#bookmark1338) chính là:

**A** = **I *X****k****V****tk* (12.19)

với **U** 2 RDxK và **V** 2 RNxK là các ma trận tạo bởi *K* cột đầu tiên của **U** và **V**, ***X****k 2 RKxK* là ma trận đường chéo con ứng với K hàng đầu tiên và K cột đầu tiên của **X**.

1. Ý tưởng của PCA

Như đã chứng minh ở [(12.5)](#bookmark1305), PCA là bài toán đi tìm ma trận trực giao **U** và ma trận mô tả dữ liệu ở không gian thấp chiều **Z** sao cho việc xấp xỉ sau đây là tốt nhất:

**X** « ***X*** *=* ***U****k****Z*** + **U***K****U****K***x *1****T* (12.20)

với **U**K, **U***K* lần lượt là các ma trận được tạo bởi K cột đầu tiên và *D — K* cột cuối cùng của ma trận trực giao **U**, và **x** là vector trung bình của dữ liệu.

Giả sử rằng vector trung bình **x** = **0**. Khi đó, [(12.20)](#bookmark1344) tương đương với

**X** « ***X*** *=* **U**K**Z** (12.21)

Bài toán tối ưu của PCA sẽ trở thành:

**U**K, **Z** = arg min **||X — U**K***Z****\\f*

,z

(12.22)

thoả mãn: **U**K**U**k = **I**K

với **I**K 2 RKxK là ma trận đơn vị trong không gian K chiều và điều kiện ràng buộc để đảm bảo các cột của **U**K tạo thành một hệ trực chuẩn.

1. Quan hệ giữa hai phương pháp

Bạn có nhận ra điểm tương đồng giữa hai bài toán tối ưu [(12.17)](#bookmark1338) và [(12.22)](#bookmark1345) với nghiệm của bài toán đầu tiên được cho trong [(12.19)](#bookmark1339)? Có the nhận ra nghiệm của bài toán [(12.22)](#bookmark1345) chính là

**U**K trong [(12.22)](#bookmark1345) = **U**K trong [(12.19)](#bookmark1339)  
**Z** trong [(12.22)](#bookmark1345) = **X**k**V**K trong [(12.19)](#bookmark1339)

Như vậy, nếu các điểm dữ liệu được biểu diễn bởi các cột của một ma trận, và trung bình các cột của ma trận đó là vector không thì nghiệm của bài toán PCA được rút ra trực tiếp từ SVD cắt ngọn của ma trận đó. Nói cách khác, việc đi tìm nghiệm cho PCA chính là việc giải một bài toán phân tích ma trận thông qua SVD.

1. Làm thế nào để chọn số chiều của dữ liệu mới

Một câu hỏi được đặt ra là, làm thế nào để chọn giá tri K - chiều của dữ liệu mới - với từng dữ liệu cụ thể?

Thông thường, *K* được chọn dựa trên việc lượng thông tin muốn giữ lại. ơ đây, toàn bộ thông tin chính là tổng phương sai của toàn bộ các chiều dữ liệu. Lượng dữ liệu muốn dữ lại là tổng phương sai của dữ liệu trong hệ trục toạ độ mới.

Nhắc lại rằng trong mọi hệ trục toạ độ, tổng phương sai của dữ liệu là như nhau và bằng tổng các tri riêng của ma trận hiệp phương sai D1 Xi. Thêm nữa, PCA giúp giữ lại lượng thông tin (tổng các phương sai) là *^2i=* 1 Xi. Vậy ta có thể coi biểu thức:

(12.23)

rK =

j Xj

là tỉ lệ thông tin được giữ lại khi số chiều dữ liệu mới sau PCA là K. Như vậy, giả sử ta muốn giữ lại 99% dữ liệu, ta chỉ cần chọn K là số tự nhiên nhỏ nhất sao cho rK > 0.99.

Khi dữ liệu phân bố quanh một không gian con, các giá tri phương sai lớn nhất ứng với các *Xi* đầu tiên cao gấp nhiều lần các phương sai còn lại. Khi đó, ta có thể chọn được K khá nhỏ để đạt được *rK >* 0.99.

1. Lưu ý về tính toán phân tích thành phần chính

Có hai trường hợp trong thực tế mà chúng ta cần lưu ý về PCA. Trường hợp thứ nhất là lượng dữ liệu có được nhỏ hơn rất nhiều so với số chiều dữ liệu. Trường hợp thứ hai là khi lượng dữ liệu trong tập huấn luyện rất lớn, việc tính toán ma trận hiệp phương sai và tri riêng đôi khi trở nên bất khả thi. Có những hướng giải quyết hiệu quả cho các trường hợp này.

Trong mục này, ta sẽ coi như dữ liệu đã được chuẩn hoá, tức đã được trừ đi vector kỳ vọng. Khi đó, ma trận hiệp phương sai sẽ là **S** = N**XX**T.

1. Số chiều dữ liệu nhiều hơn số điểm dữ liệu

Đó là trường hợp *D > N*, tức ma trận dữ liệu **X** là một *ma trận cao.* Khi đó, số tri riêng khác không của ma trận hiệp phương sai **S** sẽ không vượt quá hạng của nó, tức không vượt quá N. Vậy ta cần chọn K < N vì không thể chọn ra được nhiều hơn N tri riêng khác không của một ma trận có hạng bằng N.

Việc tính toán các tri riêng và vector riêng cũng có thể được thực hiện một cách hiệu quả dựa trên các tính chất sau đây:

1. Tri riêng của **A** cũng là tri riêng của *k****A*** với *k* = 0 bất kỳ. Điều này có thể được suy ra trực tiếp từ đinh nghĩa của tri riêng và vector riêng.
2. Tri riêng của **AB** cũng là tri riêng của **BA** với **A** 2 Rd1Xd2, **B** 2 Rd2Xd1 là các ma trận bất kỳ và *d1, d2* là các số tự nhiên khác không bất kỳ.

Như vậy, thay vì tìm tri riêng của ma trận hiệp phương sai **S** 2 RDxD, ta đi tìm tri riêng của ma trận **T** = **X**T**X** 2 RN*xN* có số chiều nhỏ hơn (vì *N < D).*

1. Nếu (A, **u**) là một cặp tri riêng, vector riêng của **T** thì (A, **Xu**) là một cặp tri riêng, vector riêng của **S**. Thật vậy:

***X****T***Xu** = **Tu** = A**u** ) *(****XX****T*)(**Xu**) = A(**Xu**) (12.24)

Dấu bằng thứ nhất xảy ra theo đinh nghĩa của tri riêng và vector riêng.

Như vậy, ta có thể hoàn toàn tính được tri riêng và vector riêng của ma trận hiệp phương sai **S** dựa trên một ma trận **T** có kích thước nhỏ hơn. Việc này trong nhiều trường hợp khiến thời gian tính toán giảm đi đáng kể.

1. Với các b ài toán quy mô lớn

Trong rất nhiều bài toán quy mô lớn, ma trận hiệp phương sai là một ma trận rất lớn. Ví dụ, có một triệu bức ảnh 1000 X 1000 pixel, như vậy D = N = 106 là các số rất lớn, việc trực tiếp tính toán tri riêng và vector riêng cho ma trận hiệp phương sai là không khả thi. Lúc này, các tri riêng và vector riêng của ma trận hiệp phương sai thường được tính thông qua *power method* [(https://goo.gl/eBRPxH)](https://goo.gl/eBRPxH).

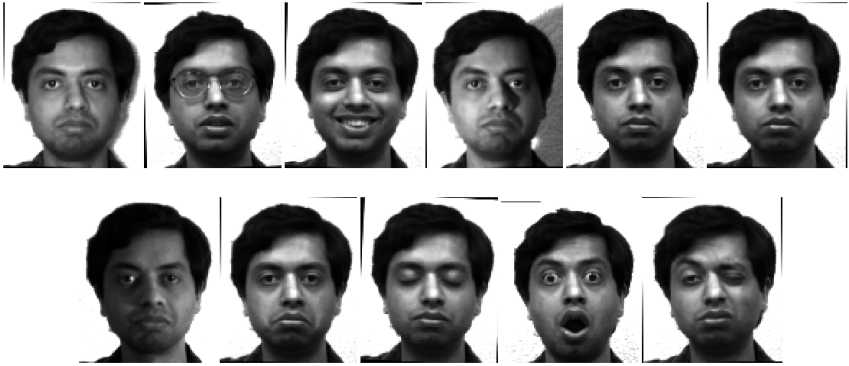
1. Một số ứng dụng

Ung dụng đầu tiên của PCA chính là việc giảm chiều dữ liệu, giúp việc lưu trữ và tính toán được thuận tiện hơn. Thực tế cho thấy, nhiều khi làm việc trên dữ liệu đã được giảm chiều mang lại kết quả tốt hơn so với dữ liệu gốc. Thứ nhất, có the phần dữ liệu mang thông tin nhỏ bi lược đi chính là phần gây nhiễu, những thông tin quan trọng hơn đã được giữ lại. Thứ hai, số điểm dữ liệu nhiều khi ít hơn số chiều dữ liệu. Khi có quá ít dữ liệu và số chiều dữ liệu quá lớn, quá khớp rất dễ xảy ra. Việc giảm chiều dữ liệu phần nào giúp khắc phục hiện tượng này.

Dưới đây là hai ví dụ về ứng dụng của PCA trong bài toán phân loại khuôn mặt và dò điểm bất thường.

1. Khuôn mặt riêng

*Khuôn mặt riêng* (eigenface) từng là một trong những kỹ thuật phổ biến trong bài toán nhận dạng khuôn mặt. Y tưởng của khuôn mặt riêng là đi tìm một không



**Hình 12.5.** Ví dụ ve ảnh của một người trong Yale Face Database.

gian có số chiều nhỏ hơn để mô tả mỗi khuôn mặt, từ đó sử dụng vector trong không gian thấp chiều này như vector đặc trưng cho bộ phân loại. Điều đáng nói là một bức ảnh khuôn mặt có kích thước khoảng 200 X 200 sẽ có số chiều là 40k - một số rất lớn, trong khi đó, vector đặc trưng thường chỉ có số chiều bằng vài trăm hoặc vài nghìn. Khuôn mặt riêng thực ra chính là PCA. Các khuôn mặt riêng chính là các vector riêng ứng với những tri riêng lớn nhất của ma trận hiệp phương sai.

Trong phần này, chúng ta làm một thí nghiệm nhỏ trên *cơ sở dữ liệu khuôn mặt Yale* [(https://goo.gl/LNg8LS)](https://goo.gl/LNg8LS). Các bức ảnh trong thí nghiệm này đã được căn chỉnh cho cùng với kích thước và khuôn mặt nằm trọn vẹn trong một hình chữ nhật có kích thước 116 X 98 điểm ảnh. Có tất cả 15 người khác nhau, mỗi người có 11 bức ảnh được chụp ở các điều kiện ánh sáng và cảm xúc khác nhau, bao gồm ’centerlight’, 'glasses', 'happy', 'leftlight', 'noglasses', 'normal' , 'rightlight','sad', 'sleepy', 'surprised', và 'wink'. Hình [12.5](#bookmark1380) minh hoạ các bức ảnh của người có id là 10.

Ta thấy rằng số chiều dữ liệu 116 X 98 = 11368 là một số khá lớn. Tuy nhiên, vì chỉ có tổng cộng 15 X 11 = 165 bức ảnh nên ta có the nén các bức ảnh này về dữ liệu mới có chiều nhỏ hơn 165. Trong ví dụ này, chúng ta chọn *K* = 100.

Dưới đây là đoạn code thực hiện PCA cho toàn bộ dữ liệu. ở đây, A trong sklearn được sử dụng:

**import** numpy as np

**from** scipy **import** misc *# for loading image*

np.random.seed(1)

* *filename structure*

path = 'unpadded/' *# path to the database*

ids = **range**(1, 16) *#15 persons*

states = ['centerlight', 'glasses', 'happy', 'leftlight',

'noglasses', 'normal', 'rightlight','sad',

'sleepy', 'surprised', 'wink' ]

prefix = 'subject'

surfix = '.pgm'

* *data dimension*

h, w, K = 116, 98, 100 *# hight, weight, new dim*

D = h \* w

N = **len**(states)\*15

* *collect all data*

X = np.zeros((D, N))

cnt = 0

**for** person\_id **in range**(1, 16):

**for** state **in** states:

fn = path + prefix + **str**(person\_id).zfill(2) + '.' + state + surfix X[:, cnt] = misc.imread(fn).reshape(D)

cnt += 1

* *Doing PCA, note that each row is a datapoint*

**from** sklearn.decomposition **import** PCA

pca = PCA(n\_components=K) *# K = 100*

pca.fit(X.T)

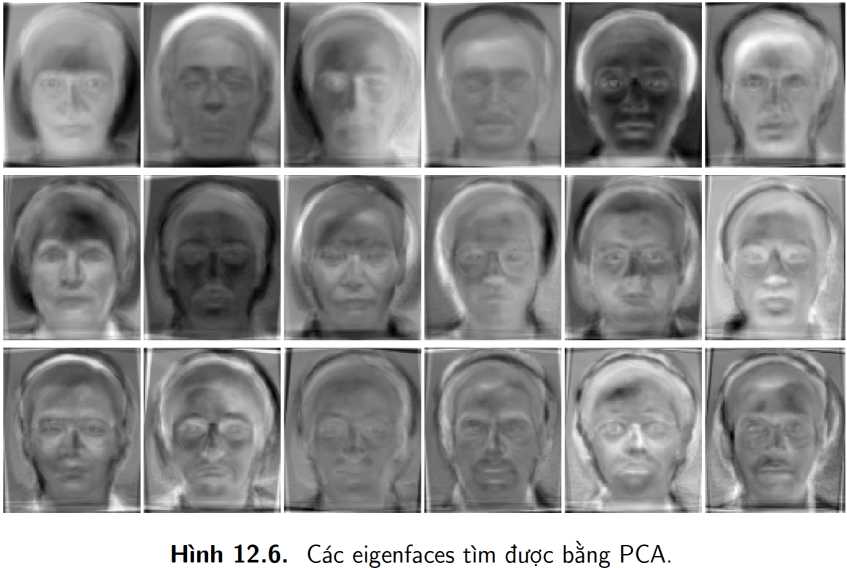
* *projection matrix*

U = pca.components\_.T

Trong dòng pca = PCA(n\_components=K), nếu n\_components là một số thực trong khoảng (0; 1), PCA sẽ thực hiện việc tìm *K* dựa trên biểu thức [(12.23)](#bookmark1291).

