

Vũ Hữu Tiệp

Machine Learning cơ bản

First Edition

March 1, 2017

Contents

1	Giới thiệu	1
1.1	Mục đích viết Blog	2
1.2	Tham khảo thêm	3
1.2.1	Các khóa học	3
1.2.2	Các trang Machine Learning tiếng Việt khác	3
2	Phân nhóm các thuật toán Machine Learning	4
2.1	1. Phân nhóm dựa trên phương thức học	4
2.1.1	Supervised Learning (Học có giám sát)	4
2.1.2	Unsupervised Learning (Học không giám sát)	6
2.1.3	Semi-Supervised Learning (Học bán giám sát)	6
2.1.4	Reinforcement Learning (Học củng cố)	7
2.2	2. Phân nhóm dựa trên chức năng	8
2.2.1	Regression Algorithms	8
2.2.2	Classification Algorithms	8
2.2.3	Instance-based Algorithms	8
2.2.4	Regularization Algorithms	8
2.2.5	Bayesian Algorithms	9

2.2.6	Clustering Algorithms	9
2.2.7	Artificial Neural Network Algorithms	9
2.2.8	Dimensionality Reduction Algorithms	9
2.2.9	Ensemble Algorithms	9
2.3	Tài liệu tham khảo	10
3	Linear Regression	11
3.1	Giới thiệu	11
3.2	Phân tích toán học	12
3.2.1	Dạng của Linear Regression	12
3.2.2	Sai số dự đoán	12
3.2.3	Hàm mất mát	13
3.2.4	Nghiệm cho bài toán Linear Regression	14
3.3	Ví dụ trên Python	15
3.3.1	Bài toán	15
3.3.2	Hiển thị dữ liệu trên đồ thị	15
3.3.3	Nghiệm theo công thức	16
3.3.4	Nghiệm theo thư viện scikit-learn	18
3.4	Thảo luận	18
3.4.1	Các bài toán có thể giải bằng Linear Regression	18
3.4.2	Hạn chế của Linear Regression	19
3.4.3	Các phương pháp tối ưu	19
3.5	Tài liệu tham khảo	19

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](#).

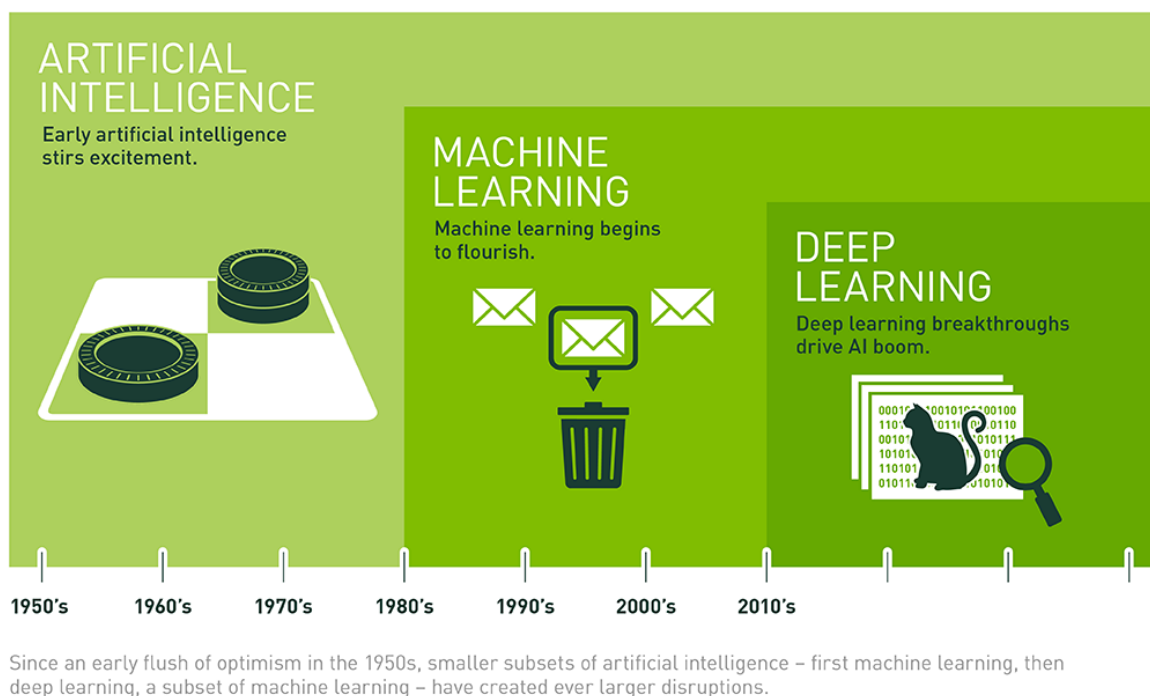


Fig. 1.1: Mối quan hệ giữa AI, Machine Learning và Deep Learning. Nguồn: [What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?](#)

