Vũ Hữu Tiệp

Machine Learning cơ bản

Theo blog: http://machinelearningcoban.com

First Edition

March 4, 2017

Contents

1

Giới thiệu

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học - *Với những từ chuyên ngành, tôi sẽ dùng song song cả tiếng Anh và tiếng Việt, tuy nhiên sẽ ưu tiên tiếng Anh vì thuận tiện hơn trong việc tra cứu*) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, ..., chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning. (Xem thêm Jarvis - trợ lý thông minh cho căn nhà của Mark Zuckerberg.)

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, *Machine learning is the subfield of computer science that "gives computers the ability to learn without being explicitly programmed"*. Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể. Bạn Nguyễn Xuân Khánh tại đại học Maryland đang viết một cuốn sách về Machine Learning bằng tiếng Việt khá thú vị, các bạn có thể tham khảo bài Machine Learning là gì?.

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước tiến dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc (Xem thêm 8 Inspirational Applications of Deep Learning).

Mối quan hệ giữa Artificial Intelligence, Machine Learning, và Deep Learning được cho trong Hình 1.1.

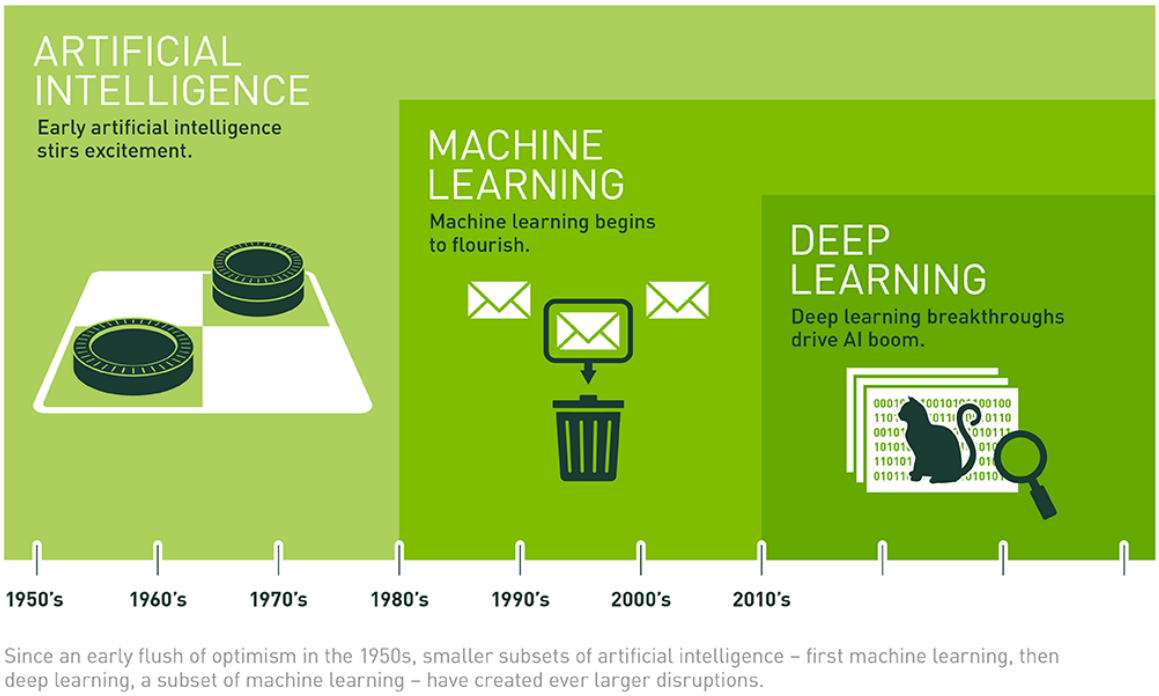

Fig. 1.1: Mối quan hệ giữa AI, Machine Learning và Deep Learning. Nguồn: What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?

1.1 Mục đích viết Blog

Nhu cầu về nhân lực ngành Machine Learning (Deep Learning) đang ngày một cao, kéo theo đó nhu cầu học Machine Learning trên thế giới và ở Việt Nam ngày một lớn. Cá nhân tôi cũng muốn hệ thống lại kiến thức của mình về lĩnh vực này để chuẩn bị cho tương lai (đây là một trong những mục tiêu của tôi trong năm 2017). Tôi sẽ cố gắng đi từ những thuật toán cơ bản nhất của Machine Learning kèm theo các ví dụ và mã nguồn trong mỗi bài viết. Tôi sẽ viết 1-2 tuần 1 bài (việc viết các công thức toán và code trên blog thực sự tốn nhiều thời gian hơn tôi từng nghĩ). Đồng thơi, tôi cũng mong muốn nhận được phản hồi của bạn đọc để qua những thảo luận, tôi và các bạn có thể nắm bắt được các thuật toán này.

Khi chuẩn bị các bài viết, tôi sẽ giả định rằng bạn đọc có một chút kiến thức về Đại Số Tuyến Tính (Linear Algebra), Xác Suất Thống Kê (Probability and Statistics) và có kinh nghiệm về lập trình Python. Nếu bạn chưa có nhiều kinh nghiệm về các lĩnh vực này, đừng quá lo lắng vì mỗi bài sẽ chỉ sử dụng một vài kỹ thuật cơ bản. Hãy để lại câu hỏi của bạn ở phần Comment bên dưới mỗi bài, tôi sẽ thảo luận thêm với các bạn.

Trong bài tiếp theo của blog này, tôi sẽ giới thiệu về các nhóm thuật toán Machine learning cơ bản. Mời các bạn theo dõi.

1.2 Tham khảo thêm

1.2.1 Các khóa học

Tiếng Anh

1. Machine Learning với thầy Andrew Ng trên Coursera (*Khóa học nổi tiếng nhất về Machine Learning*)
2. Deep Learning by Google trên Udacity (*Khóa học nâng cao hơn về Deep Learning với Tensorflow*)
3. Machine Learning mastery (*Các thuật toán Machine Learning cơ bản*)

Tiếng Việt

Lưu ý: *Các khóa học này tôi chưa từng tham gia, chỉ đưa ra để các bạn tham khảo.*

1. Machine Learning 1/2017
2. Nhập môn Machine Learning, Tech Master- Cao Thanh Hà *POSTECH*()

1.2.2 Các trang Machine Learning tiếng Việt khác

1. Machine Learning trong Xử Lý Ngôn Ngữ Tự Nhiên - Nhóm Đông Du *Nhật Bản*
2. Machine Learning cho người mới bắt đầu - Ông Xuân Hồng *JAIST*.
3. Machine Learning book for Vietnamese - Nguyễn Xuân Khánh *University of Maryland*

2

Phân nhóm các thuật toán Machine Learning

Có hai cách phổ biến phân nhóm các thuật toán Machine learning. Một là dựa trên phương thức học (learning style), hai là dựa trên chức năng (function) (của mỗi thuật toán).

2.1 Phân nhóm dựa trên phương thức học

Theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Supervise learning, Unsupervised learning, Semi-supervised lerning và Reinforcement learning. *Có một số cách phân nhóm không có Semi-supervised learning hoặc Reinforcement learning.*

2.1.1 Supervised Learning (Học có giám sát)

Supervised learning là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (*input, outcome*) đã biết từ trước. Cặp dữ liệu này còn được gọi là (*data. label*), tức (*dữ liệu, nhãn*). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine Learning.

Một cách toán học, Supervised learning là khi chúng ra có một tập hợp biến đầu vào $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ và một tập hợp nhãn tương ứng $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$, trong đó $\mathbf{x}_i, \mathbf{y}_i$ là các vector. Các cặp dữ liệu biết trước $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$ được gọi là tập *training data* (dữ liệu huấn luyện). Từ tập traing data này, chúng ta cần tạo ra một hàm số ánh xạ mỗi phần tử từ tập $\mathcal{X}$ sang một phần tử (xấp xỉ) tương ứng của tập $\mathcal{Y}$:

$$\mathbf{y}_i \approx f(\mathbf{x}_i), \quad \forall i = 1, 2, \ldots, N$$

Mục đích là xấp xỉ hàm số f thật tốt để khi có một dữ liệu $\mathbf{x}$ mới, chúng ta có thể tính được nhãn tương ứng của nó $\mathbf{y} = f(\mathbf{x})$.