Hình 12.6 biểu diễn 18 vector riêng đầu tiên (18 cột đầu tiên của **U**k) tìm được bằng PCA. Các vector đã được reshape về cùng kích thước như các bức ảnh gốc. Nhận thấy các vector thu được ít nhiều mang thông tin của mặt người. Thực tế, một khuôn mặt gốc sẽ được xấp xỉ như tổng có trọng số của các khuôn mặt này. Vì các vector riêng này đóng vai trò như cơ sở của không gian mới với ít chiều hơn, chúng còn được gọi là *khuôn mặt riêng* hoặc *khuôn mặt chính.* Từ *chính* được dùng vì nó đi kèm với van cảnh của *phân tích thành phần chính.*

Để xem mức độ hiệu quả của phương pháp này, chúng ta minh hoạ các bức ảnh gốc và các bức ảnh được xấp xỉ bằng PCA như trên Hình [12.7](#bookmark1386). Các khuôn mặt nhận được vẫn mang khá đầy đủ thông tin của các khuôn mặt gốc. Đáng chú ý hơn, các khuôn mặt trong hàng dưới được suy ra từ một vector 100 chiều, so với 11368 chiều như ở hàng trên.

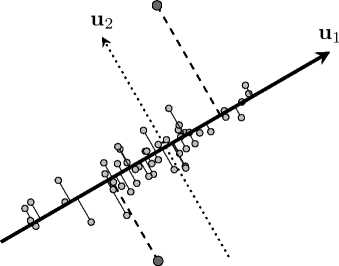




**Hình 12.7.** Hàng trên: các ảnh gốc. Hàng dưới: các ảnh được tái tạo dòng khuôn mặt riêng. Ảnh ở hàng dưới có nhiễu nhưng vẫn mang những đặc điểm riêng mà mắt người có thê phân biệt được.

1. Dò tìm điểm bất thường

Ngoài các ứng dụng về nén và phân loại, PCA còn được sử dụng trong nhiều lĩnh vực khác. *Dò tìm điểm bất thường* (abnormal detection hoặc outlier detection) là một trong số đó.

**Hình 12.8.** PCA cho bài toán dò tìm diem bat thường. Giả sử các sự kiện bình thường chiếm da số và nằm gan một không gian con nào dó. Khi dó, nếu làm PCA trên toàn bộ dữ liệu, không gian con thu dược gan vói không gian con của tập các sự kiện bình thường. Lúc này, các diêm hình tròn to dậm hon có thê dược coi là các sự kiện bat thường vì chúng nằm xa không gian con chính.

Ý tưởng cơ bản là giả sử tồn tại một không gian con mà các sự kiện bình thường nằm gần trong khi các sự kiện bất thường nằm xa không gian con đó. Hơn nữa, số sự kiện bất thường có một tỉ lệ nhỏ. Như vậy, PCA có thể được sử dụng trên toàn bộ dữ liệu để tìm ra các thành phần chính, từ đó suy ra không gian con mà các điểm bình thường nằm gần. Việc xác đinh một điểm là bình thường hay bất thường được xác đinh bằng cách đo khoảng cách từ điểm đó tới không gian con tìm được. Hình 12.8 minh hoạ cho việc xác đinh các sự kiện bất thường bằng PCA.

1. Thảo luận

* PCA là phương pháp giảm chiều dữ liệu dựa trên việc tối đa lượng thông tin được giữ lại. Lượng thông tin được giữ lại được đo bằng tổng các phương sai trên mỗi thành phần của dữ liệu. Lượng dữ liệu sẽ được giữ lại nhiều nhất khi các chiều dữ liệu còn lại tương ứng với các vector riêng của tri riêng lớn nhất của ma trận hiệp phương sai.
* Với các bài toán quy mô lớn, đôi khi việc tính toán trên toàn bộ dữ liệu là không khả thi vì vấn đề bộ nhớ. Giải pháp là thực hiện PCA lần đầu trên một tập con dữ liệu vừa với bộ nhớ, sau đó lấy một tập con khác để từ từ *(incrementally*) cập nhật nghiệm của PCA tới khi hội tụ. Ý tưởng này khá giống với mini-batch gradient descent, và được gọi là incremental PCA.

1. BÀI TẬP CUỐI CHƯƠNG

**Bài 1.** Động lực chính để giảm chiều dữ liệu là gì? Những hạn chế chính của các kỹ thuật giảm chiều là gì?

**Bài 2.** Lời nguyền chiều dữ liệu là gì?

**Bài 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?

**Bài 4.** Liệu PCA có thể được sử dụng để giảm chiều của tập dữ liệu có tính phi tuyến cao không?

**Bài 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?

**Bài 6.** Trong những trường hợp nào ta nên chọn kỹ thuật PCA thông thường, PCA Tang dần, PCA Ngẫu nhiên, PCA Hạt nhân?

**Bài 7.** Ta thực hiện đánh giá chất lượng của một thuật toán giảm chiều trên một tập dữ liệu như thế nào?

**Bài 8.** Liệu việc sử dụng liên tiếp hai thuật toán giảm chiều khác nhau có ý nghĩa không?

**Bài 9.** Phân cụm là gì? Kể tên một số thuật toán phân cụm?

**Bài 10.** Liệt kê một vài ứng dụng chính của các thuật toán phân cụm?

**Bài 12.** Mô tả hai kỹ thuật chọn số cụm phù hợp khi sử dụng K-điểm trung bình.

**Bài 13.** Lan truyền nhãn là gì? Tại sao ta cần lập trình nó và làm điều đó như thế nào?

**Bài 14.** Điểm khác biệt giữa phát hiện bất thường và phát hiện tính mới là gì?

**Bài 15.** Mô hình hỗn hợp Gauss là gì? Sử dụng nó trong tác vụ nào?

**Bài 16.** Trình bày hai kỹ thuật tìm kiếm số cụm phù hợp khi sử dụng mô hình hỗn hợp Gauss?

**Bài 17.** Hãy huấn luyện một mô hình hỗn hợp Gauss trên tập dữ liệu khuôn mặt Olivetti.

Chương IV

HỆ THỐNG GỢI ý

Chương này giúp sinh viên làm quen với các kỹ thuật xây dựng hệ thống gợi ý cơ bản.

***Bài 13***

Hệ thống gỢỈ ý dựa trên nội dung

1. Giới thiệu

Hệ thống gợi ý là một mảng khá rộng của machine learning và có xuất hiện sau phân loại hay hồi quy vì internet mới chỉ thực sự bùng nổ khoảng 10-15 nam gần đây. Có hai thực thể chính trong một hệ thống gợi ý là *người dùng* (user) và *sản phẩm* (item). Mục đích chính của các hệ thống gợi ý là dự đoán mức độ quan tâm của một người dùng tới một sản phẩm nào đó, qua đó có chiến lược gợi ý phù hợp.

1. Hiện tượng *đuôi dài*

Chúng ta cùng đi vào việc so sánh điểm khác nhau can bản giữa các cửa hàng thực và cửa hàng điện tử trên khía cạnh lựa chọn sản phẩm để quảng bá. ở đây, chúng ta tạm quên đi khía cạnh cảm giác thật chạm vào sản phẩm của các cửa hàng thực và tập trung vào phần làm thế nào để quảng bá đúng sản phẩm tới khách hàng.

Chúng ta đã biết *Nguyên lý Pareto* (quy tắc 20/80) [(https://goo.gl/NujWjH)](https://goo.gl/NujWjH): *phần lớn kết quả được gây ra bởi phần nhỏ nguyên nhãn.* Phần lớn số từ sử dụng hàng ngày chỉ là một phần nhỏ trong từ điển. Phần lớn của cải được sở hữu bởi phần nhỏ số người. Trong hương mại, những sản phẩm bán chạy nhất chiếm phần nhỏ trên tổng số sản phẩm.

Các cửa hàng thực thường có hai khu vực: khu trưng bày và kho. Nguyên tắc dễ thấy để đạt doanh thu cao là trưng ra các sản phẩm phổ biến ở những nơi dễ thấy nhất và cất những sản phẩm ít phổ biến hơn trong kho. Cách làm này có một hạn chế rõ rệt: những sản phẩm được trưng ra mang tính phổ biến nhưng chưa chắc đã phù hợp với nhu cầu của một khách hàng cụ thể. Một cửa hàng có thể có món hàng một người đang tìm kiếm nhưng không bán được vì khách hàng đó không tìm thấy sản phẩm. Điều này dẫn đến việc khách hàng không tiếp cận được sản phẩm ngay cả khi chúng đã được trưng ra. Ngoài ra, vì không gian có hạn, cửa hàng không thể trưng ra tất cả các sản phẩm mà mỗi loại chỉ đưa ra một số lượng nhỏ. ở đây, phần lớn doanh thu (80%) đến từ phần nhỏ số sản phẩm phổ biến nhất (20%). Nếu sắp xếp các sản phẩm của cửa hàng theo doanh số từ cao đến thấp, ta sẽ nhận thấy có thể phần nhỏ các sản phẩm tạo ra phần lớn doanh số. Và một danh sách dài phía sau chỉ đóng góp một lượng nhỏ. Hiện tượng này còn được gọi là *đuôi dài* (long tail phenomenon).

Với các cửa hàng điện tử, nhược điểm trên hoàn toàn có thể tránh được vì gian trưng bày của các cửa hàng điện tử gần như là vô tận, mọi sản phẩm đều có the được trưng ra. Hơn nữa, việc sắp xếp online là linh hoạt, tiện lợi với chi phí chuyển đổi gần như bằng không khiến việc mang đúng sản phẩm tới khách hàng trở nên thuận tiện. Doanh thu vì thế có thể được tang lên.

1. Hai nhóm thuật toán trong hệ thống gợi ý

Các thuật toán trong hệ thống gợi ý được chia thành hai nhóm lớn:

1. *Hệ thống dựa trên nội dung*: Gợi ý dựa trên đặc tính của sản phẩm. Ví dụ, hệ thống nên gợi ý các bộ phim hình sự tới những người thích xem phim “Cảnh sát hình sự” hay “Người phán xử”. Cách tiếp cận này yêu cầu sắp xếp các sản phẩm vào từng nhóm hoặc đi tìm các đặc trưng của từng sản phẩm. Tuy nhiên, có những sản phẩm không có rơi vào một nhóm cụ the và việc xác đinh nhóm hoặc đặc trưng của từng sản phẩm đôi khi bất khả thi.
2. *Lọc cộng tác* (collaborative filtering): Hệ thống gợi ý các sản phẩm dựa trên sự tương quan giữa người dùng và/hoặc sản phẩm. ở nhóm này, một sản phẩm được gợi ý tới một người dùng dựa trên những người dùng có sở thích tương tự hoặc những sản phẩm tương ựu. Ví dụ, ba người dùng *A, B, C* đều thích các bài hát của Noo Phước Thinh. Ngoài ra, hệ thống biết rằng người dùng *B, C* cũng thích các bài hát của Bích Phương nhưng chưa có thông tin về việc liệu người dùng *A* có thích ca sĩ này hay không. Dựa trên thông tin của những người dùng tương tự là *B và C,* hệ thống có thể dự đoán rằng *A* cũng thích Bích Phương và gợi ý các bài hát của ca sĩ này tới A.

Trong bài này, chúng ta sẽ làm quen với nhóm thuật toán thứ nhất. Nhóm thuật toán thứ hai, lọc cộng tác, sẽ được trình bày trong các bài tiếp theo.

1. Ma trận tiện ích

Có hai thực thể chính trong các hệ thống gợi ý là *người dùng* và *sản phẩm.* Mỗi người dùng có mức quan tâm tới từng sản phẩm khác nhau. Thông tin về mức

|  | A | B | C | D | E | F |
| --- | --- | --- | --- | --- | --- | --- |
| Mưa nửa đêm | 5 | 5 | 0 | 0 | 1 | ? |
| Cỏ úa | 5 | ? | ? | 0 | ? | ? |
| Vùng lá me bay | ? | 4 | 1 | ? | ? | 1 |
| Con cò bé bé | 1 | 1 | 4 | 4 | 4 | ? |
| Em yêu trưòng em | 1 | 0 | 5 | ? | ? | ? |

**Hình 13.1.** Víd ụ ve ma trận tiện ích vói hệ thống gợi ý bài hát. Các bài hát được người dòng đánh giá theo mức độ từ 0 đến 5 sao. Các dấu '?' nen màu xám ứng vói việc dữ liệu còn thiếu. Hệ thống gợi ý can dự đoán các giá trị này.

độ quan tâm của một người dùng tới một sản phẩm có thể được thu thập thông qua một hệ thống đánh giá (review và rating), qua việc người dùng đã click vào thông tin của sản phẩm hoặc qua thời lượng người dùng xem thông tin của một sản phẩm. Các ví dụ trong phần này đều dựa trên hệ thống đánh giá sản phẩm.

1. Ma trận tiện ích

Với một hệ thống đánh giá sản phẩm, mức độ quan tâm của một người dùng tới một sản phẩm được đo bằng số sao trên tổng số sao, chẳng hạn nam sao. Tập hợp tất cả các đánh giá *ở* dạng số, bao gồm cả những giá tri cần được dự đoán, tạo nên một ma trận gọi là *ma trận tiện ích* (utility matrix). Xét ví dụ trong Hình 13.1, có sáu người dùng *A, B, C, D, E, F* và nam bài hát. Các ô đã được đánh số thể hiện việc một người dùng đã đánh giá một bài hát từ 0 (không thích) đến 5 (rất thích). Các ô có dấu ’?’ tương ứng với các ô chưa có dữ liệu. Công việc của một hệ thống gợi ý là dự đoán giá tri tại các ô màu xám này, từ đó đưa ra gợi ý cho người dùng. Vì vậy, bài toán hệ thống gợi ý đôi khi được coi là bài toán *hoàn thiện ma trận* (matrix completion).