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 thời, 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*

Phân nhóm các thuật toán Machine Learning

2.1 1. Phân nhóm dựa trên phương thức học

Theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Supervised learning, Unsupervised learning, Semi-supervised learning 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 ta có một tập hợp biến đầu vào $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ và một tập hợp nhãn tương ứng $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$, 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 training 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} :

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

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, 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.

```

<!--  -->
```

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 gắn 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 người* 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 xm^2 , có y phòng ngủ và cách trung tâm thành phố zkm 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 (dimension 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 Batman, 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 toán 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 Cường 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.

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 1: AlphaGo 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^{170} , 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](#)).

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.

Huấn luyện cho máy tính chơi game Mario

2.2 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 Representation-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 Shrinkage and Selection Operator (LASSO)
3. Least-Angle Regression (LARS)

2.2.5 Bayesian Algorithms

1. Naive Bayes
2. Gaussian 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 3. Tài liệu tham khảo

1. [A Tour of Machine Learning Algorithms](#)
2. [Điểm qua các thuật toán Machine Learning hiện đại](#)

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 m², 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:

$$y \approx f(\mathbf{x}) = \hat{y}$$
$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + w_3x_3 + w_0 \quad (1)$$

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 gọi 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*) $(\mathbf{x}_i, y_i), i = 1, 2, \dots, N$, 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:

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

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 (2). Đặt $\mathbf{y} = [y_1; y_2; \dots; y_N]$ là một vector cột chứa tất cả các *output* của *training data*; $\bar{\mathbf{X}} = [\bar{\mathbf{x}}_1; \bar{\mathbf{x}}_2; \dots; \bar{\mathbf{x}}_N]$ 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:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N (y_i - \bar{\mathbf{x}}_i \mathbf{w})^2 \\ &= \frac{1}{2} \|\mathbf{y} - \bar{\mathbf{X}} \mathbf{w}\|_2^2 \quad (3) \end{aligned}$$

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).

3.2.4 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:

$$\bar{\mathbf{X}}^T \bar{\mathbf{X}}\mathbf{w} = \bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b} \quad (4)$$

(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 trình (4) vô nghiệm, hoặc lập 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:

$$\mathbf{w} = \mathbf{A}^\dagger \mathbf{b} = (\bar{\mathbf{X}}^T \bar{\mathbf{X}})^\dagger \bar{\mathbf{X}}^T \mathbf{y} \quad (5)$$

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 (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 họa trong mặt phẳng.

Chúng ta có 1 bảng dữ liệu về chiều cao và cân nặng của 15 người như dưới đây:

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		

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.

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 Hiện thị dữ liệu trên đồ thị