Ví dụ 1: trong nhận dạng chữ viết tay (Hình 2.1), ta có ảnh của hàng nghìn ví dụ của mỗi chữ số được viết bởi nhiều người khác nhau. Chúng ta đưa các bức ảnh này vào trong một

thuật toán và chỉ cho nó biết mỗi bức ảnh tương ứng với chữ số nào. Sau khi thuật toán tạo ra (sau khi *học*) một mô hình, tức một hàm số mà đầu vào là một bức ảnh và đầu ra là một chữ số, khi nhận được một bức ảnh mới mà mô hình **chưa nhìn thấy bao giờ**, nó sẽ dự đoán bức ảnh đó chứa chữ số nào.

Fig. 2.1: MNIST: bộ cơ sở dữ liệu của chữ số viết tay. (Nguồn: Simple Neural Network implementation in Ruby)

Ví dụ này khá giống với cách học của con người khi còn nhỏ. Ta đưa bảng chữ cái cho một đứa trẻ và chỉ cho chúng đây là chữ A, đây là chữ B. Sau một vài lần được dạy thì trẻ có thể nhận biết được đâu là chữ A, đâu là chữ B trong một cuốn sách mà chúng chưa nhìn thấy bao giờ.

Ví dụ 2: Thuật toán dò các khuôn mặt trong một bức ảnh đã được phát triển từ rất lâu. Thời gian đầu, facebook sử dụng thuật toán này để chỉ ra các khuôn mặt trong một bức ảnh và yêu cầu người dùng *tag friends* - tức gán nhãn cho mỗi khuôn mặt. Số lượng cặp dữ liệu (*khuôn mặt, tên người*) càng lớn, độ chính xác ở những lần tự động *tag* tiếp theo sẽ càng lớn.

Ví dụ 3: Bản thân thuật toán dò tìm các khuôn mặt trong 1 bức ảnh cũng là một thuật toán Supervised learning với training data (dữ liệu học) là hàng ngàn cặp (*ảnh, mặt người*) và (*ảnh, không phải mặt người*) được đưa vào. Chú ý là dữ liệu này chỉ phân biệt *mặt người* và *không phải mặt ngươi* mà không phân biệt khuôn mặt của những người khác nhau.

Thuật toán supervised learning còn được tiếp tục chia nhỏ ra thành hai loại chính:

Classification (Phân loại)

Một bài toán được gọi là *classification* nếu các *label* của *input data* được chia thành một số hữu hạn nhóm. Ví dụ: Gmail xác định xem một email có phải là spam hay không; các hãng tín dụng xác định xem một khách hàng có khả năng thanh toán nợ hay không. Ba ví dụ phía trên được chia vào loại này.

Regression (Hồi quy)

(tiếng Việt dịch là *Hồi quy*, tôi không thích cách dịch này vì bản thân không hiểu nó nghĩa là gì)

Nếu *label* không được chia thành các nhóm mà là một giá trị thực cụ thể. Ví dụ: một căn nhà rộng x m^2, có y phòng ngủ và cách trung tâm thành phố z km sẽ có giá là bao nhiêu?

Gần đây Microsoft có một ứng dụng dự đoán giới tính và tuổi dựa trên khuôn mặt. Phần dự đoán giới tính có thể coi là thuật toán **Classification**, phần dự đoán tuổi có thể coi là thuật toán **Regression**. *Chú ý rằng phần dự đoán tuổi cũng có thể coi là* ***Classification*** *nếu ta coi tuổi là một số nguyên dương không lớn hơn 150, chúng ta sẽ có 150 class (lớp) khác nhau.*

2.1.2 Unsupervised Learning (Học không giám sát)

Trong thuật toán này, chúng ta không biết được *outcome* hay *nhãn* mà chỉ có dữ liệu đầu vào. Thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó, ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimention reduction) để thuận tiện trong việc lưu trữ và tính toán.

Một cách toán học, Unsupervised learning là khi chúng ta chỉ có dữ liệu vào $\mathcal{X}$ mà không biết *nhãn* $\mathcal{Y}$ tương ứng.

Những thuật toán loại này được gọi là Unsupervised learning vì không giống như Supervised learning, chúng ta không biết câu trả lời chính xác cho mỗi dữ liệu đầu vào. Giống như khi ta học, không có thầy cô giáo nào chỉ cho ta biết đó là chữ A hay chữ B. Cụm *không giám sát* được đặt tên theo nghĩa này.

Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại:

Clustering (phân nhóm)

Một bài toán phân nhóm toàn bộ dữ liệu $\mathcal{X}$ thành các nhóm nhỏ dựa trên sự liên quan giữa các dữ liệu trong mỗi nhóm. Ví dụ: phân nhóm khách hàng dựa trên hành vi mua hàng. Điều này cũng giống như việc ta đưa cho một đứa trẻ rất nhiều mảnh ghép với các hình thù và màu sắc khác nhau, ví dụ tam giác, vuông, tròn với màu xanh và đỏ, sau đó yêu cầu trẻ phân chúng thành từng nhóm. Mặc dù không cho trẻ biết mảnh nào tương ứng với hình nào hoặc màu nào, nhiều khả năng chúng vẫn có thể phân loại các mảnh ghép theo màu hoặc hình dạng.

Association

Là bài toán khi chúng ta muốn khám phá ra một quy luật dựa trên nhiều dữ liệu cho trước. Ví dụ: những khách hàng nam mua quần áo thường có xu hướng mua thêm đồng hồ hoặc thắt lưng; những khán giả xem phim Spider Man thường có xu hướng xem thêm phim Bat Man, dựa vào đó tạo ra một hệ thống gợi ý khách hàng (Recommendation System), thúc đẩy nhu cầu mua sắm.

2.1.3 Semi-Supervised Learning (Học bán giám sát)

Các bài toán khi chúng ta có một lượng lớn dữ liệu $\mathcal{X}$ nhưng chỉ một phần trong chúng được gán nhãn được gọi là Semi-Supervised Learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm được nêu bên trên.

Một ví dụ điển hình của nhóm này là chỉ có một phần ảnh hoặc văn bản được gán nhãn (ví dụ bức ảnh về người, động vật hoặc các văn bản khoa học, chính trị) và phần lớn các bức ảnh/văn bản khác chưa được gán nhãn được thu thập từ internet. Thực tế cho thấy rất nhiều các bài toàn Machine Learning thuộc vào nhóm này vì việc thu thập dữ liệu có nhãn tốn rất nhiều thời gian và có chi phí cao. Rất nhiều loại dữ liệu thậm chí cần phải có chuyên gia mới gán nhãn được (ảnh y học chẳng hạn). Ngược lại, dữ liệu chưa có nhãn có thể được thu thập với chi phí thấp từ internet.

2.1.4 Reinforcement Learning (Học Củng Cố)

Reinforcement learning là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất (maximizing the performance). Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.

Ví dụ 1: AlphaGo (Hình 2.2) gần đây nổi tiếng với việc chơi cờ vây thắng cả con người. Cờ vây được xem là có độ phức tạp cực kỳ cao với tổng số nước đi là xấp xỉ 10^{761}, so với cờ vua là 10^{120} và tổng số nguyên tử trong toàn vũ trụ là khoảng 10^{80}!! Vì vậy, thuật toán phải chọn ra 1 nước đi tối ưu trong số hàng nhiều tỉ tỉ lựa chọn, và tất nhiên, không thể áp dụng thuật toán tương tự như IBM Deep Blue) (IBM Deep Blue đã thắng con người trong môn cờ vua 20 năm trước). Về cơ bản, AlphaGo bao gồm các thuật toán thuộc cả Supervised learning và Reinforcement learning. Trong phần Supervised learning, dữ liệu từ các ván cờ do con người chơi với nhau được đưa vào để huấn luyện. Tuy nhiên, mục đích cuối cùng của AlphaGo không phải là chơi như con người mà phải thậm chí thắng cả con người. Vì vậy, sau khi *học* xong các ván cờ của con người, AlphaGo tự chơi với chính nó với hàng triệu ván chơi để tìm ra các nước đi mới tối ưu hơn. Thuật toán trong phần tự chơi này được xếp vào loại Reinforcement learning. (Xem thêm tại Google DeepMind's AlphaGo: How it works).