Nhận thấy có hai thể loại nhạc khác nhau: ba bài đầu là nhạc bolero và hai bài sau là nhạc thiếu nhi. Từ dữ liệu này, ta cũng có thể đoán được rằng *A, B* thích the loại nhạc Bolero; trong khi *C, D, E, F* thích nhạc thiếu nhi. Từ đó, một hệ thống tốt nên gợi ý “Cỏ úa” cho *B*; “Vùng lá me bay” cho A; “Em yêu trường em” cho *D, E, F.* Giả sử chỉ có hai the loại nhạc này, khi có một bài hát mới, ta cần phân loại rồi đưa ra gợi ý với từng người dùng.

Thông thường, có rất nhiều người dùng và sản phẩm trong hệ thống nhưng mỗi người dùng chỉ đánh giá một lượng nhỏ các sản phẩm, thậm chí có những người dùng không đánh giá sản phẩm nào. Vì vậy, lượng ô màu xám của ma trận tiện ích thường rất lớn so với lượng ô màu trắng đã biết.

Rõ ràng, càng nhiều ô được điền thì độ chính xác của hệ thống sẽ càng được cải thiện. Vì vậy, các hệ thống luôn khuyến khích người dùng bày tỏ sự quan tâm của họ tới các sản phẩm thông qua việc đánh giá các sản phẩm đó. Việc đánh giá không những giúp người dùng khác biết được chất lượng của sản phẩm mà còn giúp hệ thống biết được sở thích của người dùng, qua đó có chính sách quảng cáo hợp lý.

1. Xây dựng ma trận tiện ích

Không có ma trận tiện ích, hệ thống gần như không thể gợi ý được sản phẩm tới người dùng. Vì vậy, việc xây dựng ma trận tiện ích là tối quan trọng trong các hệ thống gợi ý. Tuy nhiên, việc xây dựng ma trận này thường gặp nhiều khó khan. Có hai hướng tiếp cận phổ biến để xác đinh giá tri đánh giá cho mỗi cặp (người dùng, sản phẩm) trong ma trận tiện ích:

1. Khuyến khích người dùng đánh giá sản phẩm. Amazon luôn khuyến khích người dùng đánh giá các sản phẩm bằng cách gửi mail nhắc nhở nhiều lần. Tuy nhiên, cách tiếp cận này cũng có một vài hạn chế. Các đánh giá có thể thiên lệch bởi những người sẵn sàng đáng giá.
2. Hướng tiếp cận thứ hai là dựa trên hành vi của người dùng. Nếu một người dùng mua một sản phẩm trên Amazon, xem một clip trên Youtube nhiều lần hay đọc một bài báo, có thể khẳng đinh người dùng này có xu hướng thích các sản phẩm đó. Facebook cũng dựa trên việc bạn *like* những nội dung nào để hiển thi trên *newsfeed* những nội dung liên quan. Bạn càng đam mê Facebook, Facebook càng được hưởng lợi. Với cách làm này, ta có thể xây dựng được một ma trận với các thành phần là 1 và 0, với 1 the hiện người dùng thích sản phẩm, 0 thể hiện chưa có thông tin. Trong trường hợp này, 0 không có nghĩa là thấp hơn 1, nó chỉ có nghĩa là người dùng chưa cung cấp thông tin. Chúng ta cũng có thể xây dựng ma trận với các giá tri cao hơn 1 thông qua thời gian hoặc số lượt mà người dùng xem một sản phẩm nào đó. Ngoài ra, đôi khi nút *dislike* cũng mang lại những lợi ích nhất đinh cho hệ thống, lúc này có the gán giá tri tương ứng bằng —1.
3. Hệ thống dựa trên nội dung
4. Xây dựng thông tin sản phẩm

Trong các hệ thống dựa trên nội dung, chúng ta cần xây dựng thông tin cho mỗi sản phẩm. Thông tin này được biểu diễn dưới dạng toán học là một vector đặc trưng. Trong những trường hợp đơn giản, vector này được trực tiếp trích xuất từ sản phẩm. Ví dụ, thông tin của một bài hát có thể được xác đinh bởi:

1. *Ca sĩ.* Cùng là bài “Thành phố buồn” nhưng có người thích bản của Đan Nguyên, có người lại thích bản của Đàm Vĩnh Hưng.
2. *Nhạc sĩ sáng tác.* Cùng là nhạc trẻ nhưng có người thích Phan Mạnh Quỳnh, người khác lại thích MTP.
3. *Năm sáng tác.* Một số người thích nhạc xưa cũ hơn nhạc hiện đại.
4. *Thể loại.* Quan họ và Bolero sẽ có thể thu hút những nhóm người khác nhau.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | vector đặc trưng |
| Mưa nửa đêm | 5 | 5 | 0 | 0 | 1 | ? | **X**1 = [0.99, 0.02]t |
| Cỏ úa | 5 | ? | ? | 0 | ? | ? | **X**2 = [0.91, 0.11]t |
| Vùng lá me bay | ? | 4 | 1 | ? | ? | 1 | **X**3 = [0.95, 0.05]t |
| Con cò bé bé | 1 | 1 | 4 | 4 | 4 | ? | **X**4 = [0.01, 0.99]t |
| Em yêu trưòng em | 1 | 0 | 5 | ? | ? | ? | **X**5 = [0.03, 0.98]t |
| Mô hình ngưòi dùng | 01 | Ớ2 | Ớ3 | Ớ4 | #5 | 06 | **tham số cần tìm** |

**Hình 13.2.** Giả sử vector đặc trưng cho mỗi sản phẩm đã biết trưóc, được cho trong cột cuối còng. Vói mỗi người dòng, chúng ta can tìm một mô hình ớu tưong ứng.

Trong ví dụ trong Hình 13.1, chúng ta đơn giản hoá bài toán bằng việc xây dựng một vector đặc trưng hai chiều cho mỗi bài hát: chiều thứ nhất là mức độ *Bolero,* chiều thứ hai là mức độ *Thiếu nhi* của bài hát đó. Giả sử ta đã xây dựng được vector đặc trưng cho mỗi bài hát là **x**1, **x**2, **x**3, **x**4, **x**5 như trong Hình 13.2. Tương tự, hành vi của mỗi người dùng cũng có thể được mô hình hoá dưới dạng tập các tham số ớ. Dữ liệu huấn luyện để xây dựng mỗi mô hình ớu là các cặp (thông tin sản phẩm, đánh giá) tương ứng với các sản phẩm người dùng đó đã đánh giá. Việc điền giá tri còn thiếu trong ma trận tiện ích chính là việc dự đoán mức độ quan tâm khi áp dụng mô hình ớu. Đầu ra này có thể được viết dưới dạng hàm số *f* (ớu,***x****i).* Việc lựa chọn dạng của *f* (ớu, ***x****i)* tuỳ thuộc vào mỗi bài toán. Trong bài này, chúng ta sẽ quan tâm tới dạng đơn giản nhất - dạng tuyến tính.

1. Xây dựng hàm mất mát

Đặt số lượng người dùng là *N*, số lượng sản phẩm là *M*; ma trận thông tin sản phẩm **X** = [**x**1, **x**2,... , **x**M] 2 *RdxM* và ma trận tiện ích là **Y** 2 RM*xN*. Thành phần ở hàng thứ *m,* cột thứ *n* của **Y** là mức độ quan tâm (ở đây là số sao đã đánh giá) của người dùng thứ *n* lên sản phẩm thứ *m* mà hệ thống đã thu thập được. Ma trận **Y** bi khuyết rất nhiều thành phần tương ứng với các giá tri cần dự đoán. Thêm nữa, gọi **R** là ma trận thể hiện việc một người dùng đã đánh giá một sản phẩm hay chưa. Cụ thể, *rmn* bằng một nếu sản phẩm thứ *m* đã được đánh giá bởi người dùng thứ n, bằng không trong trường hợp ngược lại.

***Mô hình tuyến tính***

Giả sử ta có thể tìm được một mô hình cho mỗi người dùng, được minh hoạ bởi một vector cột hệ số **w**n 2 và hệ số điều chỉnh *bn* sao cho mức độ quan tâm của một người dùng tới một sản phẩm tính được bằng một hàm tuyến tính:

*ymn =* **w**T ***x****m* + *bn* (13.1)

Xét người dùng thứ n, nếu coi tập huấn luyện là tập hợp các thành phần đã biết của **y**n (cột thứ *n* của ma trận **Y**), ta có thể xây dựng hàm mất mát tương tựnhư hồi quy ridge (hồi quy tuyến tính với kiểm soát *l2)* như sau:

Ln(Wn, *bn)* = Ỵ4 *(****x****^****W****n* + bn *- ymn))* + -7- l|Wn||2

(13.2)

2sn -I 2 sn

*m:rmn = 1*

trong đó, thành phần thứ hai đóng vai trò kiểm soát và A là một tham số dương; *sn* là số lượng các sản phẩm mà người dùng thứ *n* đã đánh giá, là tổng các phần tử trên cột thứ *n* của ma trận **R**, tức *sn* 1 rmn. Chú ý rằng cơ chế kiểm

soát thường không được áp dụng lên hệ số điều chỉnh *bn.*

Vì biểu thức hàm mất mát [(13.2)](#bookmark1460) chỉ phụ thuộc vào các sản phẩm đã được đánh giá, ta có thể rút gọn nó bằng cách đặt **y**n 2 là vector con của **y**n, được xây dựng bằng cách trích các thành phần đã biết ở cột thứ *n* của **Y**. Đồng thời, đặt ***X****n 2* là ma trận con của ma trận đặc trưng **X**, thu được bằng cách trích các cột tương ứng với những sản phẩm đã được đánh giá bởi người dùng thứ n. Biểu thức hàm mất mát của mô hình cho người dùng thứ *n* được viết gọn thành:

Ln(**w**n,bn) = -1-*\\****Xw****n* + *bn****e****n* **- y**n||2 + -^ **W**n 112

(13.3)

trong đó, **e**n là vector cột với tất cả các thành phần bằng một. Đây chính là hàm mất mát của hồi quy ridge. Cặp nghiệm **w**n,bn có thể được tìm thông qua các thuật toán gradient descent. Trong bài này, chúng ta sẽ trực tiếp sử dụng class Ridge trong thư viện sklearn.linear\_model. Một điểm đáng lưu ý ở đây là **w**n chỉ được xác đinh nếu người dùng thứ *n* đã đánh giá ít nhất một sản phẩm.

1. Ví dụ về hàm mất mát cho người dùng *E*

Quay trở lại ví dụ trong Hình 13.2, ma trận đặc trưng cho các sản phẩm (mỗi cột tương ứng với một sản phẩm) là

0.99 0.91 0.95 0.01 0.03

(13.4)

0.02 0.11 0.05 0.99 0.98

Xét trường hợp của người dùng *E* với *n* = 5, **y**5 = [1, ?, ?, 4, ?]T. Từ đó, vector nhi phân **r**5 = [1, 0, 0,1, 0]T. Vì E mới chỉ đánh giá sản phẩm thứ nhất và thứ tư nên s5 = 2. Hơn nữa,

| 0.99 0.01 |  | 1 |  | 1 |
| --- | --- | --- | --- | --- |
| 0.02 0.99 | , y5 = | 4 | , **e**5 = | 1 |

**X** 5 =

(13.5)

Khi đó, hàm mất mát cho hệ số tương ứng với người dùng E là:

*c5(****w****5,b5)* = 4

0.99 0.02

0.01 0.99

**w**5 + b5

11 2 Ă

4 +4 llwsH2

2

(13.6)

Chúng ta sẽ áp dụng những phân tích trên đây để đi tìm nghiệm cho một bài toán gần với thực tế.

1. Bài toán MovieLens 100k
2. Cơ sở dữ liệu MovieLens 100k

Bộ cơ sở dữ liệu MovieLens 100k [(https://goo.gl/BzHgtq)](https://goo.gl/BzHgtq) được công bố nam 1998 bởi GroupLens [(https://grouplens.org)](https://grouplens.org). Bộ cơ sở dữ liệu này bao gồm 100,000 (100k) đánh giá từ 943 người dùng cho 1682 bộ phim. Các bạn cũng có thể tìm thấy các bộ cơ sở dữ liệu tương tự với khoảng 1M, 10M, 20M đánh giá.

Bộ cơ sở dự liệu này bao gồm nhiều file, chúng ta cần quan tâm các file sau:

* u.data: Chứa toàn bộ các đánh giá của 943 người dùng cho 1682 bộ phim. Mỗi người dùng đánh giá ít nhất 20 bộ phim. Thông tin về thời điểm đánh giá cũng được cho nhưng chúng ta không sử dụng trong ví dụ này.
* ua.base, ua.test, ub.base, ub.test: Là hai cách chia toàn bộ dữ liệu ra thành hai tập con: tập huấn luyện và tập kiểm tra. Chúng ta sẽ thực hành trên ua.base và ua.test. Bạn đọc có the thử với cách chia dữ liệu còn lại.
* u.user: Chứa thông tin về người dùng, bao gồm: id, tuổi, giới tính, nghề nghiệp, mã vùng (zipcode). Những thông tin này có thể ảnh hưởng tới sở thích của người dùng; tuy nhiên, chúng ta chỉ sử dụng *id* để xác đinh người dùng khác nhau.
* u.genre: Chứa tên của 19 thể loại phim, gồm: unknown, Action, Adventure, Animation, Children’s, Comedy, Crime, Documentary, Drama, Fantasy, Film­Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western,
* u.item: Thông tin về mỗi bộ phim. Một vài dòng đầu tiên của file:

1|Toy Story (1995)|01-Jan-1995||<http://us.imdb.com/M/title-exact?Toy%20Story> %20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0

2|GoldenEye (1995)|01-Jan-1995||<http://us.imdb.com/M/title-exact?GoldenEye> %20(1995)|0|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0

3|Four Rooms (1995)|01-Jan-1995||<http://us.imdb.com/M/title-exact?Four%20> Rooms%20(1995)|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|1|0|0

4|Get Shorty (1995)|01-Jan-1995||<http://us.imdb.com/M/title-exact?Get%20>

Shorty%20(1995)|0|1|0|0|0|1|0|0|1|0|0|0|0|0|0|0|0|0|0

Trong mỗi dòng, chúng ta sẽ thấy id của phim, tên phim, ngày phát hành, đường dẫn và các số nhi phân 0, 1 thể hiện bộ phim thuộc các thể loại nào trong 19 thể loại đã cho. Một bộ phim có thể thuộc nhiều thể loại khác nhau. Thông tin về thể loại này sẽ được dùng để xây dựng thông tin sản phẩm.