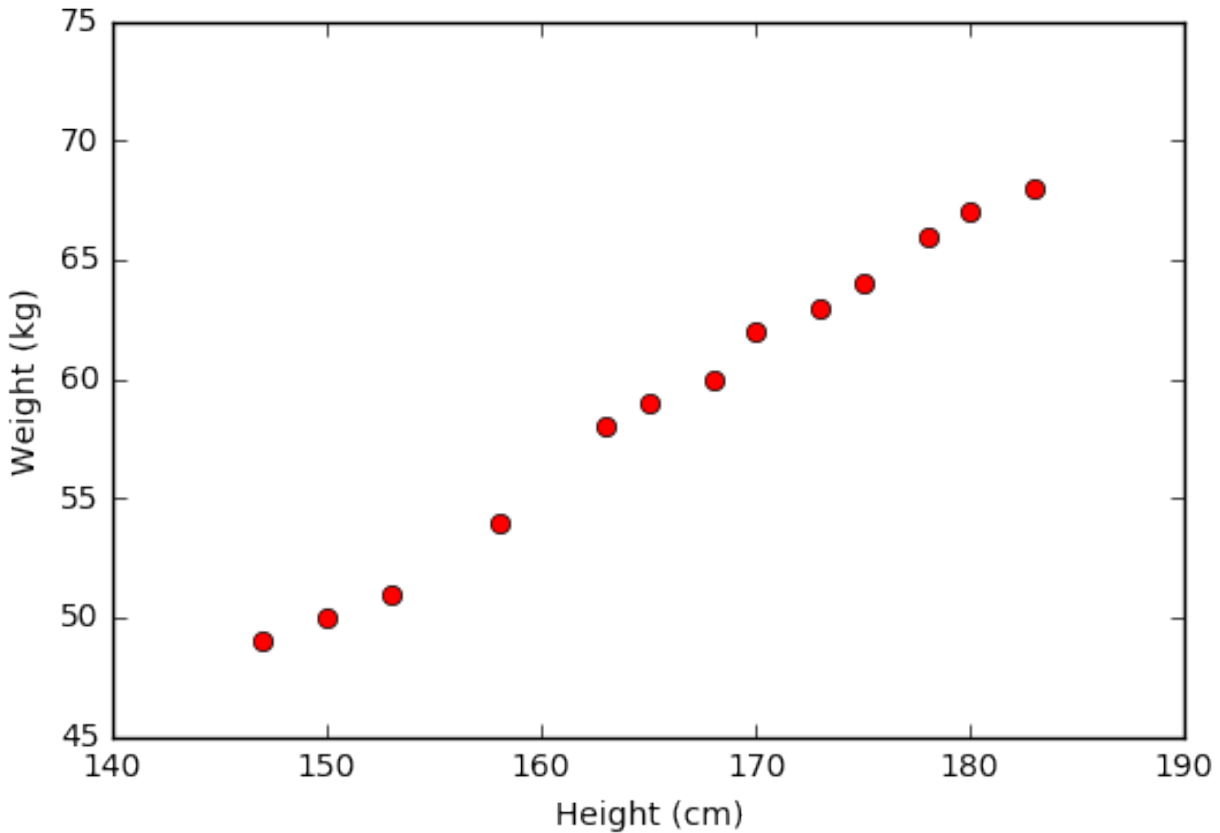
Trước tiên, chúng ta cần có hai thư viện [numpy](#) cho đại số tuyến tính và [matplotlib](#) cho việc vẽ hình.

```
# To support both python2 and python3
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
# Visualized 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 này 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:

$$(\text{cân nặng}) = w_1 * (\text{chiều cao}) + w_0$$

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

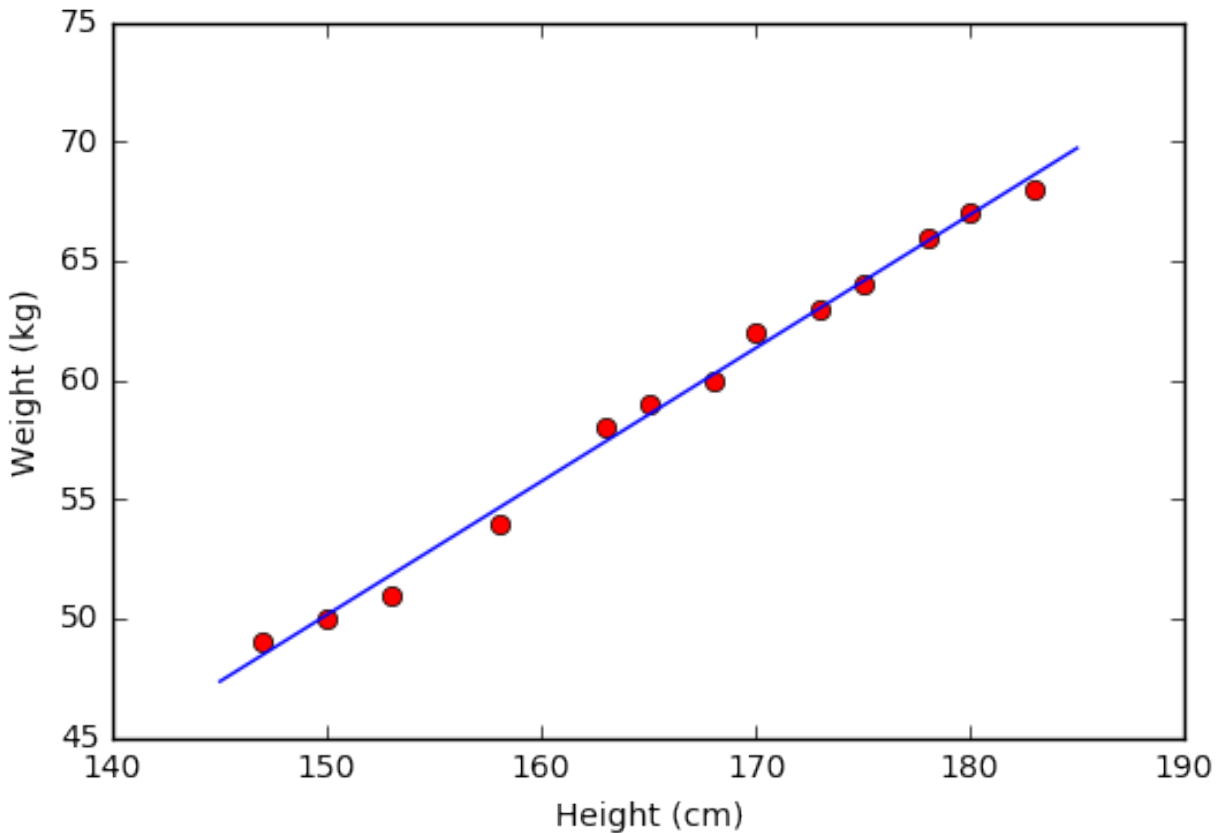
# Drawing the fitting line
plt.plot(X.T, y.T, 'ro') # data
plt.plot(x0, y0)         # the fitting line
plt.axis([140, 190, 45, 75])
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()