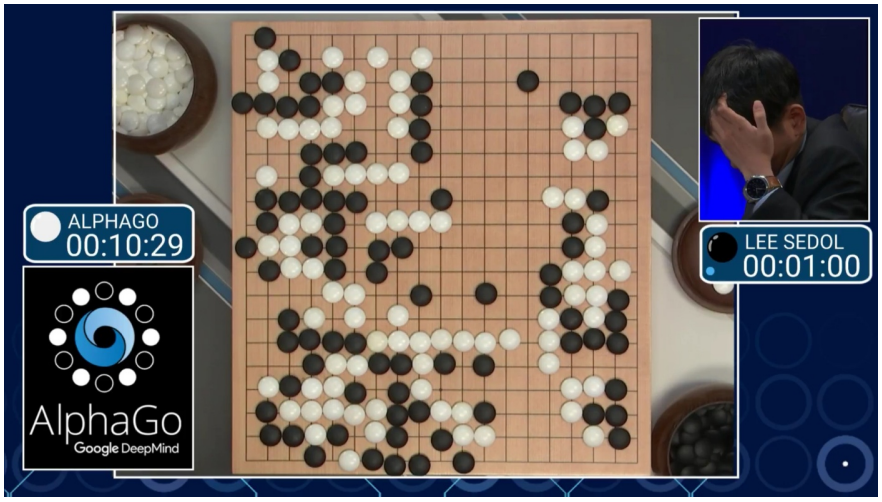

Fig. 2.2: AlphaGo chơi cờ vây với Lee Sedol. AlphaGo là một ví dụ của Reinforcement learning (Nguồn AlphaGo AI Defeats Sedol Again, With 'Near Perfect Game')

Ví dụ 2: Huấn luyện cho máy tính chơi game Mario. Đây là một chương trình thú vị dạy máy tính chơi game Mario. Game này đơn giản hơn cờ vây vì tại một thời điểm, người chơi chỉ phải bấm một số lượng nhỏ các nút (di chuyển, nhảy, bắn đạn) hoặc không cần bấm nút nào. Đồng thời, phản ứng của máy cũng đơn giản hơn và lặp lại ở mỗi lần chơi (tại thời điểm cụ thể sẽ xuất hiện một chướng ngại vật cố định ở một vị trí cố định). Đầu vào của thuật toán là sơ đồ của màn hình tại thời điểm hiện tại, nhiệm vụ của thuật toán là với đầu vào đó, tổ hợp phím nào nên được bấm. Việc huấn luyện này được dựa trên điểm số cho việc di chuyển được bao xa trong thời gian bao lâu trong game, càng xa và càng nhanh thì được điểm thưởng càng cao (điểm thưởng này không phải là điểm của trò chơi mà là điểm do chính người lập trình tạo ra). Thông qua huấn luyện, thuật toán sẽ tìm ra một cách tối ưu để tối đa số điểm trên, qua đó đạt được mục đích cuối cùng là cứu công chúa.

2.2 Phân nhóm dựa trên chức năng

Có một cách phân nhóm thứ hai dựa trên chức năng của các thuật toán. Trong phần này, tôi xin chỉ liệt kê các thuật toán. Thông tin cụ thể sẽ được trình bày trong các bài viết khác tại blog này. Trong quá trình viết, tôi có thể sẽ thêm bớt một số thuật toán.

2.2.1 Regression Algorithms

1. Linear Regression
2. Logistic Regression
3. Stepwise Regression

2.2.2 Classification Algorithms

1. Linear Classifier
2. Support Vector Machine (SVM)
3. Kernel SVM
4. Sparse Represntation-based classification (SRC)

2.2.3 Instance-based Algorithms

1. k-Nearest Neighbor (kNN)
2. Learnin Vector Quantization (LVQ)

2.2.4 Regularization Algorithms

1. Ridge Regression
2. Least Absolute Shringkage and Selection Operator (LASSO)
3. Least-Angle Regression (LARS)

2.2.5 Bayesian Algorithms

1. Naive Bayes
2. Gausian Naive Bayes

2.2.6 Clustering Algorithms

1. k-Means clustering
2. k-Medians
3. Expectation Maximisation (EM)

2.2.7 Artificial Neural Network Algorithms

1. Perceptron
2. Softmax Regression
3. Multi-layer Perceptron
4. Back-Propagation

2.2.8 Dimensionality Reduction Algorithms

1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)

2.2.9 Ensemble Algorithms

1. Boosting
2. AdaBoost
3. Random Forest

Và còn rất nhiều các thuật toán khác.

2.3 Tài liệu tham khảo

1. A Tour of Machine Learning Algorithms
2. Điểm qua các thuật toán Machine Learning hiện đại

3

Linear Regression

Trong bài này, tôi sẽ giới thiệu một trong những thuật toán cơ bản nhất (và đơn giản nhất) của Machine Learning. Đây là một thuật toán *Supervised learning* có tên **Linear Regression** (Hồi Quy Tuyến Tính). Bài toán này đôi khi được gọi là *Linear Fitting* (trong thống kê) hoặc *Linear Least Square*.

3.1 Giới thiệu

Quay lại ví dụ đơn giản được nêu trong bài trước: một căn nhà rộng x_1 m^2, có x_2 phòng ngủ và cách trung tâm thành phố x_3 km có giá là bao nhiêu. Giả sử chúng ta đã có số liệu thống kê từ 1000 căn nhà trong thành phố đó, liệu rằng khi có một căn nhà mới với các thông số về diện tích, số phòng ngủ và khoảng cách tới trung tâm, chúng ta có thể dự đoán được giá của căn nhà đó không? Nếu có thì hàm dự đoán $y = f(\mathbf{x})$ sẽ có dạng như thế nào. Ở đây $\mathbf{x} = [x_1, x_2, x_3]$ là một vector hàng chứa thông tin *input*, y là một số vô hướng (scalar) biểu diễn *output* (tức giá của căn nhà trong ví dụ này).

Lưu ý về ký hiệu toán học: *trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ x_1, N, y, k. Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ $\mathbf{y}, \mathbf{x}_1$. Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ $\mathbf{X}, \mathbf{Y}, \mathbf{W}$.*

Một cách đơn giản nhất, chúng ta có thể thấy rằng: i) diện tích nhà càng lớn thì giá nhà càng cao; ii) số lượng phòng ngủ càng lớn thì giá nhà càng cao; iii) càng xa trung tâm thì giá nhà càng giảm. Một hàm số đơn giản nhất có thể mô tả mối quan hệ giữa giá nhà và 3 đại lượng đầu vào là:

$$y \approx \hat{y} = f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 \tag{3.1}$$

trong đó, w_1, w_2, w_3, w_0 là các hằng số, w_0 còn được gọi là bias. Mối quan hệ $y \approx f(\mathbf{x})$ bên trên là một mối quan hệ tuyến tính (linear). Bài toán chúng ta đang làm là một bài toán thuộc loại regression. Bài toán đi tìm các hệ số tối ưu
w_1, w_2, w_3, w_0 chính vì vậy được gọi là bài toán Linear Regression.

Chú ý 1: y là giá trị thực của *outcome* (dựa trên số liệu thống kê chúng ta có trong tập *training data*), trong khi $\hat{y}$ là giá trị mà mô hình Linear Regression dự đoán được. Nhìn chung, y và $\hat{y}$ là hai giá trị khác nhau do có sai số mô hình, tuy nhiên, chúng ta mong muốn rằng sự khác nhau này rất nhỏ.

Chú ý 2: *Linear* hay *tuyến tính* hiểu một cách đơn giản là *thẳng, phẳng*. Trong không gian hai chiều, một hàm số được gọi là *tuyến tính* nếu đồ thị của nó có dạng một *đường thẳng*. Trong không gian ba chiều, một hàm số được goi là *tuyến tính* nếu đồ thị của nó có dạng một *mặt phẳng*. Trong không gian nhiều hơn 3 chiều, khái niệm *mặt phẳng* không còn phù hợp nữa, thay vào đó, một khái niệm khác ra đời được gọi là *siêu mặt phẳng* (*hyperplane*). Các hàm số tuyến tính là các hàm đơn giản nhất, vì chúng thuận tiện trong việc hình dung và tính toán. Chúng ta sẽ được thấy trong các bài viết sau, *tuyến tính* rất quan trọng và hữu ích trong các bài toán Machine Learning. Kinh nghiệm cá nhân tôi cho thấy, trước khi hiểu được các thuật toán *phi tuyến* (non-linear, không phẳng), chúng ta cần nắm vững các kỹ thuật cho các mô hình *tuyến tính*.