Chúng ta sử dụng thư viện pandas [(http://pandas.pydata.org)](http://pandas.pydata.org) để đọc dữ liệu:

**from** future **import** print\_function

**import** numpy as np **import** pandas as pd

*# Reading user file:*

u\_cols = ['user\_id', 'age', 'sex', 'occupation', 'zip\_code']

users = pd.read\_csv('ml-100k/u.user', sep='|', names=u\_cols) n\_users = users.shape[0]

**print**('Number of users:', n\_users)

*#Reading ratings file:*

r\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp']

ratings\_base = pd.read\_csv('ml-100k/ua.base', sep='\t', names=r\_cols) ratings\_test = pd.read\_csv('ml-100k/ua.test', sep='\t', names=r\_cols)

rate\_train = ratings\_base.as\_matrix() rate\_test = ratings\_test.as\_matrix()

**print**('Number of traing rates:', rate\_train.shape[0])

**print**('Number of test rates:', rate\_test.shape[0])

Kết quả:

Number of users: 943

Number of traing rates: 90570

Number of test rates: 9430

Ta sẽ chỉ quan tâm tới 19 giá tri nhi phân ở cuối mỗi hàng để xây dựng thông tin sản phẩm.

X0 = items.as\_matrix()

X\_train\_counts = X0[:, -19:]

1. Xây dựng thông tin sản phẩm

Công việc quan trọng trong hệ thống gợi ý dựa trên nội dung là xây dựng vector đặc trưng cho mỗi sản phẩm. Trước hết, chúng ta cần lưu thông tin về các sản phẩm vào biến items:

*#Reading items file:*

i\_cols = ['movie id', 'movie title' ,'release date','video release date', '

IMDb URL', 'unknown', 'Action', 'Adventure', 'Animation', 'Children\'s',

'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', '

Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', ' Western']

items = pd.read\_csv('ml-100k/u.item', sep='|', names=i\_cols) n\_items = items.shape[0] **print**('Number of items:', n\_items)

Kết quả:

Number of items: 1682

Tiếp theo, chúng ta hiển thi một vài hàng đầu tiên của ma trận rate\_train:

**print**(rate\_train[:4, :])

Kết quả:

|  |  |  |  |
| --- | --- | --- | --- |
| [[ | 1 | 1 | 5 874965758] |
| [ | 1 | 2 | 3 876893171] |
| [ | 1 | 3 | 4 878542960] |
| [ | 1 | 4 | 3 876893119]] |

Hàng thứ nhất được hiểu là người dùng thứ nhất đánh giá bộ phim thứ nhất nam sao. Cột cuối cùng là thời điểm đánh giá, chúng ta sẽ bỏ qua thông số này.

Tiếp theo, chúng ta sẽ xây dựng vector đặc trưng cho mỗi sản phẩm dựa trên ma trận thể loại phim và đặc trưng TF-IDF [(https://goo.gl/bpDdQ8)](https://goo.gl/bpDdQ8) trong thư viện sklearn:

*#tfidf*

**from** sklearn.feature\_extraction.text **import** TfidfTransformer transformer = TfidfTransformer(smooth\_idf=True, norm ='l2')

X = transformer.fit\_transform(X\_train\_counts.tolist()).toarray()

Sau bước này, mỗi hàng của X tương ứng với vector đặc trưng của một bộ phim.

1. Xây dựng mô hình cho mỗi người dùng

Với mỗi người dùng, chúng ta cần đi tìm những bộ phim nào mà người dùng đó đã đánh giá, và giá tri của các đánh giá đó.

**def** get\_items\_rated\_by\_user(rate\_matrix, user\_id):

ri ff ff

*return (item\_ids, scores)*

fl fl fl

y = rate\_matrix[:,0] *# all users*

* *item indices rated by user\_id*
* *we need to +1 to user\_id since in the rate\_matrix, id starts from 1*
* *but id in python starts from 0*

ids = np.where(y == user\_id +1)[0]

item\_ids = rate\_matrix[ids, 1] scores = rate\_matrix[ids, 2] **return** (item\_ids, scores)

1 *# index starts from 0*

Bây giờ, ta có thể đi tìm vector trọng số của mỗi người dùng:

**from** sklearn.linear\_model **import** Ridge

**from** sklearn **import** linear\_model

d = X.shape[1] *# data dimension*

W = np.zeros((d, n\_users))

b = np.zeros(n\_users)

**for** n **in range**(n\_users):

ids, scores = get\_items\_rated\_by\_user(rate\_train, n) model = Ridge(alpha=0.01, fit\_intercept = True) Xhat = X[ids, :] model.fit(Xhat, scores)

W[:, n] = model.coef\_

b[n] = model.intercept\_

Sau khi tính được các hệ số W và b, mức độ quan tâm của mỗi người dùng tới một bộ phim được dự đoán bởi:

Yhat = X.dot(W) + b

Dưới đây là một ví dụ với người dùng có id bằng 10:

n = 10

np.set\_printoptions(precision=2) *# 2 digits after .* ids, scores = get\_items\_rated\_by\_user(rate\_test, n) **print**('Rated movies ids :', ids ) **print**('True ratings :', scores)

**print**('Predicted ratings:', Yhat[ids, n])

Kết quả:

Rated movies ids : [ 37 109 110 226 424 557 722 724 731 739]

True ratings : [3343435334]

Predicted ratings: [3.18 3.13 3.42 3.09 3.35 5.2 4.01 3.35 3.42 3.72]

1. Đánh giá mô hình

Để đánh giá mô hình tìm được, chúng ta sẽ sử dụng *căn bậc hai sai số trung bình bình phương* (root mean squared error, RMSE):

**def** evaluate(Yhat, rates, W, b):

se = cnt = 0

**for** n **in xrange**(n\_users):

ids, scores\_truth = get\_items\_rated\_by\_user(rates, n) scores\_pred = Yhat[ids, n] e = scores\_truth - scores pred se += (e\*e).**sum**(axis = 0) cnt += e.size **return** np.sqrt(se/cnt)

**print**('RMSE for training: %.2f' %evaluate(Yhat, rate\_train, W, b))

**print**('RMSE for test : %.2f' %evaluate(Yhat, rate\_test, W, b))

Kết quả:

RMSE for training: 0.91

RMSE for test : 1.27

Như vậy, với training set, sai số vào khoảng 0.91 (sao); với test set, sai số lớn hìn một chút, khoảng 1.27. Các kết quả này chưa thực sự tốt vì mô hình đã được đìn giản hoá quá nhiều. Kết quả tốt hìn có thể được thấy trong các bài tiếp theo về lọc cộng tác.

1. Thảo luận

* Hệ thống gợi ý dựa trên nội dung là một phưìng pháp gợi ý đìn giản. Đặc điểm của phưìng pháp này là việc xây dựng mô hình cho mỗi người dùng không phụ thuộc vào người dùng khác.
* Việc xây dựng mô hình cho mỗi người dùng có thể coi như bài toán hồi quy với dữ liệu huấn luyện là thông tin sản phẩm và đáng giá của người dùng đó về sản phẩm đó. Thông tin sản phẩm không phụ thuộc vào người dùng mà phụ thuộc vào các đặc điểm mô tả của sản phẩm.

Đọc thêm

1. *Recommendation Systems* - *Stanford InfoLab* [(https://goo.gl/P1pesC)](https://goo.gl/P1pesC).
2. *Recommendation systems - Machine Learning, Andrew Ng* [(https://goo.gl/](https://goo.gl/jdFvej) [jdFvej)](https://goo.gl/jdFvej).
3. *Content Based Recommendations - Stanford University* [(https://goo.gl/3wnbZ4)](https://goo.gl/3wnbZ4).

***B ài 14***

Lọc cộng tác lân cận

1. Giới thiệu

Trong hệ thống gợi ý dựa trên nội dung, chúng ta đã làm quen với một hệ thống gợi ý sản phẩm đơn giản dựa trên vector đặc trưng của mỗi sản phẩm. Đặc điểm của các hệ thống này là việc xây dựng mô hình cho mỗi người dùng không phụ thuộc vào các người dùng khác mà chỉ phụ thuộc vào thông tin sản phẩm. Việc làm này có lợi thế là tiết kiệm bộ nhớ và thời gian tính toán nhưng có hai nhược điểm cơ bản. Thứ nhất, việc xây dựng thông tin cho sản phẩm không phải lúc nào cũng thực hiện được. Thứ hai, khi xây dựng mô hình cho một người dùng, các hệ thống gợi ý theo nội dung không tận dụng được thông tin đã có từ những người dùng khác. Những thông tin này thường rất hữu ích vì hành vi mua hàng của người dùng thường được chia thành một vài nhóm cơ bản. Nếu biết hành vi mua hàng của một vài người dùng trong nhóm, hệ thống nên có khả nang dự đoán hành vi của những người dùng còn lại trong nhóm đó.

Những nhược điểm này có thể được giải quyết bằng một kỹ thuật có tên là *lọc cộng tấc* (collaborative filtering - CF). Trong bài này, chúng ta cùng làm quen với một phương pháp CF có tên là lọc *cộng tấc dựa trên lân cận* (neighborhood-based collaborative filtering - NBCF). Bài tiếp theo sẽ trình bày về một phương pháp CF khác có tên *lọc cộng tấc phân tích ma trận* (matrix factorization collaborative filtering). Nếu chỉ nói *lọc cộng tấc,* ta gầm hiểu rằng đó là *lọc cộng tấc dựa trên lân cận*.

Y tưởng của NBCF là xác đinh mức độ quan tâm của một người dùng tới một sản phẩm dựa trên những người dùng có hành vi tương tự. Việc xác đinh sự tương tự giữa những người dùng có thể được xác đinh thông qua mức độ quan tâm của họ tới các sản phẩm khác mà hệ thống đã biết. Ví dụ, *A* và *B* thích phim “Cảnh sát hình sự”, đều đã đánh giá bộ phim này nam sao. Ta đã biết thêm *A* thích “Người phán xử”, vậy nhiều khả nang *B* cũng thích bộ phim này.

Có hai câu hỏi chính khi xây dựng một hệ thống lọc cộng tác dựa trên lân cận:

1. Làm thế nào xác đinh được sự tương tự giữa hai người dùng?
2. Khi đã xác đinh được các người dùng có hành vi gần giống nhau, làm thế nào dự đoán được mức độ quan tâm của một người dùng lên một sản phẩm?

Việc xác đinh mức độ quan tâm của mỗi người dùng tới một sản phẩm dựa trên mức độ quan tâm của những người dùng tương tự tới sản phẩm đó còn được gọi là *lọc cộng tác người dùng* (user-user collaborative filtering). Có một hướng tiếp cận khác thường cho kết quả tốt hơn là *lọc cộng tác sản phẩm* (item-item collaborative filtering). Trong hướng tiếp cận này, thay vì xác đinh độ tương tự giữa các người dùng, hệ thống sẽ xác đinh độ tương tự giữa các sản phẩm. Từ đó, hệ thống gợi ý một sản phẩm tương tự những sản phẩm khác mà người dùng đó có mức độ quan tâm cao.

Cấu trúc của bài như sau: Mục [14.2](#bookmark1520) trình bày lọc cộng tác người dùng. Mục [14.3](#bookmark1539) nêu một số hạn chế của phương pháp này và cách khắc phục bằng lọc cộng tác sản phẩm. Kết quả của hai phương pháp này được trình bày qua ví dụ trên cơ sở dữ liệu MovieLens 100k trong Mục [14.4](#bookmark1551). Mục [14.5](#bookmark1569) thảo luận các ưu nhược điểm của NBCF.

1. Lọc cộng tác theo người dùng
2. Hàm số đo độ tương tự

Việc quan trọng nhất trong lọc cộng tác người dùng là xác đinh được *độ tương tự* (similarity) giữa hai người dùng. Giả sử thông tin duy nhất chúng ta có là ma trận tiện ích **Y**. Độ tương tự giữa hai người dùng sẽ được xác đinh dựa trên các cột tương ứng với họ trong ma trận này.

Xét ví dụ trong Hình [14.1](#bookmark1466). Giả sử có những người dùng từ *u0* đến *u6* và các sản phẩm từ i0 đến i4. Các số trong mỗi ô vuông thể hiện số sao mà mỗi người dùng đã đánh giá sản phẩm với giá tri cao hơn thể hiện mức quan tâm cao hơn. Các dấu hỏi chấm là các giá tri mà hệ thống cần tìm. Đặt mức độ tương tự của hai người dùng *ui,Uj* là *sim(ui,Uj*). Có thể nhận thấy *u0,u1* thích *i0,* i1,i2 hơn «3,i4. Trong khi đó *u2,u3,u4,u5,u6* thích *i3,i4* hơn i0,i1,i2. Vì vậy, một *hàm đo độ tương tự* (similarity function) tốt cần đảm bảo:

sim(u0,u1) > sim(u0,ui), 8i > 1, (14.1)

với giá tri cao hơn ứng với độ giống nhau cao hơn.

|  | *uo* | *u1* | U2 | U3 | u4 | U5 | u6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| io | 5 | 5 | 2 | 0 | 1 | ? | ? |
|  | 3 | ? | ? | 0 | ? | ? | ? |
| Ì2 | ? | 4 | 1 | ? | ? | 1 | 2 |
| Ì3 | 2 | 2 | 3 | 4 | 4 | ? | 4 |
| Ì4 | 2 | 0 | 4 | ? | ? | ? | 5 |

**Hình 14.1.** Ví d ụ về ma trận tiện ích dựa trên số sao người dòng đánh giá sản phàm. Nhận thay hành vi của u0 giống *U1* hơn *U2; U3; U4; U5; U6.* Từ đó có the dự đoán rằng u0 sẽ quan tâm tói i2 vì *U1* cũng quan tâm tói sản phàm này.