```

```

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

Predict weight of person with height 155 cm: 52.94 (kg), real number: 52 (kg) Predict weight of person with height 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: [[-33.735410210.55920496]]
Solution found by (5): [[-33.735410210.55920496]]
```

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:

$$y \approx w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 \sin(x_2) + w_5 x_1 x_2 + w_0$$

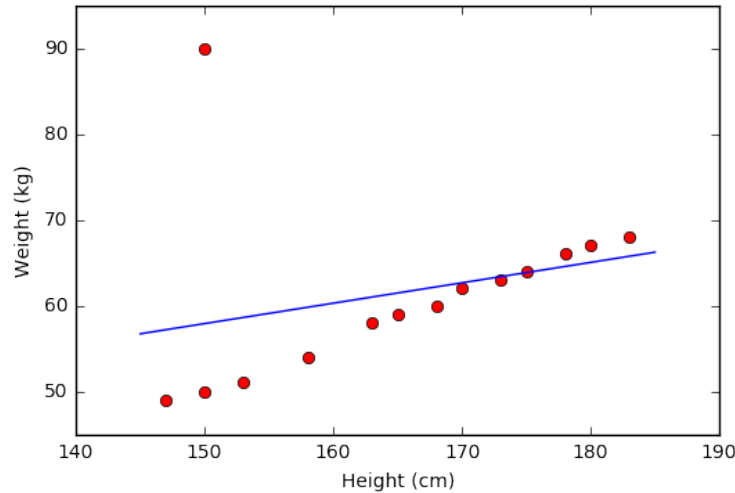
là một hàm tuyến tính theo \mathbf{w} và vì vậy cũng có thể được giải bằng Linear Regression. Với mỗi dữ liệu đầu vào $\mathbf{x} = [x_1; x_2]$, chúng ta tính toán dữ liệu mới $\tilde{\mathbf{x}} = [x_1, x_2, x_1^2, \sin(x_2), x_1 x_2]$ (đọc là *x tilde* trong tiếng Anh) rồi áp dụng Linear Regression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

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

a figure should be here

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 *nh nhiễu* (150 cm, 90kg) thì kết quả sẽ sai khác đi rất nhiều. Xem hình dưới đây:



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_1x_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](#)

```

import numpy as np

def incmatrix(genl1, genl2):
    m = len(genl1)
    n = len(genl2)
    M = None # to become the incidence matrix
    VT = np.zeros((n*m, 1), int) # dummy variable

    # compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2), 1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r, c] = np.where(M2 == M1[i, j])
            for k in range(len(r)):
                VT[(i)*n+r[k]] = 1;
                VT[(i)*n+c[k]] = 1;
                VT[(j)*n+r[k]] = 1;
                VT[(j)*n+c[k]] = 1;

            if M is None:
                M = np.copy(VT)
            else:
                M = np.concatenate((M, VT), 1)

            VT = np.zeros((n*m, 1), int)

    return M

```

Listing 3.1: Python example

asdf