3.2 Phân tích toán học

3.2.1 Dạng của Linear Regression

Trong phương trình (1) phía trên, nếu chúng ta đặt $\mathbf{w} = [w_0, w_1, w_2, w_3]^T =$ là vector (cột) hệ số cần phải tối ưu và $\bar{\mathbf{x}} = [1, x_1, x_2, x_3]$ (đọc là x *bar* trong tiếng Anh) là vector (hàng) dữ liệu đầu vào *mở rộng*. Số 1 ở đầu được thêm vào để phép tính đơn giản hơn và thuận tiện cho việc tính toán. Khi đó, phương trình (1) có thể được viết lại dưới dạng:

$$y \approx \bar{\mathbf{x}}\mathbf{w} = \hat{y}$$

Chú ý rằng $\bar{\mathbf{x}}$ là một vector hàng. (Xem thêm về ký hiệu vector hàng và cột tại đây)

3.2.2 Sai số dự đoán

Chúng ta mong muốn rằng sự sai khác e giữa giá trị thực y và giá trị dự đoán $\hat{y}$ (đọc là y *hat* trong tiếng Anh) là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

$$\frac{1}{2}e^2 = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \bar{\mathbf{x}}\mathbf{w})^2$$

trong đó hệ số $\frac{1}{2}$ (*lại*) là để thuận tiện cho việc tính toán (khi tính đạo hàm thì số $\frac{1}{2}$ sẽ bị triệt tiêu). Chúng ta cần e^2 vì $e = y - \hat{y}$ có thể là một số âm, việc nói e nhỏ nhất sẽ không đúng vì khi $e = -\infty$ là rất nhỏ nhưng sự sai lệch là rất lớn. Bạn đọc có thể tự đặt câu hỏi: **tại sao không dùng trị tuyệt đối $\|e\|$ mà lại dùng bình phương e^2 ở đây?** Câu trả lời sẽ có ở phần sau.

3.2.3 Hàm mất mát

Điều tương tự xảy ra với tất cả các cặp *(input, outcome)* $(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N$, với N là số lượng dữ liệu quan sát được. Điều chúng ta muốn, tổng sai số là nhỏ nhất, tương đương với việc tìm $\mathbf{w}$ để hàm số sau đạt giá trị nhỏ nhất:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \bar{\mathbf{x}}_\mathbf{i}\mathbf{w})^2 \tag{3.2}$$

Hàm số $\mathcal{L}(\mathbf{w})$ được gọi là **hàm mất mát** (loss function) của bài toán Linear Regression. Chúng ta luôn mong muốn rằng sự mất mát (sai số) là nhỏ nhất, điều đó đồng nghĩa với việc tìm vector hệ số $\mathbf{w}$ sao cho giá trị của hàm mất mát này càng nhỏ càng tốt. Giá trị của $\mathbf{w}$ làm cho hàm mất mát đạt giá trị nhỏ nhất được gọi là *điểm tối ưu* (optimal point), ký hiệu:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

Trước khi đi tìm lời giải, chúng ta đơn giản hóa phép toán trong phương trình hàm mất mát (3.2). Đặt $\mathbf{y} = [y_1; y_2; \ldots; y_N]$ là một vector cột chứa tất cả các *output* của *training data*; $\bar{\mathbf{X}} = [\bar{\mathbf{x}}_1; \bar{\mathbf{x}}_2; \ldots; \bar{\mathbf{x}}_N]$ là ma trận dữ liệu đầu vào (mở rộng) mà mỗi hàng của nó là một điểm dữ liệu. Khi đó hàm số mất mát $\mathcal{L}(\mathbf{w})$ được viết dưới dạng ma trận đơn giản hơn:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \bar{\mathbf{x}}_i\mathbf{w})^2 = \frac{1}{2}\|\mathbf{y} - \bar{\mathbf{X}}\mathbf{w}\|_2^2 \tag{3.3}$$

với $\|\mathbf{z}\|_2$ là Euclidean norm (chuẩn Euclid, hay khoảng cách Euclid), nói cách khác $\|\mathbf{z}\|_2^2$ là tổng của bình phương mỗi phần tử của vector $\mathbf{z}$. Tới đây, ta đã có một dạng đơn giản của hàm mất mát được viết như phương trình (3.3).

3.2.4 Nghiệm cho bài toán Linear Regression

Cách phổ biến nhất để tìm nghiệm cho một bài toán tối ưu (chúng ta đã biết từ khi học cấp 3) là giải phương trình đạo hàm (gradient) bằng 0! Tất nhiên đó là khi việc tính đạo hàm và việc giải phương trình đạo hàm bằng 0 không quá phức tạp. Thật may mắn, với các mô hình tuyến tính, hai việc này là khả thi.

Đạo hàm theo $\mathbf{w}$ của hàm mất mát là:

$$\frac{\partial\mathcal{L}(\mathbf{w})}{\partial\mathbf{w}} = \bar{\mathbf{X}}^T(\bar{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

Các bạn có thể tham khảo bảng đạo hàm theo vector hoặc ma trận của một hàm số trong mục D.2 của tài liệu này. *Đến đây tôi xin quay lại câu hỏi ở phần Sai số dự đoán phía trên về việc tại sao không dùng trị tuyệt đối mà lại dùng bình phương. Câu trả lời là hàm bình phương có đạo hàm tại mọi nơi, trong khi hàm trị tuyệt đối thì không (đạo hàm không xác định tại 0).*

Phương trình đạo hàm bằng 0 tương đương với:

$$\bar{\mathbf{X}}^T\bar{\mathbf{X}}\mathbf{w} = \bar{\mathbf{X}}^T\mathbf{y} \triangleq \mathbf{b} \tag{3.4}$$

(ký hiệu $\bar{\mathbf{X}}^T\mathbf{y} \triangleq \mathbf{b}$ nghĩa là *đặt* $\bar{\mathbf{X}}^T\mathbf{y}$ *bằng* $\mathbf{b}$).

Nếu ma trận vuông $\mathbf{A} \triangleq \bar{\mathbf{X}}^T\bar{\mathbf{X}}$ khả nghịch (non-singular hay inversable) thì phương trình (4) có nghiệm duy nhất: $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$.

Vậy nếu ma trận $\mathbf{A}$ không khả nghịch (có định thức bằng 0) thì sao? Nếu các bạn vẫn nhớ các kiến thức về hệ phương trình tuyến tính, trong trường hợp này thì hoặc phương trinh (3.4) vô nghiệm, hoặc là nó có vô số nghiệm. Khi đó, chúng ta sử dụng khái niệm *giả nghịch đảo* $\mathbf{A}^\dagger$ (đọc là *A dagger* trong tiếng Anh). (*Giả nghịch đảo (pseudo inverse) là trường hợp tổng quát của nghịch đảo khi ma trận không khả nghịch hoặc thậm chí không vuông. Trong khuôn khổ bài viết này, tôi xin phép được lược bỏ phần này, nếu các bạn thực sự quan tâm, tôi sẽ viết một bài khác chỉ nói về giả nghịch đảo. Xem thêm: Least Squares, Pseudo-Inverses, PCA & SVD.*)

Với khái niệm giả nghịch đảo, điểm tối ưu của bài toán Linear Regression có dạng:

$$\mathbf{w} = \mathbf{A}^\dagger\mathbf{b} = (\bar{\mathbf{X}}^T\bar{\mathbf{X}})^\dagger\bar{\mathbf{X}}^T\mathbf{y} \tag{3.5}$$

3.3 Ví dụ trên Python

3.3.1 Bài toán

Trong phần này, tôi sẽ chọn một ví dụ đơn giản về việc giải bài toán Linear Regression trong Python. Tôi cũng sẽ so sánh nghiệm của bài toán khi giải theo phương trình (3.5) và nghiệm tìm được khi dùng thư viện scikit-learn của Python. (*Đây là thư viện Machine Learning được sử dụng rộng rãi trong Python*). Trong ví dụ này, dữ liệu đầu vào chỉ có 1 giá trị (1 chiều) để thuận tiện cho việc minh hoạ trong mặt phẳng.

Chúng ta có 1 bảng dữ liệu về chiều cao và cân nặng của 15 người như trong Bảng 3.1.

Bài toán đặt ra là: liệu có thể dự đoán cân nặng của một người dựa vào chiều cao của họ không? (*Trên thực tế, tất nhiên là không, vì cân nặng còn phụ thuộc vào nhiều yếu tố khác nữa, thể tích chẳng hạn*). Vì blog này nói về các thuật toán Machine Learning đơn giản nên tôi sẽ giả sử rằng chúng ta có thể dự đoán được.

Table 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		

Chúng ta có thể thấy là cân nặng sẽ tỉ lệ thuận với chiều cao (càng cao càng nặng), nên có thể sử dụng Linear Regression model cho việc dự đoán này. Để kiểm tra độ chính xác của model tìm được, chúng ta sẽ giữ lại cột 155 và 160 cm để kiểm thử, các cột còn lại được sử dụng để huấn luyện (train) model.

3.3.2 Hiển thị dữ liệu trên đồ thị

Trước tiên, chúng ta cần có hai thư viện numpy cho đại số tuyến tính và matplotlib cho việc vẽ hình.

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import numpy as np
import matplotlib.pyplot as plt
```

Tiếp theo, chúng ta khai báo và biểu diễn dữ liệu trên một đồ thị.