Để xác đinh mức độ quan tâm của u0 lên i2, chúng ta nên dựa trên hành vi của *U1* lên sản phẩm này. Vì đã biết *U1* thích *i2*, hệ thống có thể gợi ý *i2* tới u0.

Câu hỏi đặt ra là, hàm đo độ tương tự cần được xây dựng như thế nào? Đe đo độ tương tự giữa hai người dùng, cách thường làm là xây dựng vector đặc trưng cho mỗi người dùng rồi áp dụng một hàm có khả nang đo độ giống nhau giữa hai vector đó. ở đây, việc xây dựng vector đặc trưng khác với việc xây dựng thông tin sản phẩm như trong các hệ thống gợi ý dựa trên nội dung. Các vector đặc trưng này được xây dựng trực tiếp dựa trên ma trận tiện ích mà không dùng thêm thông tin bên ngoài. Với mỗi người dùng, thông tin duy nhất chúng ta biết là các đánh giá mà người dùng đó đã thực hiện, có thể tìm thấy trong cột tương ứng trong ma trận tiện ích. Tuy nhiên, khó khan là các cột này thường có nhiều giá tri bi khuyết (các dấu ‘?’ trong Hình [14.1)](#bookmark1466) vì mỗi người dùng thường chỉ đánh giá một lượng nhỏ các sản phẩm. Một cách khắc phục là điền các *ước lượng thô* (raw estimation) vào các ô '?’ sao cho việc này không ảnh hưởng nhiều tới độ tương tự giữa hai vector. Các giá tri ước lượng này chỉ phục vụ việc tính độ tương tự, không phải là kết quả cuối cùng hệ thống cần xác đinh.

Vậy mỗi dấu ‘ ?’ nên được thay bởi giá tri nào để hạn chế sai lệch khi ước lượng? Lựa chọn đầu tiên có thể nghĩ đến là thay các dấu ‘?’ bằng 0. Điều này không thực sự tốt vì giá tri 0 dễ bi nhầm với với mức độ quan tâm thấp nhất; và một người dùng chưa đánh giá một sản phẩm không có nghĩa là họ hoàn toàn không quan tâm tới sản phẩm đó. Một giá tri an toàn hơn là trung bình cộng của khoảng giá tri, ở đây là 2.5 trên hệ thống đánh giá nam sao. Tuy nhiên, giá tri này có nhược điểm đối với những người dùng dễ tính hoặc khó tính. Những người dùng dễ tính có thể đánh giá ba sao cho các sản phẩm họ không thích; ngược lại, những người dùng khó tính có the đánh giá ba sao cho những sản phẩm họ thích. Việc thay đồng loạt các phần tử khuyết bởi 2.5 trong trường hợp này chưa mang lại hiệu quả.

Một giá tri khả dĩ hơn cho việc này là ước lượng các phần tử khuyết bởi giá tri trung bình mà một người dùng đã đánh giá. Điều này giúp tránh việc một người dùng quá khó tính hoặc dễ tính. Các giá tri ước lượng này phụ thuộc vào từng người dùng. Quan sát ví dụ trong Hình [14.2](#bookmark1467).

Hàng cuối cùng trong Hình [14.2a](#bookmark1467) là trung bình các đánh giá của mỗi người dùng. Các giá tri cao tương ứng với những người dùng dễ tính và ngược lại. Khi đó, nếu tiếp tục trừ từ mỗi đánh giá đi giá tri trung bình này và thay các giá tri chưa biết bằng 0, ta sẽ được một *ma trận tiện ích chuẩn hoá* (normalized utility matrix) như trong Hình [14.2b](#bookmark1467). Việc làm này có một vài ưu điểm:

* Việc trừ mỗi giá tri đi trung bình cộng của cột tương ứng trong ma trận tiện ích khiến mỗi cột có cả những giá tri dương và âm. Những giá tri dương ứng với những sản phẩm được người dùng quan tâm hơn. Những ô có giá tri 0 tương ứng với việc người dùng chưa đánh giá sản phẩm tương ứng. Tâ cần đự đoán giá tri ở các ô này.
* Về mặt kỹ thuật, số chiều của ma trận tiện ích là rất lớn với hàng triệu người dùng và sản phẩm, việc lưu toàn bộ các giá tri này trong một ma trận sẽ yêu cầu bộ nhớ lớn. Vì số lượng đánh giá biết trước thường là một số rất nhỏ so với kích thước của ma trận tiện ích, sẽ tốt hơn nếu chúng ta lưu ma trận này dưới dạng một ma trận thưa, tức chỉ lưu các giá tri khác không và vi trí của chúng. Vì vậy, tốt hơn hết, các dấu ’?’ nên được thay bằng giá tri ’0’, tức chưa xác đinh liệu người dùng có thích sản phẩm hay không. Việc này không những tối ưu bộ nhớ mà việc tính toán ma trận tương tự về sau hiệu quả hơn. ở đây, phần tử ở hàng thứ *i,* cột thứ *j* của ma trận tương tự là độ tương tự giữa người dùng thứ i và thứ j.

Sau khi dữ liệu đã được chuẩn hoá, hàm tương tự thường được sử dụng là *tương tự cos* (cosine similarity):

cosine\_similarity(**u**i, **u**2) = cos(**u**i, **u**2) = -—1—— (14.2)

II u1 II2• II u2 II2

Trong đó **u**1;2 là các vector tương ứng với hai người dùng trong ma trận tiện ích chuẩn hoá. Có một hàm trong Python giúp cách tính hàm số này một cách hiệu quả, chúng ta sẽ thấy trong phần lập trình.

Mức độ tương tự của hai vector là một số thực trong đoạn [-1, 1]. Giá tri bằng 1 thể hiện hai vector hoàn toàn giống nhau. Hàm số cos của một góc bằng 1 xảy ra khi góc giữa hai vector bằng 0, tức hai vector có cùng phương và cùng hướng. Giá tri của hàm *cos* bằng -1 khi hai vector hoàn toàn trái ngược nhau, tức cùng phương nhưng khác hướng. Điều này có nghĩa là nếu hành vi của hai người dùng là hoàn toàn ngược nhau thì độ tương tự giữa họ là thấp nhất.

Ví dụ về tương tự cos của người dùng (đã được chuẩn hoá) trong Hình [14.2b](#bookmark1467) được cho trong Hình [14.2c](#bookmark1467). Ma trận tương tự **S** là một ma trận đối xứng vì cos

|  | «0 | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ì0 | 5 | 5 | 2 | 0 | 1 | ? | ? |
| i1 | 4 | ? | ? | 0 | ? | 2 | ? |
| Ì2 | ? | 4 | 1 | ? | ? | 1 | 1 |
| Ì3 | 2 | 2 | 3 | 4 | 4 | ? | 4 |
| Ì4 | 2 | 0 | 4 | ? | ? | ? | 5 |
|  | ị | ị | ị | ị | ị | ị | ị |
| «j | 3.25 | 2.75 | 2.5 | 1.33 | 2.5 | 1.5 | 3.33 |

|  | «0 | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ì0 | 1.75 | 2.25 | -0.5 | -1.33 | -1.5 | 0 | 0 |
| i1 | 0.75 | 0 | 0 | -1.33 | 0 | 0.5 | 0 |
| Ì2 | 0 | 1.25 | -1.5 | 0 | 0 | -0.5 | -2.33 |
| Ì3 | -1.25 | -0.75 | 0.5 | 2.67 | 1.5 | 0 | 0.67 |
| Ì4 | -1.25 | -2.75 | 1.5 | 0 | 0 | 0 | 1.67 |

a) Ma trận tiện ích ban đầu **Y** và trung bình độ quan tâm của người dùng

b) Ma trận tiện ích chuẩn hoá **Y**

|  | «0 | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| «0 | 1 | 0.83 | -0.58 | -0.79 | -0.82 | 0.2 | -0.38 |
| «1 | 0.83 | 1 | -0.87 | -0.40 | -0.55 | -0.23 | -0.71 |
| «2 | -0.58 | -0.87 | 1 | 0.27 | 0.32 | 0.47 | 0.96 |
| «3 | -0.79 | -0.40 | 0.27 | 1 | 0.87 | -0.29 | 0.18 |
| «4 | -0.82 | -0.55 | 0.32 | 0.87 | 1 | 0 | 0.16 |
| «5 | 0.2 | -0.23 | 0.47 | -0.29 | 0 | 1 | 0.56 |
| «6 | -0.38 | -0.71 | 0.96 | 0.18 | 0.16 | 0.56 | 1 |

c) Ma trận tương tự người dùng **S**

|  | «0 | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ì0 | 1.75 | 2.25 | -0.5 | -1.33 | -1.5 |  | -0.63 |
| i1 | 0.75 | \V8 | -0.17 | -1.33 | -1.33 | 0.5 | '0'05 |
| Ì2 |  | 1.25 | -1.5 | -1.84 | -1.78 | -0.5 | -2.33 |
| Ì3 | -1.25 | -0.75 | 0.5 | 2.67 | 1.5 | \ị\v | 0.67 |
| Ì4 | -1.25 | -2.75 | 1.5 | 'ị' |  | 1’59 | 1.67 |

d) Ma trận tiện ích chuẩn hoá sau hoàn thiện

Dự đoán độ quan tâm chuẩn hoá của «1 cho i1 vói

Người dùng đã đánh giá *i1* : {«0, «3, "5}

Độ tương tự tương ứng: {0.83, -0.40, -0.23}

*k* người dùng giống nhất: N(«1,i1) ={u0,us} với đánh giá chuẩ’n hoá {0.75, 0.5}

0.83\*0.75 + (-0.23)\*0.5

0.83+1-0.23|

0.48

| 2 | «0 | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ì0 | 5 | 5 | 2 | 0 | 1 |  | 2.70 |
| i1 | 4 |  | 2.33 | 0 | 1.67 | 2 |  |
| Ì2 |  | 4 | 1 | -0.5 | 0.71 | 1 | 1 |
| Ì3 | 2 | 2 | 3 | 4 | 4 | \\\\'' | 4 |
| Ì4 | 2 | 0 | 4 |  | 4.06 |  | 5 |

e) Ví dụ cách tính ô viền đậm trong d)

f) Ma trận tiện ích sau hoàn thiện

**Hình 14.2.** Ví dụ mô tả lọc cộng tác người dòng. a) Ma trận tiện ích ban đầu. b) Ma trận tiện ích đã được chuân hoá. c) Ma trận tương tự giữa người dòng. d) Dự đoán độ quan tâm (chuan hoá) còn thiếu. e) Ví dụ về cách dự đoán độ quan tâm chuan hoá của *U1* tói i1. f) Dự đoán các độ quan tâm còn thiếu.

là một hàm chẵ[n7](#bookmark6), và nếu *A* tương tự *B* thì điều ngược lại cũng đúng. Các ô trên đường chéo đều là cos của góc giữa một vector và chính nó, tức cos(0) = 1. Khi tính toán ở các bước sau, chúng ta không cần quan tâm tới các giá tri này. Tiếp tục quan sát các vector hàng tương ứng với *u0,u1,u2,* chúng ta sẽ thấy một vài điều thú vi:

* *u0* gần với *U1* và *U5* (độ tương tự là dương) hơn các người dùng còn lại. Việc độ tương tự cao giữa *u0* và *U1* là dễ hiểu vì cả hai đều có xu hướng quan tâm tới *i0, i1, i2* hơn các sản phẩm còn lại. Việc *u0 gần* với *U5* thoạt đầu có vẻ vô lý vì *u5* đánh giá thấp các sản phẩm mà u0 đánh giá cao (Hình [14.2a](#bookmark1467)); tuy nhiên khi nhìn vào ma trận tiện ích đã chuẩn hoá trong Hình [14.2b](#bookmark1467), ta thấy rằng điều này là hợp lý vì sản phẩm duy nhất mà cả hai người dùng này đã cung cấp thông tin là *i1* với các giá tri tương ứng đều lớn hơn không, tức đều mang hướng tích cực.
* *U1* gần với u0 và xa những người dùng còn lại.
* u2 gần với *U3,U4,U5, U6* và xa những người dùng còn lại.

Từ ma trận tương tự này, ta có thể phân các người dùng ra thành hai cụm *{u0,* u1} và {u2, *U3, U4, U5, U6}.* Vì ma trận **S** nhỏ nên chúng ta có thể quan sát thấy điều này; khi số người dùng lớn hơn, việc xác đinh bằng mắt thường là không khả thi. Thuật toán *phân cụm người dùng* (users clustering) sẽ được trình bày trong bài tiếp theo.

1. Hoàn thiện ma trận tiện ích

Việc dự đoán mức độ quan tâm của một người dùng tới một sản phẩm dựa trên các người dùng tương tự này khá giống với K lân cận (KNN) với hàm khoảng cách được thay bằng hàm tương tự.