```
# height (cm)
X = np.array([[147, 150, 153, 158, 163, 165, 168, 170, 173, 175, 178, 180, 183]]).T
# weight (kg)
y = np.array([[ 49, 50, 51,  54, 58, 59, 60, 62, 63, 64, 66, 67, 68]]).T
# Visualize data
plt.plot(X, y, 'ro')
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

Từ đồ thị trong Hình 3.1 ta thấy rằng dữ liệu được sắp xếp gần như theo 1 đường thẳng, vậy mô hình Linear Regression nhiều khả năng sẽ cho kết quả tốt:

(cân nặng) = `w_1`*(chiều cao) + `w_0`

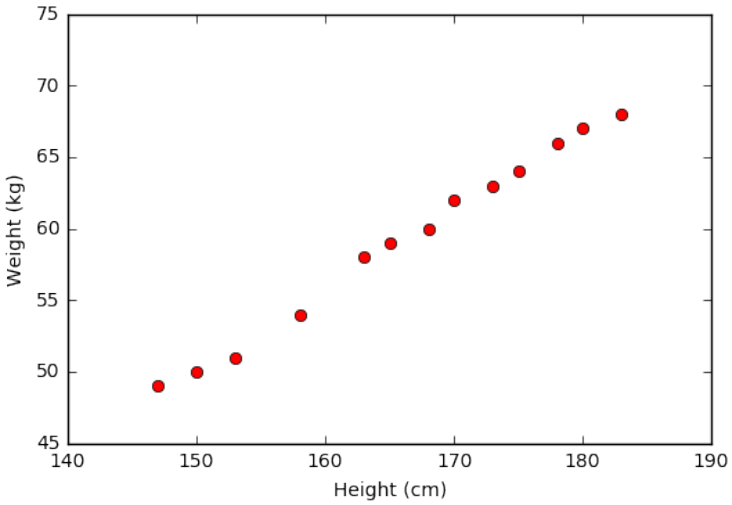

Fig. 3.1: Phân bố của dữ liệu trong ví dụ có dạng đường thẳng

3.3.3 Nghiệm theo công thức

Tiếp theo, chúng ta sẽ tính toán các hệ số `w_1` và `w_0` dựa vào công thức (5). Chú ý: giả nghịch đảo của một ma trận `A` trong Python sẽ được tính bằng `numpy.linalg.pinv(A)`, `pinv` là từ viết tắt của *pseudo inverse*.

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

# Calculating weights of the fitting line
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w = np.dot(np.linalg.pinv(A), b)
print('w = ', w)
# Preparing the fitting line
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(145, 185, 2)
y0 = w_0 + w_1*x0

# Drawing the fitting line
plt.plot(X.T, y.T, 'ro')     # data
plt.plot(x0, y0)               # the fitting line
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

```
w =  [[-33.73541021]
 [  0.55920496]]
```

Từ đồ thị bên trên ta thấy rằng các điểm dữ liệu màu đỏ nằm khá gần đường thẳng dự đoán màu xanh. Vậy mô hình Linear Regression hoạt động tốt với tập dữ liệu *training*. Bây giờ,

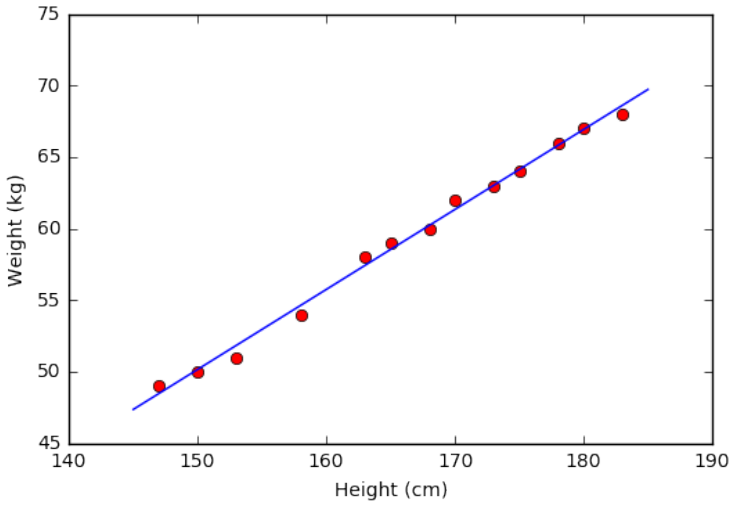

Fig. 3.2: Minh họa nghiệm của ví dụ

chúng ta sử dụng mô hình này để dự đoán cân nặng của hai người có chiều cao 155 và 160 cm mà chúng ta đã không dùng khi tính toán nghiệm.

```
y1 = w_1*155 + w_0
y2 = w_1*160 + w_0

print( 'Predict weight of person 155 cm: %.2f (kg), real number: 52 (kg)'  %(y1) )
print( 'Predict weight of person 160 cm: %.2f (kg), real number: 56 (kg)'  %(y2) )
```

```
Predict weight of person 155 cm: 52.94 (kg), real number: 52 (kg)
Predict weight of person 160 cm: 55.74 (kg), real number: 56 (kg)
```

Chúng ta thấy rằng kết quả dự đoán khá gần với số liệu thực tế.

3.3.4 Nghiệm theo thư viện scikit-learn

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

```
from sklearn import datasets, linear_model

# fit the model by Linear Regression
regr = linear_model.LinearRegression(fit_intercept=False) # fit_intercept = False for
    calculating the bias
regr.fit(Xbar, y)

# Compare two results
print( 'Solution found by scikit-learn  : ', regr.coef_ )
print( 'Solution found by (5): ', w.T)
```

```
    Solution found by scikit-learn  :  [[  -33.73541021 0.55920496]]
    Solution found by (5):  [[  -33.73541021 0.55920496 ]]
```

Chúng ta thấy rằng hai kết quả thu được như nhau! (*Nghĩa là tôi đã không mắc lỗi nào trong cách tìm nghiệm ở phần trên*)

Source code Jupyter Notebook cho bài này.

3.4 Thảo luận

3.4.1 Các bài toán có thể giải bằng Linear Regression

Hàm số $y \approx f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ là một hàm tuyến tính theo cả $\mathbf{w}$ và $\mathbf{x}$. Trên thực tế, Linear Regression có thể áp dụng cho các mô hình chỉ cần tuyến tính theo $\mathbf{w}$. Ví dụ:

$$y \approx w_1x_1 + w_2x_2 + w_3x_1^2 + w_4\sin(x_2) + w_5x_1x_2 + w_0$$

là một hàm tuyến tính theo $\mathbf{w}$ và vì vậy cũng có thể được giải bằng Linear Regression. Với mỗi dữ liệu đầu vào $\mathbf{x} = [x_1; x_2]$, chúng ta tính toán dữ liệu mới $\tilde{\mathbf{x}} = [x_1, x_2, x_1^2, \sin(x_2), x_1x_2]$ (đọc là x *tilde* trong tiếng Anh) rồi áp dụng Linear Regression với dữ liệu mới này.

Xem thêm ví dụ về Quadratic Regression (Hồi Quy Bậc Hai).

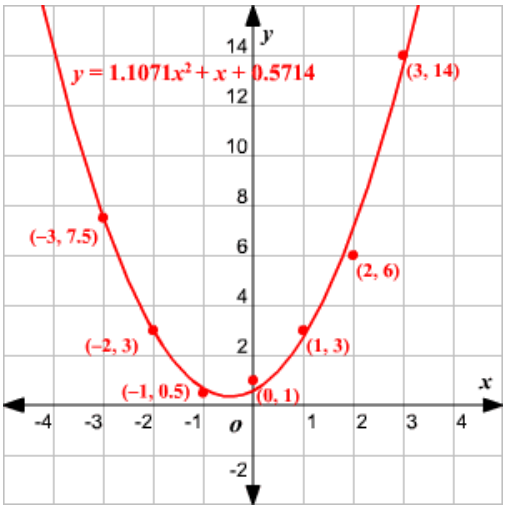

Fig. 3.3: Quadratic Regression (Nguồn: Quadratic Regression)

3.4.2 Hạn chế của Linear Regression

Hạn chế đầu tiên của Linear Regression là nó rất **nhạy cảm với nhiễu** (sensitive to noise). 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 3.4).

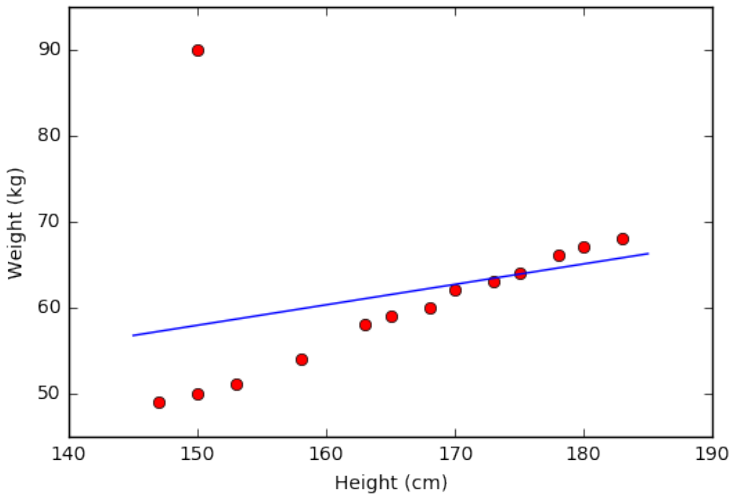

Fig. 3.4: Linear Regression rất nhạy cảm với nhiễu

Vì vậy, trước khi thực hiện Linear Regression, các nhiễu (*outlier*) cần phải được loại bỏ. Bước này được gọi là tiền xử lý (pre-processing).

Hạn chế thứ hai của Linear Regression là nó **không biểu diễn được các mô hình phức tạp**. Mặc dù trong phần trên, chúng ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa *outcome* và *input* không nhất thiết phải là tuyến tính, nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế. Hơn nữa, chúng ta sẽ tự hỏi: làm thế nào để xác định được các hàm $x_1^2, \sin(x_2), x_1 x_2$ như ở trên?!

3.4.3 Các phương pháp tối ưu

Linear Regression là một mô hình đơn giản, lời giải cho phương trình đạo hàm bằng 0 cũng khá đơn giản. *Trong hầu hết các trường hợp, chúng ta không thể giải được phương trình đạo hàm bằng 0.*

Nhưng có một điều chúng ta nên nhớ, **còn tính được đạo hàm là còn có cơ hội**.

3.5 Tài liệu tham khảo

1. Linear Regression - Wikipedia
2. Simple Linear Regression Tutorial for Machine Learning
3. Least Squares, Pseudo-Inverses, PCA & SVD

4

K-means Clustering

4.1 Giới thiệu

Trong bài trước, chúng ta đã làm quen với thuật toán Linear Regression - là thuật toán đơn giản nhất trong Supervised learning. Bài này tôi sẽ giới thiệu một trong những thuật toán cơ bản nhất trong Unsupervised learning - thuật toán K-means clustering (phân cụm K-means).

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho *dữ liệu trong cùng một cụm có tính chất giống nhau.*

Ví dụ: Một công ty muốn tạo ra những 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 năm 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 đó có rất nhiều dữ liệu của rất nhiều khách hàng nhưng chưa có cách nào chia toàn bộ khách hàng đó thành một số nhóm/cụm khác nhau. Nếu một người biết Machine Learning được đặt câu hỏi này, phương pháp đầu tiên anh (chị) ta nghĩ đến sẽ là K-means Clustering. Vì nó là một trong những thuật toán đầu tiên mà anh ấy tìm được trong các cuốn sách, khóa học về Machine Learning. Và tôi cũng chắc rằng anh ấy đã đọc blog Machine Learning cơ bản. Sau khi đã phân ra được từng nhóm, nhân viên công ty đó có thể lựa chọn ra một vài khách hàng trong mỗi nhóm để quyết định xem 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 rất nhiều.

Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm *ở gần nhau trong một không gian nào đó* (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn). Hình bên dưới là một ví dụ về 3 cụm dữ liệu (từ giờ rôi sẽ viết gọn là *cluster*).

Giả sử mỗi cluster có một điểm đại diện (*center*) màu vàng. Và những điểm xung quanh mỗi center thuộc vào cùng nhóm với center đó. Một cách đơn giản nhất, xét một điểm bất kỳ, ta xét xem điểm đó gần với center nào nhất thì nó thuộc về cùng nhóm với center đó.

Tới đây, chúng ta có một bài toán thú vị: *Trên một vùng biển hình vuông lớn có ba đảo hình vuông, tam giác, và tròn màu vàng như hình trên. Một điểm trên biển được gọi là thuộc lãnh hải của một đảo nếu nó nằm gần đảo này hơn so với hai đảo kia . Hãy xác định ranh giới lãnh hải của các đảo.*

Hình dưới đây là một hình minh họa cho việc phân chia lãnh hải nếu có 5 đảo khác nhau được biểu diễn bằng các hình tròn màu đen:

Chúng ta thấy rằng đường phân định giữa các lãnh hải là các đường thẳng (chính xác hơn thì chúng là các đường trung trực của các cặp điểm gần nhau). Vì vậy, lãnh hải của một đảo sẽ là một hình đa giác.

Cách phân chia này trong toán học được gọi là Voronoi Diagram.

Trong không gian ba chiều, lấy ví dụ là các hành tinh, thì (tạm gọi là) *lãnh không* của mỗi hành tinh sẽ là một đa diện. Trong không gian nhiều chiều hơn, chúng ta sẽ có những thứ (mà tôi gọi là) *siêu đa diện* (hyperpolygon).

Quay lại với bài toán phân nhóm và cụ thể là thuật toán K-means clustering, chúng ta cần một chút phân tích toán học trước khi đi tới phần tóm tắt thuật toán ở phần dưới. Nếu bạn không muốn đọc quá nhiều về toán, bạn có thể bỏ qua phần này. (*Tốt nhất là đừng bỏ qua, bạn sẽ tiếc đấy*).

4.2 Phân tích toán học

Mục đích cuối cùng của thuật toán phân nhóm này là: từ dữ liệu đầu vào và số lượng nhóm chúng ta muốn tìm, hãy chỉ ra center của mỗi nhóm và phân các điểm dữ liệu vào các nhóm tương ứng. Giả sử thêm rằng mỗi điểm dữ liệu chỉ thuộc vào đúng một nhóm.

4.2.1 Một số ký hiệu toán học

Giả sử có N điểm dữ liệu là $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ và $K < N$ là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K \in \mathbb{R}^{d \times 1}$ và label của mỗi điểm dữ liệu.

Lưu ý về ký hiệu toán học: *trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ x_1, N, y, k. Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ $\mathbf{m}, \mathbf{x}_1$. Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ $\mathbf{X}, \mathbf{M}, \mathbf{Y}$. Lưu ý này đã được nêu ở bài Linear Regression. Tôi xin được không nhắc lại trong các bài tiếp theo.*

Với mỗi điểm dữ liệu $\mathbf{x}_i$ đặt $\mathbf{y}_i = [y_{i1}, y_{i1}, \ldots, y_{iK}]$ là label vector của nó, trong đó nếu $\mathbf{x}_i$ được phân vào cluster k thì $y_{ik} = 1$ và $y_{ij} = 0, \forall j \neq k$. Điều này có nghĩa là có đúng một phần tử của vector $\mathbf{y}_i$ là bằng 1 (tương ứng với cluster của $\mathbf{x}_i$), các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là $[1, 0, 0, \ldots, 0]$ thì nó thuộc vào cluster 1, là $[0, 1, 0, \ldots, 0]$ thì nó thuộc vào cluster 2, ... Cách mã hóa label của dữ liệu như thế này được goi là biểu diễn *one-hot*. Chúng ta sẽ thấy cách biểu diễn one-hot này rất phổ biến trong Machine Learning ở các bài tiếp theo.

Ràng buộc của $\mathbf{y}_i$ có thể viết dưới dạng toán học như sau:

$$y_{ik} \in 0, 1, \quad \sum_{k=1}^{K} y_{ik} = 1 \quad (1) \tag{4.1}$$

4.2.2 Hàm mất mát và bài toán tối ưu

Nếu ta coi center $\mathbf{m}_k$ là center (hoặc representative) của mỗi cluster và *ước lượng* tất cả các điểm được phân vào cluster này bởi $\mathbf{m}_k$, thì một điểm dữ liệu $\mathbf{x}_i$ được phân vào cluster k sẽ bị sai số là $(\mathbf{x}_i - \mathbf{m}_k)$. Chúng ta mong muốn sai số này có trị tuyệt đối nhỏ nhất nên (giống như trong bài Linear Regression) ta sẽ tìm cách để đại lượng sau đây đạt giá trị nhỏ nhất:

$$\|\mathbf{x}_i - \mathbf{m}_k\|_2^2 \tag{4.2}$$

Hơn nữa, vì $\mathbf{x}_i$ được phân vào cluster k nên $y_{ik} = 1, y_{ij} = 0, \ \forall j \neq k$. Khi đó, biểu thức bên trên sẽ được viết lại là:

$$y_{ik}\|\mathbf{x}_i - \mathbf{m}_k\|_2^2 = \sum_{j=1}^{K} y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \tag{4.3}$$

(*Hy vọng chỗ này không quá khó hiểu*)

Sai số cho toàn bộ dữ liệu sẽ là:

$$\mathcal{L}(\mathbf{Y}, \mathbf{M}) = \sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \tag{4.4}$$

Trong đó $\mathbf{Y} = [\mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_N]$, $\mathbf{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots \mathbf{m}_K]$ lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. Hàm số mất mát trong bài toán K-means clustering của chúng ta là hàm $\mathcal{L}(\mathbf{Y}, \mathbf{M})$ với ràng buộc như được nêu trong phương trình (1).