Giống với như KNN, NBCF cũng dùng thông tin của *k* người dùng lân cận để dự đoán. Tất nhiên, để đánh giá độ quan tâm của một người dùng lên một sản phẩm, chúng ta chỉ quan tâm tới những người dùng đã đánh giá sản phẩm đó trong lân cận. Giá tri cần điền thường được xác đinh là trung bình có trọng số của các đánh giá đã chuẩn hoá. Có một điểm cần lưu ý, trong KNN, các trọng số được xác đinh dựa trên khoảng cách giữa hai điểm, và các khoảng cách này luôn là các số không âm. Trong NBCF, các trọng số được xác đinh dựa trên độ tương tự giữa hai người dùng. Những trọng số này có thể là các số âm. Công thức phổ biến được sử dụng để dự đoán độ quan tâm của người dùng u tới sảm phẩm *i* là:[[[9]](#footnote-10)](#bookmark7)

'ĩ^uj 2N(u.i) yi.uj sim(u, Uj )

(14.3)

E«,eV(«.o |sim(u'u

trong đó N(u, i) là tập hợp k người dùng tương tự với *u* nhất *đã đánh giá i.*

Hình [14.2d](#bookmark1467) hoàn thiện ma trận tiện ích đã chuẩn hoá. Các ô nền sọc chéo thehiện các giá tri dương, tức các sản phàm nên được gợi ý tới người dùng tương ứng. ở đây, ngưỡng được lấy là 0, ngưỡng này hoàn toàn có thể được thay đổi tuỳ thuộc vào việc ta muốn gợi ý nhiều hay ít sản phẩm.

Một ví dụ về việc tính độ quan tâm chuẩn hoá của *U1* tới *i1* được cho trong Hình [14.2e](#bookmark1467) với số lân cận *k* = 2. Các bước thực hiện như sau:

Xác đinh những người dùng đã đánh giá i1, ở đây là *u0,u3,u5.*

a.

b.

c.

Mức độ tương tự giữa *U1* và những người dùng này lần lượt là {0.83, —0.40, —0.23}.

Hai (k = 2) giá tri lớn nhất là 0.83 và —0.23 tương ứng với *u0* và *u5.*

Xác đinh các đánh giá (đã chuẩn hoá) của *u0* và *u5* tới *i1*, ta thu được hai giá tri lần lượt là 0.75 và 0.5.

d.

Dự đoán kết quả:

0.83 X 0.75 + (-0.23) X 0.5 n (

(14.4)

~~" . , " Z~~ ~ 0.48

yii,ui

0.83+ I - 0.23|

Việc quy đổi các giá tri đánh giá chuẩn hoá về thang nam có thể được thực hiện bằng cách cộng các cột của ma trận **Y** với giá tri đánh giá trung bình của mỗi người dùng như đã tính trong Hình [14.2a](#bookmark1467). Việc hệ thống quyết đinh gợi ý sản phẩm nào cho mỗi người dùng có thể được xác đinh bằng nhiều cách khác nhau. Hệ thống có thể sắp xếp các sản phẩm chưa được đánh giá theo độ giảm dần của mức độ quan tâm được dự đoán, hoặc có thể chỉ chọn các sản phẩm có độ quan tâm chuẩn hoá dương - tương ứng với việc người dùng này có nhiều khả nang thích hơn.

1. Lọc cộng tác sản phẩm

Lọc cộng tác người dùng có một số hạn chế như sau:

* Khi lượng người dùng lớn hơn số lượng sản phẩm (điều này thường xảy ra), mỗi chiều của ma trận tương tự bằng với số lượng người dùng. Việc lưu trữ một ma trận với kích thước lớn đôi khi không khả thi.
* Ma trận tiện ích **Y** thường rất thưa, tức chỉ có một tỉ lệ nhỏ các phần tử đã biết. Khi lượng người dùng lớn so với lượng sản phẩm, nhiều cột của ma trận này có ít phần tử khác không vì người dùng thường *lười* đánh giá sản phẩm. Vì thế, khi một người dùng thay đổi hoặc thêm các đánh giá, trung bình cộng các đánh giá cũng như vector chuẩn hoá tương ứng với người dùng này thay đổi theo. Kéo theo đó, việc tính toán ma trận tương tự, vốn tốn nhiều bộ nhớ và thời gian, cần được thực hiện lại.

Có một cách tiếp cận khác, thay vì tìm sự tương tự giữa các người dùng, ta có thể tìm sự tương tự giữa các sản phẩm. Từ đó nếu một người dùng thích một sản phẩm thì hệ thống nên gợi ý các sản phẩm tương tự tới người dùng đó.

Khi lượng sản phẩm nhỏ hơn lượng người dùng, việc xây dựng mô hình dựa trên dự tương tự giữa các sản phẩm có một số ưu điểm:

* Ma trận tương tự (vuông) có kích thước nhỏ hơn với số hàng bằng số sản phẩm. Việc này khiến việc lưu trữ và tính toán *ở* các bước sau được thực hiện một cách hiệu quả hơn.
* Khi ma trận tiện ích có số hàng ít hơn số cột, trung bình số lượng phần tử đã biết trong mỗi hàng sẽ nhiều hơn trung bình số lượng phần tử đã biết trong mỗi cột. Nói cách khác, trung bình số sản phẩm được đánh giá bởi một người dùng sẽ ít hơn trung bình số người dùng đã đánh giá một sản phẩm. Kéo theo đó, việc tính độ tương tự giữa các hàng trong ma trận tiện ích cũng đáng tin cậy hơn. Hơn nữa, giá tri trung bình của mỗi hàng cũng thay đổi ít hơn khi có thêm một vài đánh giá. Như vậy, ma trận tương tự cần được cập nhật ít thường xuyên hơn.

Cách tiếp cận thứ hai này được gọi là *lọc cộng tác sản phẩm* (item-item CF). Khi lượng sản phẩm ít hơn số lượng người dùng, phương pháp này được ưu tiên sử dụng hơn.

Quy trình hoàn thiện ma trận tiện tích tương tự như trong lọc cộng tác người dùng, chỉ khác là bây giờ ta cần tính độ tương tự giữa các hàng của ma trận đó.

Liên hệ giữa lọc cộng tác sản phẩm và lọc cộng tác người dùng

*Về mặt toán học, lọc cộng tác sản phẩm có thể nhận được từ lọc cộng tác ngừoi dùng bằng cách chuyển vị ma trận tiện ích và coi như sản phẩm đang đánh giá ngược người dùng. Sau khi hoàn thiện ma trận, ta cần chuyển vị một lần nữa để thu được kết quả.*

Hình [14.3](#bookmark1538) mô tả quy trình này cho cùng ví dụ trong Hình [14.2.](#bookmark1467) Một điểm thú vi trong ma trận tương tự trong Hình [14.3c](#bookmark1538) là các phần tử trong hai khu vực hình vuông lớn đều không âm, các phần tử bên ngoài là các số âm. Việc này thể hiện rằng các sản phẩm có thể được chia thành hai cụm rõ rệt. Như vậy, một cách vô tình, chúng ta đã thực hiện việc phâm cụm sản phẩm. Việc này giúp ích cho việc dự đoán ở phần sau vì các sản phẩm gần giống nhau rất có thể đã được phân vào một cụm. Kết quả cuối cùng về việc chọn sản phẩm nào để gợi ý cho mỗi người dùng được thể hiện bởi các ô có nền sọc chéo trong Hình [14.3d](#bookmark1538). Kết quả này có khác một chút so với kết quả tìm được bởi lọc cộng tác người dùng ở hai cột cuối cùng tương ứng với *u5,u6.* Nhưng dường như kết quả này hợp lý hơn vì từ ma

|  | «o | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ỈQ | 5 | 5 | 2 | 0 | 1 | ? | ? |
| i1 | 4 | ? | ? | 0 | ? | 2 | ? |
| Ì2 | ? | 4 | 1 | ? | ? | 1 | 1 |
| Ì3 | 2 | 2 | 3 | 4 | 4 | ? | 4 |
| Ì4 | 2 | 0 | 4 | ? | ? | ? | 5 |

a) Ma trận tiện ích ban đầu **Y**và trung bình của các hàng

2.6

1.75

3.17

2.75

|  | «o | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \*o | 2.4 | 2.4 | -.6 | -2.6 | -1.6 | 0 | 0 |
| i1 | 2 | 0 | 0 | -2 | 0 | 0 | 0 |
| Ì2 | 0 | 2.25 | -0.75 | 0 | 0 | -0.75 | -0.75 |
| Ì3 | -1.17 | -1.17 | -0.17 | 0.83 | 0.83 | 0 | 0.83 |
| Ì4 | -0.75 | -2.75 | 1.25 | 0 | 0 | 0 | 2.25 |

b) Ma trận tiện ích chuẩn hoá.

|  | \*o | i1 | Ì2 | Ì3 | Ì4 |
| --- | --- | --- | --- | --- | --- |
| \*o |  |  | \X\V | -0.89 | -0.52 |
| i1 | ^7 | ^1\\ | Ệ | -0.64 | -0.14 |
| Ì2 |  |  |  | -0.55 | -0.88 |
| Ì3 | -0.89 | -0.64 | -0.55 | 1 | 0.68 |
| Ì4 | -0.52 | -0.14 | -0.88 | 0.68 | 1 |

c) Ma trận tương tự sản phẩm **S**.

|  | «o | «1 | «2 | «3 | «4 | «5 | «6 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \*o | 2.4 | 2.4 | -.6 | -2.6 | -1.6 | -0.29 | -1.52 |
| i1 | 2 |  | -0.6 | -2 | -1.25 | 0 | -2.25 |
| Ì2 | ^2^41 | 2.25 | -0.75 | -2.6 | -1.20 | -0.75 | -0.75 |
| Ì3 | -1.17 | -1.17 | -0.17 | 0.83 | 0.83 |  | 0.83 |
| Ì4 | -0.75 | -2.75 | 1.25 | x\\\' | \\\\ |  | 2.25 |

d) Ma trận tiện ích chuẩn hoá sau hoàn thiện

**Hình 14.3.** Ví dụ mô tả item-item CF. a) Ma trận utility ban đầu. b) Ma trận utility đã được chuân hoá. c) User similarity matrix. d) Dự đoán các (normalized) *rating* còn thiếu.

trận tiện ích, ta nhận thấy có hai nhóm người dùng có sở thích khác nhau. Nhóm thứ nhất là *u0* và *U1*; nhóm thứ hai là những người dùng còn lại.

Mục [14.4](#bookmark1551) sau đây mô tả cách lập trình cho NNCF trên Python. Thư viện sklearn hiện chưa hỗ trợ các thuật toán gợi ý. Các em có thể tham khảo một thư viện khác khá tốt trên python là surprise [(http://surpriselib.com/)](http://surpriselib.com/).

1. Lập trình trên Python

Thuật toán lọc cộng tác tương đối đơn giản và không chứa bài toán tối ưu nào. Chúng ta tiếp tục sử dụng bộ cơ sở dữ liệu MovieLens 100k như trong bài trước. Class uuCF trong đoạn code dưới đây thực hiện quy trình lọc cộng tác người dùng. Có hai phương thức chính của class này là fit - tính ma trận tương tự, và predict - dự đoán số sao mà một người dùng sẽ đánh giá một sản phẩm:

**from** future **import** print\_function

**import** pandas as pd

**import** numpy as np

**from** sklearn.metrics.pairwise **import** cosine\_similarity

**from** scipy **import** sparse

**class** uuCF(**object**):

**def** init (self, Y\_data, k, sim\_func=cosine\_similarity):

self.Y\_data = Y\_data *# a 2d array of shape (n\_users, 3)*

* *each row of Y\_data has form [user\_id, item\_id, rating]* self.k = k *# number of neighborhood*
* *similarity function, default: cosine\_similarity* self.sim\_func = sim\_func

self.Ybar = None *# normalize data*

* *number of users*

self.n\_users = **int**(np.**max**(self.Y\_data[:, 0])) +1

* *number of items*

self.n\_items = **int**(np.**max**(self.Y\_data[:, 1])) +1

**def** fit(self):

* *normalized Y\_data -> Ybar*

users = self.Y\_data[:, 0] *# all users - first column of Y\_data* self.Ybar = self.Y\_data.copy()

self.mu = np.zeros((self.n\_users,))

**for** n **in xrange**(self.n\_users):

* *row indices of ratings made by user n*

ids = np.where(users == n)[0].astype(np.int32)

* *indices of all items rated by user n*

item\_ids = self.Y\_data[ids, 1]

* *ratings made by user n*

ratings = self.Y\_data[ids, 2]

* *avoid zero division*

self.mu[n] = np.mean(ratings) **if** ids.size > 0 **else** 0 self.Ybar[ids, 2] = ratings - self.mu[n]

* *form the rating matrix as a sparse matrix.*
* *see more: <https://goo.gl/i2mmT2>*

self.Ybar = sparse.coo\_matrix((self.Ybar[:, 2],

1], self.Ybar[:, 0])),

(self.Ybar[:,  
(self.n\_items

self.n\_users)).tocsr()

self.S = self.sim\_func(self.Ybar.T, self.Ybar.T)

**def** pred(self, u, i):

*""" predict the rating of user u for item i"""*

* *find item i*

ids = np.where(self.Y\_data[:, 1] == i)[0].astype(np.int32)

* *all users who rated i*

users\_rated\_i = (self.Y\_data[ids, 0]).astype(np.int32)

sim = self.S[u, users\_rated\_i] *# sim. of u and those users*

nns = np.argsort(sim)[-self.k:] *# most k similar users* nearest\_s = sim[nns] *# and the corresponding similarities* r = self.Ybar[i, users\_rated\_i[nns]] *# the corresponding ratings* eps = 1e-8 *# a small number to avoid zero division* **return** (r\*nearest\_s).**su^**()/(np.**abs**(nearest\_s).**sum**()+eps)+self.mu[u]

Tiếp theo, ta áp dụng vào MoviesLen 100k:

r\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp'] ratings\_base = pd.read\_csv('ml-100k/ua.base', sep='\t', names=r\_cols) ratings\_test = pd.read\_csv('ml-100k/ua.test', sep='\t', names=r\_cols) rate\_train = ratings\_base.as\_matrix()

rate\_test = ratings\_test.as\_matrix()

rate\_train[:, :2] -= 1 *# since indices start from 0*

rate\_test[:, :2] -= 1

rs = uuCF(rate\_train, k = 40)

rs.fit()

n\_tests = rate\_test.shape[0]

SE = 0 *# squared error*

**for** n **in xrange**(n\_tests):

pred = rs.pred(rate\_test[n, 0], rate\_test[n, 1])

SE += (pred - rate\_test[n, 2])\*\*2

RMSE = np.sqrt(SE/n\_tests)

**print**('User-user CF, RMSE =', RMSE)

Kết quả:

User-user CF, RMSE = 0.976614028929

Như vậy, trung bình mỗi đánh giá bi dự đoán lệch khoảng 0.976. Kết quả này tốt hìn kết quả có được bởi gợi ý dựa trên nội dung trong bài trước.