Tóm lại, chúng ta cần tối ưu bài toán sau:

$$\mathbf{Y}, \mathbf{M} = \arg\min_{\mathbf{Y},\mathbf{M}} \sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (2) \tag{4.5}$$

$$\text{subject to: } y_{ij} \in 0, 1 \ \ \forall i, j; \quad \sum_{j=1}^{K} y_{ij} = 1 \ \ \forall i \tag{4.6}$$

(*subject to* nghĩa là *thỏa mãn điều kiện*).

Nhắc lại khái niệm arg min: Chúng ta biết ký hiệu min là *giá trị nhỏ nhất của hàm số*, arg min chính là *giá trị của biến số để hàm số đó đạt giá trị nhỏ nhất đó*. Nếu $f(x) = x^2 - 2x + 1 = (x-1)^2$ thì giá trị nhỏ nhất của hàm số này bằng 0, đạt được khi $x = 1$. Trong ví dụ này $\min_x f(x) = 0$ và $\arg\min_x f(x) = 1$. Thêm ví dụ khác, nếu $x_1 = 0, x_2 = 10, x_3 = 5$ thì ta nói $\arg\min_i x_i = 1$ vì 1 là chỉ số để x_i đạt giá trị nhỏ nhất (bằng 0). Biến số viết bên dưới min là biến số cúng ta cần tối ưu. Trong các bài toán tối ưu, ta thường quan tâm tới arg min hơn là min.

4.2.3 Thuật toán tối ưu hàm mất mát

Bài toán (2) là một bài toán khó tìm *điểm tối ưu* vì nó có thêm các điều kiện ràng buộc. *Bài toán này thuộc loại mix-integer programming (điều kiện biến là số nguyên) - là loại rất khó tìm nghiệm tối ưu toàn cục (global optimal point, tức nghiệm làm cho hàm mất mát đạt giá trị nhỏ nhất có thể).* Tuy nhiên, trong một số trường hợp chúng ta vẫn có thể tìm được phương pháp để tìm được nghiệm gần đúng hoặc điểm cực tiểu. (*Nếu chúng ta vẫn nhớ chương trình toán ôn thi đại học thì điểm cực tiểu chưa chắc đã phải là điểm làm cho hàm số đạt giá trị nhỏ nhất*).

Một cách đơn giản để giải bài toán (2) là xen kẽ giải $\mathbf{Y}$ và $\mathbf{M}$ khi biến còn lại được cố định. Đây là một thuật toán lặp, cũng là kỹ thuật phổ biến khi giải bài toán tối ưu. Chúng ta sẽ lật lượt giải quyết hai bài toán sau đây:

Cố định M, tìm Y

Giả sử đã tìm được các centers, hãy tìm các label vector để hàm mất mát đạt giá trị nhỏ nhất. Điều này tương đương với việc tìm cluster cho mỗi điểm dữ liệu.

Khi các centers là cố định, bài toán tìm label vector cho toàn bộ dữ liệu có thể được chia nhỏ thành bài toán tìm label vector cho từng điểm dữ liệu $\mathbf{x}_i$ như sau:

$$\mathbf{y}_i = \arg\min_{\mathbf{y}_i} \sum_{j=1}^{K} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \quad (3) \tag{4.7}$$

$$\text{subject to: } y_{ij} \in 0, 1 \ \ \forall j; \quad \sum_{j=1}^{K} y_{ij} = 1 \tag{4.8}$$

Vì chỉ có một phần tử của label vector $\mathbf{y}_i$ bằng 1 nên bài toán (3) có thể tiếp tục được viết dưới dạng đơn giản hơn:

$$j = \arg\min_j \|\mathbf{x}_i - \mathbf{m}_j\|_2^2 \tag{4.9}$$

Vì $\|\mathbf{x}_i - \mathbf{m}_j\|_2^2$ chính là bình phương khoảng cách tính từ điểm $\mathbf{x}_i$ tới center $\mathbf{m}_j$, ta có thể kết luận rằng **mỗi điểm $\mathbf{x}_i$ thuộc vào cluster có center gần nó nhất**! Từ đó ta có thể dễ dàng suy ra label vector của từng điểm dữ liệu.

Cố định Y, tìm M

Giả sử đã tìm được cluster cho từng điểm, hãy tìm center mới cho mỗi cluster để hàm mất mát đạt giá trị nhỏ nhất.

Một khi chúng ta đã xác định được label vector cho từng điểm dữ liệu, bài toán tìm center cho mỗi cluster được rút gọn thành:

$$\mathbf{m}_j = \arg\min_{\mathbf{m}_j} \sum_{i=1}^{N} y_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|_2^2. \tag{4.10}$$

Tới đây, ta có thể tìm nghiệm bằng phương pháp giải đạo hàm bằng 0, vì hàm cần tối ưu là một hàm liên tục và có đạo hàm xác định tại mọi điểm. *Và quan trọng hơn, hàm này là hàm convex (lồi) theo $\mathbf{m}_j$ nên chúng ta sẽ tìm được giá trị nhỏ nhất và điểm tối ưu tương ứng. Sau này nếu có dịp, tôi sẽ nói thêm về tối ưu lồi (convex optimization) - một mảng cực kỳ quan trọng trong toán tối ưu.*

Đặt $l(\mathbf{m}_j)$ là hàm bên trong dấu arg min, ta có đạo hàm:

$$\frac{\partial l(\mathbf{m}_j)}{\partial \mathbf{m}_j} = 2 \sum_{i=1}^{N} y_{ij} (\mathbf{m}_j - \mathbf{x}_i) \tag{4.11}$$

Giải phương trình đạo hàm bằng 0 ta có:

$$\mathbf{m}_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij} \mathbf{x}_i \tag{4.12}$$

$$\Rightarrow \mathbf{m}_j = \frac{\sum_{i=1}^{N} y_{ij} \mathbf{x}_i}{\sum_{i=1}^{N} y_{ij}} \tag{4.13}$$

Nếu để ý một chút, chúng ta sẽ thấy rằng mẫu số chính là phép đếm *số lượng các điểm dữ liệu* trong cluster j (*Bạn có nhận ra không?*). Còn tử số chính là *tổng các điểm dữ liệu* trong

cluster j. (*Nếu bạn đọc vẫn nhớ điều kiện ràng buộc của các y_{ij} thì sẽ có thể nhanh chóng nhìn ra điều này*).

Hay nói một cách đơn giản hơn nhiều: **$\mathbf{m}_j$ là trung bình cộng của các điểm trong cluster j.**

Tên gọi *K-means clustering* cũng xuất phát từ đây.

4.2.4 Tóm tắt thuật toán

Tới đây tôi xin được tóm tắt lại thuật toán (*đặc biệt quan trọng với các bạn bỏ qua phần toán học bên trên*) như sau:

Đầu vào: Dữ liệu $\mathbf{X}$ và số lượng cluster cần tìm K.

Đầu ra: Các center $\mathbf{M}$ và label vector cho từng điểm dữ liệu $\mathbf{Y}$.

1. Chọn K điểm bất kỳ làm các center ban đầu. 2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất. 3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán. 4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cluster đó sau bước 2. 5. Quay lại bước 2.

Chúng ta có thể đảm bảo rằng thuật toán sẽ dừng lại sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Theo kiến thức về dãy số trong chương trình cấp 3: *nếu một dãy số giảm và bị chặn dưới thì nó hội tụ!* Hơn nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

Chúng ta sẽ có một vài thảo luận về thuật toán này, về những hạn chế và một số phương pháp khắc phục. Nhưng trước hết, hãy xem nó thể hiện như thế nào trong một ví dụ cụ thể dưới đây.

4.3 Ví dụ trên Python

4.3.1 Giới thiệu bài toán

Để kiểm tra mức độ hiểu quả của một thuật toán, chúng ta sẽ làm một ví dụ đơn giản (thường được gọi là *toy example*). Trước hết, chúng ta chọn center cho từng cluster và tạo dữ liệu cho từng cluster bằng cách lấy mẫu theo phân phối chuẩn có kỳ vọng là center của cluster đó và ma trận hiệp phương sai (covariance matrix) là ma trận đơn vị.

Trước tiên, chúng ta cần khai báo các thư viện cần dùng. Chúng ta cần `numpy` và `matplotlib` như trong bài Linear Regression cho việc tính toán ma trận và hiển thị dữ liệu. Chúng ta cũng cần thêm thư viện `scipy.spatial.distance` để tính khoảng cách giữa các cặp điểm trong hai tập hợp một cách hiệu quả.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(11)
```

Tiếp theo, ta tạo dữ liệu bằng cách lấy các điểm theo phân phối chuẩn có kỳ vọng tại các điểm có tọa độ `(2, 2)`, `(8, 3)` và `(3, 6)`, ma trận hiệp phương sai giống nhau và là ma trận đơn vị. Mỗi cluster có 500 điểm. (*Chú ý rằng mỗi điểm dữ liệu là một hàng của ma trận dữ liệu.*

```
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

original_label = np.asarray([0]*N + [1]*N + [2]*N).T
```

4.3.2 Hiển thị dữ liệu trên đồ thị

Chúng ta cần một hàm `kmeans_display` để hiển thị dữ liệu. Sau đó hiển thị dữ liệu theo nhãn ban đầu.

```
def kmeans_display(X, label):
    K = np.amax(label) + 1
    X0 = X[label == 0, :]
    X1 = X[label == 1, :]
    X2 = X[label == 2, :]

    plt.plot(X0[:, 0], X0[:, 1], 'b^', markersize = 4, alpha = .8)
    plt.plot(X1[:, 0], X1[:, 1], 'go', markersize = 4, alpha = .8)
    plt.plot(X2[:, 0], X2[:, 1], 'rs', markersize = 4, alpha = .8)

    plt.axis('equal')
    plt.plot()
    plt.show()

kmeans_display(X, original_label)
```

Trong đồ thị trên, mỗi cluster tương ứng với một màu. Có thể nhận thấy rằng có một vài điểm màu đỏ bị lẫn sang phần cluster màu xanh.

4.3.3 Các hàm số cần thiết cho K-means clustering

Viết các hàm:

1. `kmeans_init_centers` để khởi tạo các centers ban đầu. 2. `kmeans_asign_labels` để gán nhán mới cho các điểm khi biết các centers. 3. `kmeans_update_centers` để cập nhật các centers mới dữa trên dữ liệu vừa được gán nhãn. 4. `has_converged` để kiểm tra điều kiện dừng của thuật toán.

```
def kmeans_init_centers(X, k):
    # randomly pick k rows of X as initial centers
    return X[np.random.choice(X.shape[0], k, replace=False)]

def kmeans_assign_labels(X, centers):
    # calculate pairwise distances btw data and centers
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)

def kmeans_update_centers(X, labels, K):
    centers = np.zeros((K, X.shape[1]))
    for k in range(K):
        # collect all points assigned to the k-th cluster
        Xk = X[labels == k, :]
        # take average
        centers[k,:] = np.mean(Xk, axis = 0)
    return centers

def has_converged(centers, new_centers):
    # return True if two sets of centers are the same
    return (set([tuple(a) for a in centers]) ==
        set([tuple(a) for a in new_centers]))
```

Phần chính của K-means clustering:

```
def kmeans(X, K):
    centers = [kmeans_init_centers(X, K)]
    labels = []
    it = 0
    while True:
        labels.append(kmeans_assign_labels(X, centers[-1]))
        new_centers = kmeans_update_centers(X, labels[-1], K)
        if has_converged(centers[-1], new_centers):
            break
        centers.append(new_centers)
        it += 1
    return (centers, labels, it)
```

Áp dụng thuật toán vừa viết vào dữ liệu ban đầu, hiển thị kết quả cuối cùng.

```
(centers, labels, it) = kmeans(X, K)
print 'Centers found by our algorithm:'
print centers[-1]

kmeans_display(X, labels[-1])
```

Centers found by our algorithm: [[1.97563391 2.01568065] [8.03643517 3.02468432] [2.99084705 6.04196062]]

Từ kết quả này chúng ta thấy rằng thuật toán K-means clustering làm việc khá thành công, các centers tìm được khá gần với kỳ vọng ban đầu. Các điểm thuộc cùng một cluster hầu như được phân vào cùng một cluster (trừ một số diểm màu đỏ ban đầu đã bị phân nhầm vào cluster màu xanh da trời, nhưng tỉ lệ là nhỏ và có thể chấp nhận được).

Dưới đây là hình ảnh động minh họa thuật toán qua từng vòng lặp, chúng ta thấy rằng thuật toán trên hội tụ rất nhanh, chỉ cần 6 vòng lặp để có được kết quả cuối cùng: <img src ="/assets/kmeans/kmeans11.gif" align = "center">

Các bạn có thể xem thêm các trang web minh họa thuật toán K-means cluster tại:

1. Visualizing K-Means Clustering
2. Visualizing K-Means Clustering - Standford

4.3.4 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`.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print 'Centers found by scikit-learn:'
print kmeans.cluster_centers_
pred_label = kmeans.predict(X)
kmeans_display(X, pred_label)
```

Centers found by scikit-learn: [[8.0410628 3.02094748] [2.99357611 6.03605255] [1.97634981 2.01123694]]

Thật may mắn (*cho tôi*), hai thuật toán cho cùng một đáp số! Với cách thứ nhất, tôi mong muốn các bạn hiểu rõ được thuật toán K-means clustering làm việc như thế nào. Với cách thứ hai, tôi hy vọng các bạn biết áp dụng thư viện sẵn có như thế nào.

4.4 Thảo luận

4.4.1 Hạn chế

Có một vài hạn chế của thuật toán K-means clustering:

Chúng ta cần biết số lượng cluster cần clustering

Để ý thấy rằng trong thuật toán nêu trên, chúng ta cần biết đại lượng K là số lượng clusters. Trong thực tế, nhiều trường hợp chúng ta không xác định được giá trị này. Có một số phương pháp giúp xác định số lượng clusters, tôi sẽ dành thời gian nói về chúng sau nếu có dịp. Bạn đọc có thể tham khảo Elbow method - Determining the number of clusters in a data set.

Nghiệm cuối cùng phụ thuộc vào các centers được khởi tạo ban đầu

Tùy vào các center ban đầu mà thuật toán có thể có tốc độ hội tụ rất chậm, ví dụ: hoặc thậm chí cho chúng ta nghiệm không chính xác (chỉ là local minimum - điểm cực tiểu - mà không phải giá trị nhỏ nhất):

Có một vài cách khắc phục đó là:

- Chạy K-means clustering nhiều lần với các center ban đầu khác nhau rồi chọn cách có hàm mất mát cuối cùng đạt giá trị nhỏ nhất.
- K-means++ -Improve initialization algorithm - wiki.
- Bạn nào muốn tìm hiểu sâu hơn có thể xem bài báo khoa học Cluster center initialization algorithm for K-means clustering.

Các cluster cần có só lượng điểm gần bằng nhau

Dưới đây là một ví dụ với 3 cluster với 20, 50, và 1000 điểm. Kết quả cuối cùng không chính xác.

Các cluster cần có dạng hình tròn

Tức các cluster tuân theo phân phối chuẩn và ma trận hiệp phương sai là ma trận đường chéo có các điểm trên đường chéo giống nhau.

Dưới đây là 1 ví dụ khi 1 cluster có dạng hình dẹt.

Khi một cluster nằm phía trong 1 cluster khác

Đây là ví dụ kinh điển về việc K-means clustering không thể phân cụm dữ liệu. Một cách tự nhiên, chúng ta sẽ phân ra thành 4 cụm: mắt trái, mắt phải, miệng, xunh quanh mặt. Nhưng vì mắt và miệng nằm trong khuôn mặt nên K-means clustering không thực hiện được:

Mặc dù có những hạn chế, K-means clustering vẫn cực kỳ quan trọng trong Machine Learning và là nền tảng cho nhiều thuật toán phức tạp khác sau này. Chúng ta cần bắt đầu từ những thứ đơn giản. *Simple is best!*

4.5 Tài liệu tham khảo

1. Clustering documents using k-means
2. Voronoi Diagram - Wikipedia
3. Cluster center initialization algorithm for K-means clustering
4. Visualizing K-Means Clustering
5. Visualizing K-Means Clustering - Standford

Index