Tiếp theo, chúng ta áp dụng lọc cộng tác sản phẩm vào tập cì sở dữ liệu này. Để áp dụng lọc cộng tác sản phẩm, ta chỉ cần chuyển vi ma trận tiện ích. Trong trường hợp này, vì ma trận tiện ích được lưu dưới dạng [user\_id, item\_id, rating] nên ta chỉ cần đổi chỗ cột thứ nhất cho cột thứ hai của Y\_data:

rate\_train = rate\_train[:, [1, 0, 2]] rate\_test = rate\_test[:, [1, 0, 2]]

rs = uuCF(rate\_train, k = 40)

rs.fit()

n\_tests = rate\_test.shape[0]

SE = 0 *# squared error*

***for*** n **in xrange**(n\_tests):

pred = rs.pred(rate\_test[n, 0], rate\_test[n, 1]) SE += (pred - rate\_test[n, 2])\*\*2

RMSE = np.sqrt(SE/n\_tests) **print**('Item-item CF, RMSE =', RMSE)

Kết quả:

Item-item CF, RMSE = 0.968846083868

Như vậy, trong trường hợp này lọc cộng tác sản phẩm cho kết quả tốt hơn, ngay cả khi số sản phẩm (1682) lớn hơn số người dùng (943). Với các bài toán khác, chúng ta nên thử cả hai phương pháp trên một tập xác thực và chọn ra phương pháp cho kết quả tốt hơn. Kích thước lân cận *k* cũng có thể được thay bằng các giá tri khác.

1. Thảo luận

* Lọc cộng tác là một phương pháp gợi ý sản phẩm dựa trên hành vi của các người dùng tương tự khác lên cùng một sản phẩm. Việc làm này được thực hiện dựa trên sự tương tự giữa người dùng được mô tả bởi ma trận tương tự.
* Để tính ma trận tương tự, trước tiên ta cần chuẩn hoá dữ liệu. Phương pháp chuẩn hoá dữ liệu phổ biến là trừ mỗi cột (hoặc hàng) của ma trận tiện ích đi trung bình của các phần tử đã biết trong cột (hàng) đó.
* Hàm tương tự thường dùng là tương tự cos.
* Một hướng tiếp cận khác là thay vì đi tìm các người dùng tương tự với một người dùng (lọc cộng tác người dùng), ta đi tìm các sản phẩm tương tự với một sản phẩm cho trước (lọc cộng tác sản phẩm). Trong nhiều trường hợp, lọc cộng tác sản phẩm mang lại kết quả tốt hơn.

***Đọc thêm***

1. M. Ekstrand *et al., Collaborative filtering recommender systems.* [(https://goo.](https://goo.gl/GVn8av) [gl/GVn8av)](https://goo.gl/GVn8av) Foundations and Trends® in Human-Computer Interaction 4.2 (2011): 81-173.

Lọc cộng tác phân tích ma trận

1. Giới thiệu

Trong bài [14](#bookmark1509), chúng ta đã làm quen với phương pháp lọc cộng tác dựa trên hành vi của người dùng hoặc sản phẩm lân cận. Trong bài này, chúng ta sẽ làm quen với một hướng tiếp cận khác cho lọc cộng tác dựa trên bài toán *phân tích ma trận thành nhân tử* (matrix factorization hoặc matrix decomposition). Phương pháp này được gọi là *lọc cộng tác phân tích ma trận* (matrix factorization collaborative filtering - MFCF) .

Nhắc lại rằng trong hệ thống gợi ý dựa trên nội dung, mỗi sản phẩm được mô tả bằng một vector thông tin **x**. Trong phương pháp đó, ta cần tìm một vector trọng số **w** tương ứng với mỗi người dùng sao cho các đánh giá đã biết của người dùng tới các sản phẩm được xấp xỉ bởi:

*y* ***xTw*** (15.1)

Với cách làm này, ma trận tiện ích **Y**, *giả sử đã được điền hết,* sẽ xấp xỉ với:

T

**x**T **W**1

TT

**x**1 **W**2 ... **x**1 **W**N

TT T

**x**2 **W**1 **x**2 **W**2 ... **x**2 **W**ỵ

**x**1T

xT

**[w**1 **w**2 ... **W**tf] = **X**T**W**

(15.2)

T

xM,

TT T

**x**M**W**1 **x**M**W**2 . . . **x**M**W**n

với *M, N* lần lượt là số lượng sản phẩm và người dùng. Chú ý rằng trong hệ thống gợi ý dựa trên nội dung, **x** được xây dựng dựa trên thông tin mô tả của sản phẩm và quá trình xây dựng này độc lập với quá trình đi tìm hệ số phù hợp cho mỗi người dùng. Như vậy, việc xây dựng thông tin sản phẩm đóng vai trò quan trọng và có ảnh hưởng trực tiếp tới hiệu nang của mô hình. Thêm nữa, việc xây dựng từng mô hình riêng lẻ cho mỗi người dùng dẫn đến kết quả chưa thực sự tốt vì không khai thác được mối quan hệ giữa người dùng.

|  | *N* |  | *K* |
| --- | --- | --- | --- |
| *M* | ***Y*** |  | *M* ***X****T* |

Ma trận tiện ích (đầy đủ) Thông tin sản phẩm

|  | *N* |
| --- | --- |
| *K* | ***W*** |

Mô hình người dùng

**Hình 15.1.** Phân tích ma trận. Ma trận tiện ích **Y** 2 R M*xN* được xấp xỉ bởi tích của hai ma trận **X** 2 R M*xK* và **W** 2 RKxN.

Bây giờ, giả sử rằng không cần xây dựng trước thông tin sản phẩm **x** mà vector này có thể được huấn luyện đồng thời với mô hình của mỗi người dùng (ở đây là một vector trọng số). Điều này nghĩa là, biến số trong bài toán tối ưu là cả **X** và **W**; trong đó, mỗi cột của **X** là thông tin về một sản phẩm, mỗi cột của **W** là mô hình của một người dùng.

Với cách làm này, chúng ta đang cố gắng xấp xỉ ma trận tiện ích **Y** 2 xN bằng tích của hai ma trận **X** 2 RKxM và **W** 2 RKxN. Thông thường, *K* được chọn là một số nhỏ hơn so với *M; N*. Khi đó, cả hai ma trận **X** và **W** đều có hạng không vượt quá K. Chính vì vậy, phương pháp này còn được gọi là *phân tích ma trận hạng thấp* (low-rank matrix factorization) (xem Hình 15.1).

Một vài điểm cần lưu ý:

* **Y** tưởng chính đằng sau lọc cộng tác phân tích ma trận là tồn tại các *đặc trưng an* (latent feature) mô tả mối quan hệ giữa sản phẩm và người dùng. Ví dụ, trong hệ thống khuyến nghi các bộ phim, đặc trưng ẩn có thể là *hình sự, chính trị, hành động, hài,...;* cũng có thể là một sự kết hợp nào đó của các thể loại này. Đặc trưng ẩn cũng có thể là bất cứ điều gì mà chúng ta không thực sự cần đặt tên. Mỗi sản phẩm sẽ mang đặc trưng ẩn ở một mức độ nào đó tương ứng với các hệ số trong vector **x** của nó, hệ số càng cao tương ứng với việc mang tính chất đó càng cao. Tương tự, mỗi người dùng cũng sẽ có xu hướng thích những tính chất ẩn nào đó được mô tả bởi các hệ số trong vector **w**. Hệ số cao tương ứng với việc người dùng thích các bộ phim có tính chất ẩn đó nhiều. Giá tri của biểu thức **x**T**w** sẽ cao nếu các thành phần tương ứng của **x** và **w** đều cao (và dương) hoặc đều thấp (và âm). Điều này nghĩa là sản phẩm mang các tính chất ẩn mà người dùng thích, vậy ta nên gợi ý sản phẩm này cho người dùng đó.
* Tại sao phân tích ma trận được xếp vào lọc cộng tác? Câu trả lời đến từ việc tối ưu hàm mất mát được thảo luận ở Mục [15.2](#bookmark1592). Về cơ bản, để tìm nghiệm của bài toán tối ưu, ta phải lần lượt đi tìm **X** và **W** khi thành phần còn lại được cố đinh. Như vậy, mỗi cột của **X** sẽ phụ thuộc vào toàn bộ các cột của

**W**. Ngược lại, mỗi cột của **W** phụ thuộc vào toàn bộ các cột của **X**. Như vậy, có những mỗi quan hệ ràng buộc chằng chịt giữa các thành phần của hai ma trận trên. Vì vậy, phưìng pháp này cũng được xếp vào lọc cộng tác.

* Trong các bài toán thực tế, số lượng sản phẩm *M* và số lượng người dùng *N* thường rất lớn. Việc tìm ra các mô hình đìn giản giúp dự đoán độ quan tâm cần được thực hiện một cách nhanh nhất có thể. Lọc cộng tác dựa trên lân cận không yêu cầu việc huấn luyện quá nhiều, nhưng trong quá trình dự đoán, ta cần đi tìm độ tưìng tự của một người dùng với toàn bộ người dùng còn lại rồi suy ra kết quả. Ngược lại, với phân tích ma trận, việc huấn luyện tạp hìn vì phải lặp đi lặp lại việc tối ưu một ma trận khi cố định ma trận còn lại. Tuy nhiên, việc dự đoán đìn giản hìn vì chỉ cần tính tích vô hướng **x**T**w**, mỗi vector có độ dài *K* là một số nhỏ hìn nhiều so với M, N. Vì vậy, quá trình dự đoán không yêu cầu nặng về tính toán. Việc này khiến phân tích ma trận trở nên phù hợp với các mô hình có tập dữ liệu lớn.
* Hìn nữa, việc lưu trữ hai ma trận **X** và **W** yêu cầu lượng bộ nhớ nhỏ so với

việc lưu toàn bộ ma trận tiện ích và tưìng tự trong lọc cộng tác lân cận. Cụ thể, ta cần bộ nhớ để chứa K(M + N) phần tử thay vì M2 hoặc N2 của ma trận tưìng tự (K *M,N*).

1. Xây dựng và tối ưu hàm mất mát
2. Xấp xỉ các đánh giá đã biết

Như đã đề cập, đánh giá của người dùng *n* tới sản phẩm *m* có thể được xấp xỉ bởi *ymn =* ***Xw****n.* Ta cũng có thể thêm các hệ số điều chỉnh vào công thức xấp xỉ này và tối ưu các hệ số đó. Cụ thể:

ymn

***X^W****n* + bm + *dn*

(15.3)

Trong đó, *bm* và *dn* lượt lượt là các hệ số điều chỉnh ứng với sản phẩm *m* và người dùng n. Vector **b** = *[b1,b2,*..., bM]T là vector điều chỉnh cho các sản phẩm, vector **d** = *[d1, d2,*..., *dN*]T là vector điều chỉnh cho các người dùng. Giống như lọc cộng tác lân cận (NBCF), các giá trị này cũng có thể được coi là các giá trị giúp chuẩn hoá dữ liệu với **b** tưìng ứng với lọc cộng tác sản phẩm và **d** tưìng ứng với lọc cộng tác người dùng. Không giống như trong NBCF, các vector này sẽ được tối ưu để tìm ra các giá trị phù hợp với tập huấn luyện nhất. Thêm vào đó, huấn luyện **d** và **b** cùng lúc giúp kết hợp cả lọc cộng tác người dùng và lân cận vào một bài toán tối ưu. Vì vậy, chúng ta mong đợi rằng phưìng pháp này sẽ mang lại hiệu quả tốt hìn.

1. Hàm mất mát

Hàm mất mát cho MFCF có thể được viết như sau:

1 X X A

*L(****X****;* ***W****;* **b**; **(p y V** (**x**^**W**n + *bm* + dn *- ymn*)2 + A(||**X||**F + **||W||**P)

X ■ 7 7/ Q / J / J X m n 1 m 1 n ì/'mn / 1 Q X11 IIP 1 II IIP/

n=1 m:rmn = 1 | }

mất mát kiểm soát

mất mát trên dữ liệu

trong đó *rmn =* 1 nếu sản phẩm thứ *m* đã được đánh giá bởi người dùng thứ *n, s* là số lượng đánh giá đã biết trong tập huấn luyện, *ymn* là đánh giá chưa chuẩn ho[á9](#bookmark1587) của người dùng thứ *n* tới sản phẩm thứ *m.* Thành phần thứ nhất của hàm mất mát chính là sai số trung bình bình phương sai số của mô hình. Thành phần thứ hai chính là kiểm soát *l2* giúp mô hình tránh quá khớp.

Việc tối ưu đồng thời **X**; **W**; **b**; **d** là tương đối phức tạp. Phương pháp được sử dụng là lần lượt tối ưu một trong hai cặp (**X**; **b**), (**W**; **d**) trong lúc cố đinh cặp còn lại. Quá trình này được lặp đi lặp lại cho tới khi hàm mất mát hội tụ.

1. Tối ưu hàm mất mát

Khi cố đinh cặp (**X**; **b**), bài toán tối ưu cặp (**W**; **d**) có thể được tách thành *N* bài toán nhỏ:

C1(**w**n;dn) = Ys (**x**^**W**n + b.m + dn - ymn)2 + A **||W**n||F

(15.4)

*m:rm„ = 1*

Mỗi bài toán có thể được tối ưu bằng gradient descent. Công việc quan trọng là tính các gradient của từng hàm mất mát nhỏ này theo **w**n và *dn.* Vì biểu thức trong dấu chỉ phụ thuộc vào các sản phẩm đã được đánh giá bởi người dùng thứ *n* (tương ứng với các *rmn* = 1), ta có thể đơn giản [(15.4)](#bookmark1609) bằng cách đặt ***X****n* là ma trận con được tạo bởi các cột của **X** tương ứng với các sản phẩm đã được đánh giá bởi người dùng *n,* **b**n là vector điều chỉnh con tương ứng, và **ỷ**n là các đánh giá tương ứng. Khi đó,

L1(**W**n;dn) = llX***W****n* + **b**n + dn**1 - ỷ**n||2 + I **||W**n||2

(15.5)

với **1** là vector với mọi phần tử bằng một với kích thước phù hợp. Các gradient của nó là:

= -***X****n(****X****n****W****n* + **b**n + *dn****1 -* ỷ**n) + A**W**n *s*

*= A****1****T(****XW****n* + **b**n + *dn****1 - ỷ****n)*

***cho W****n* ***và*** *dn****:***

-***X****n(****X****n****W****n* + **b**n + *d,n****1 -* ỷ**n) + A**w**™  
*dn dn* - - ***1****T****XW****n* + **b**n + *dn****1 -* ỷ**nộ

9 việc chuẩn hoá sẽ được tự động thực hiện thông qua việc huấn luyện **b** và **d**

VWn L1

(15.6)

L1

(15.7)

***Công thùc cập nhật***

(15.8)

(15.9)

Tương tự, mỗi cột **x**m của **X** và *bm* sẽ được tìm bằng cách tối ưu bài toán



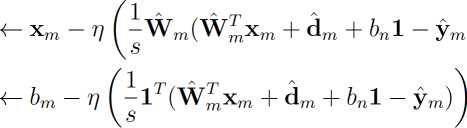
*n:rmn = 1*

Đặt ***W****m* là ma trận con tạo bởi các cột của **W** ứng với các người dùng đã đánh giá sản phẩm *m,* **d***m* là vector điều chỉnh con tương ứng, và **y***m* là vector các đánh giá tương ứng. Bài toán [(15.10)](#bookmark1610) trở thành

*L(xm;* bm) — 2 ll^W*mxm* + dm + *bn1* ym|| +2 llxmk2'

(15.11)

Tương tự ta có



***Công thức cập nhật cho x****m* ***và*** *bm****:***

*xm* ) + *^xm*

(15.12)

(15.13)

1. Lập trình Python

Chúng ta sẽ viết một class MF thực hiện việc tối ưu các biến với ma trận tiện ích được cho dưới dạng Y\_data giống như với NBCF.

Trước tiên, ta khai báo một vài thư viện cần thiết và khởi tạo class MF:

**from** future **import** print\_function

**import** pandas as pd

**import** numpy as np

**from** sklearn.metrics.pairwise **import** cosine\_similarity

**from** scipy **import** sparse

**class** MF(**object**):

**def** init (self, Y, K, lam = 0.1, Xinit = None, Winit = None,

learning\_rate = 0.5, max\_iter = 1000, print\_every = 100): self.Y = Y *# represents the utility matrix* self.K = K

self.lam = lam *# regularization parameter* self.learning\_rate = learning\_rate *# for gradient descent* self.max\_iter = max\_iter *# maximum number of iterations* self.print\_every = print\_every *# print loss after each a few iters* self.n\_users = **int**(np.**max**(Y[:, 0])) +1 self.n\_items = **int**(np.**max**(Y[:, 1])) +1 self.n\_ratings = Y.shape[0] *# number of known ratings* self.X = np.random.randn(self.n\_items, K) **if** Xinit **is** None\

**else** Xinit

self.W = np.random.randn(K, self.n\_users) **if** Winit **is** None\ **else** Winit

self.b = np.random.randn(self.n\_items) *# item biases* self.d = np.random.randn(self.n\_users) *# user biases*

Tiếp theo, chúng ta viết các phương thức loss, updateXb, updateWd cho class MF.

**def** loss(self):

L = 0

**for** i **in range**(self.n\_ratings):

* *user\_id, item\_id, rating*

n, m, rating = **int**(self.Y[i,0]), **int** (self.Y[i,1]), self.Y[i,2]

L += 0.5\*(self.X[m].dot(self.W[:, n])\

+ self.b[m] + self.d[n] - rating)\*\*2

L /= self.n\_ratings

* *regularization, don't ever forget this*

**return** L + 0.5\*self.lam\*(np.**sum**(self.X\*\*2) + np.**sum**(self.W\*\*2))

**def** updateXb(self):

**for** m **in range**(self.n\_items):

* *get all users who rated item m and corresponding ratings* ids = np.where(self.Y[:, 1] == m)[0] *# row indices of items m* user\_ids, ratings=self.Y[ids, 0].astype(np.int32),self.Y[ids, 2] Wm, dm = self.W[:, user\_ids], self.d[user\_ids]

**for** i **in range**(30): *#30 iteration for each sub problem*

xm = self.X[m]

error = xm.dot(Wm) + self.b[m] + dm - ratings grad\_xm = error.dot(Wm.T)/self.n\_ratings + self.lam\*xm grad\_bm = np.**sum** (error)/self.n\_ratings

* *gradient descent*

self.X[m] -= self.learning\_rate\*grad\_xm.reshape(-1) self.b[m] -= self.learning\_rate\*grad\_bm

**def** updateWd(self): *# and d*

**for** n **in range**(self.n\_users):

* *get all items rated by user n, and the corresponding ratings* ids = np.where(self.Y[:,0] == n)[0] *#indexes of items rated by n* item\_ids,ratings=self.Y[ids, 1].astype(np.int32), self.Y[ids, 2] Xn, bn = self.X[item\_ids], self.b[item\_ids]

**for** i **in range**(30): *#30 iteration for each sub problem*

wn = self.W[:, n]

error = Xn.dot(wn) + bn + self.d[n] - ratings grad\_wn = Xn.T.dot(error)/self.n\_ratings + self.lam\*wn grad\_dn = np.**sum** (error)/self.n\_ratings

* *gradient descent*

self.W[:, n] -= self.learning\_rate\*grad\_wn.reshape(-1) self.d[n] -= self.learning\_rate\*grad\_dn

Phần tiếp theo là quá trình tối ưu chính của MF (fit), dự đoán đánh giá (pred) và đánh giá chất lượng mô hình bằng RMSE (evaluate RMSE).

**def** fit(self):

**for** it **in range**(self.max\_iter):

self.updateWd()

self.updateXb()

**if** (it + 1) % self.print\_every == 0:

rmse\_train = self.evaluate\_RMSE(self.Y)

**print**('iter = %d, loss = %.4f, RMSE train = %.4f'%(it + 1,

self.loss(), rmse\_train))

**def** pred(self, u, i):

ff H H

*predict the rating of user u for item i*

ft If ft

u, i = **int** (u), **int**(i)

pred = self.X[i, :].dot(self.W[:, u]) + self.b[i] + self.d[u] **return max**(0, **min**(5, pred)) *# 5-scale in MoviesLen*

**def** evaluate\_RMSE(self, rate\_test):

n\_tests = rate\_test.shape[0] *# number of test*

SE = 0 *# squared error*

**for** n **in range**(n\_tests):

pred = self.pred(rate\_test[n, 0], rate\_test[n, 1])

SE += (pred - rate\_test[n, 2])\*\*2

RMSE = np.sqrt(SE/n\_tests) **return** RMSE

Tới đây, class MF đã được xây dựng với các phương thúc cần thiết. Ta cần kiểm tra chất lượng mô hình khi áp dụng lên tập dữ liệu MoviesLen 100k:

|  |  |
| --- | --- |
| r\_cols = ['user\_id', 'movie\_id', 'rating', 'unix\_timestamp'] ratings\_base = pd.read\_csv('ml-100k/ua.base', sep='\t', names=r\_cols) ratings\_test = pd.read\_csv('ml-100k/ua.test', sep='\t', names=r\_cols) | |
| rate\_train = ratings\_base.as\_matrix() rate\_test = ratings\_test.as\_matrix() |  |
| *# indices start from 0*  rate\_train[:, :2] -= 1  rate\_test[:, :2] -= 1 |  |
| rs = MF(rate\_train, K = 50, lam = .01, print\_every = 5, max\_iter = 30)  rs.fit()  *# evaluate on test data*  RMSE = rs.evaluate\_RMSE(rate\_test)  **print**('\nMatrix Factorization CF, RMSE = %.4f' %RMSE) | learning\_rate = 50, |

Kết quả:

iter = 5, loss = 0.4447, RMSE train = 0.9429 iter = 10, loss = 0.4215, RMSE train = 0.9180

iter = 15, loss = 0.4174, RMSE train = 0.9135

iter = 20, loss = 0.4161, RMSE train = 0.9120

iter = 25, loss = 0.4155, RMSE train = 0.9114

iter = 30, loss = 0.4152, RMSE train = 0.9110

Matrix Factorization CF, RMSE = 0.9621

RMSE thu được là 0.9621, tốt hơn so với NBCF trong bài trước (0.9688).

1. Thảo luận

* **Phân tích ma trận không âm:**. Khi ma trận tiện ích chưa được chuẩn hoá, các phần tử đều là giá tri không âm. Ke cả trong trường hợp dải giá tri của các đánh giá có chứa giá tri âm, ta chỉ cần cộng thêm vào ma trận tiện ích một giá tri hợp lý để có được các thành phần là các số không âm. Khi đó, một phương pháp phân tích ma trận thường mang lại hiệu quả cao trong các hệ thống gợi ý là *phân tích ma trận không âm* (nonnegative matrix factorization - NMF), tức phân tích ma trận thành tích các ma trận có các phần tử không âm. Lúc này, đặc trưng ẩn của một sản phẩm và hệ số tương ứng của người dùng là các số không âm.

Thông qua phân tích ma trận, người dùng và sản phẩm được liên kết với nhau bởi các *đặc trưng ẩn.* Độ liên kết của mỗi người dùng và sản phẩm tới mỗi đặc trưng ẩn được đo bằng thành phần tương ứng trong vector đặc trưng, giá tri càng lớn thể hiện việc người dùng hoặc sản phẩm có liên quan đến đặc trưng ẩn đó càng lớn. Bằng trực giác, sự liên quan của một người dùng hoặc sản phẩm đến một đặc trưng ẩn nên là một số không âm với giá tri không thể hiện việc không liên quan thay vì giá tri âm. Hơn nữa, mỗi người dùng và sản phẩm chỉ liên quan đến một vài đặc trưng ẩn nhất đinh. Vì vậy, các vector đặc trưng cho người dùng và sản phẩm nên là các vector không âm và có rất nhiều giá tri bằng không. Những nghiệm này có the đạt được bằng cách cho thêm ràng buộc không âm vào các thành phần của **X** và **W**. Đây chính là nguồn gốc của ý tưởng và tên gọi phân tích ma trận không âm.

* **Phân tích ma trận điều chỉnh nhỏ**: thời gian dự đoán của một hệ thống gợi ý sử dụng phân tích ma trận là rất nhanh nhưng thời gian huấn luyện là khá lâu với các bài toán quy mô lớn. Thực tế cho thấy, ma trận tiện ích thay đổi liên tục vì có thêm người dùng, sản phẩm cũng như các đánh giá mới, vì vậy các tham số mô hình cũng phải thường xuyên được cập nhật. Điều này đồng nghĩa với việc ta phải tiếp tục thực hiện quá trình huấn luyện vốn tốn khá nhiều thời gian. Thay vì huấn luyện lại toàn bộ mô hình, ta có the điều chỉnh các ma trận **X** và **W** bằng cách huấn luyện thêm một vài vòng lặp. Kỹ thuật này được gọi là *phân tích ma trận điều chỉnh nhỏ* (incremental matrix factorization), được áp dụng nhiều trong các bài toán quy mô lớn.
* Có nhiều các giải bài toán tối ưu của phân tích ma trận ngoài cách áp dụng gra­dient descent. Các em có thể xem thêm *alternating least square (ALS)* [(https:](https://goo.gl/g2M4fb) [//goo.gl/g2M4fb)](https://goo.gl/g2M4fb), *generalized low rank models* [(https://goo.gl/DrDWyW)](https://goo.gl/DrDWyW).

1. Làm mềm Laplace được sử dụng để tránh trường hợp một từ trong tập kiểm tra chưa xuất hiện trong tập huấn luyện. [↑](#footnote-ref-2)
2. Hình 8.8a: Cả bốn phương pháp trên đây đều có thể áp dụng được. [↑](#footnote-ref-3)
3. Phần chứng minh coi như một bài tập nhỏ cho bạn đọc. [↑](#footnote-ref-4)
4. Phần chứng minh coi như một bài tập nhỏ khác cho sinh viên. [↑](#footnote-ref-5)
5. Việc lấy trung bình này giống cách đo trong các thí nghiệm vật lý. Để đo một đại lượng, người ta thường thực hiện việc đo nhiều lằn rồi lấy kết quả trung bình đe tránh sai số. ơ đây, về mặt toán học, *b* phải như nhau theo mọi cách tính. Tuy nhiên, khi tính toán bằng máy tính, chúng ta có the gặp các sai số nhỏ. Việc lấy trung bình sẽ làm giảm sai số đó. [↑](#footnote-ref-6)
6. SVC là viết tắt của *bộ phăn loại vector hễ trợ* (support vector classifier). [↑](#footnote-ref-7)
7. Tổng các trị riêng của một ma trận vuông bất kỳ luôn bằng vết của ma trận đó. [↑](#footnote-ref-8)
8. Mỗi thành phần trên đường chéo chính của ma trận hiệp phương sai chính là phương sai của thành phần dữ liệu tương ứng. [↑](#footnote-ref-9)
9. Sự khác biệt so với trung bình có trọng số là mẫu số có sử dụng trị tuyệt đối. [↑](#footnote-ref-10)