BỘ QUỐC PHÒNG
TRƯỜNG ĐẠI HỌC THÔNG TIN LIÊN LẠC

KHÓA LUẬN TỐT NGHIỆP

Đề tài :

TÌM HIỂU VÀ VẬN DỤNG PHƯƠNG PHÁP XỬ LÝ DỮ LIỆU LỚN

GVHD: ThS. Cao Mạnh Hùng

SINH VIÊN: Trịnh Đình Phúc

LỚP: ĐHCN1A **KHOA:** CNTT

KHÓA HỌC: 2014 - 2018

Khánh Hòa, tháng 06 năm 2018

BỘ QUỐC PHÒNG
TRƯỜNG ĐẠI HỌC THÔNG TIN LIÊN LẠC

KHÓA LUẬN TỐT NGHIỆP

Đề tài :

TÌM HIỂU VÀ VẬN DỤNG PHƯƠNG PHÁP XỬ LÝ DỮ LIỆU LỚN

GVHD: ThS. Cao Mạnh Hùng

SINH VIÊN: Trịnh Đình Phúc

LỚP: ĐHCN1A **KHOA:** CNTT

KHÓA HỌC: 2014 - 2018

Khánh Hòa, tháng 06 năm 2018

LỜI CẢM ƠN

Trong thời gian làm đồ án tốt nghiệp, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến ThS. Cao Mạnh Hùng, giảng viên Bộ môn Kỹ Thuật Máy Tính - Trường ĐH Thông Tin Liên Lạc người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình làm khoá luận.

Em cũng xin chân thành cảm ơn các thầy cô giáo trong trường ĐH Thông Tin Liên Lạc đã cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập.

Cuối cùng, em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên em trong suốt quá trình học tập và hoàn thành khoá luận tốt nghiệp.

Em xin chân thành cảm ơn!

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH VẼ

BẢNG CHỮ VIẾT TẮT

Chữ viết tắt	Mô Tả
BoW	Bag-of-Word túi đựng từ
CRISP-DM	Cross-industry standard process for Data Mining Quy trình tiêu chuẩn liên ngành cho khai thác dữ liệu.
ETL	Extract: là đi thu gom dữ liệu từ nhiều nguồn khác. Transform: là chuyển đổi dữ liệu. Load: Sau khi được chuyển đổi thì toàn bộ các dữ liệu này được đưa vào một nơi lưu trữ mới gọi là DataWarehouse.
GD	Gradient Decent: Thuật toán xuống đồi.
TF-IDF	Term Frequency– Inverse Document Frequency là trọng số của một từ trong văn bản thu được qua thống kê thể hiện mức độ quan trọng của từ này trong một văn bản, mà bản thân văn bản đang xét nằm trong một tập hợp các văn bản.

MỤC LỤC

MỞ ĐẦU

Với sự bùng nổ của thời đại công nghệ thông tin như hiện nay thì việc bùng nổ dữ liệu cũng tăng theo. Dữ liệu chính là tài nguyên quý giá. Việc đưa ra các công cụ khai thác hiệu quả trên khối lượng lớn dữ liệu ấy nhằm rút trích các tri thức phục vụ cho nhu cầu sử dụng của con người là vô cùng cấp thiết.

Với sự lớn lên theo cấp số mũ của dữ liệu, các công cụ truyền thống nhằm rút trích tri thức hiện nay không còn hoạt động hiệu quả nữa. Cần có những phương pháp tiếp cận mới tốt hơn về mặt hiệu suất cũng như độ chính xác. Một trong những cách tiếp cận mới và đem lại hiệu quả cao là sử dụng Machine Learning tức máy học. Với khả năng "học" của máy dựa trên dữ liệu cho trước, con người có thể sử dụng Machine Learning để rút trích các tri thức quý giá từ dữ liệu.

Chính vì thế em đã chọn đề tài "TÌM HIỂU VÀ VẬN DỤNG PHƯƠNG PHÁP XỬ LÝ DỮ LIỆU LỚN" nhằm áp dụng quy trình xử lý dữ liệu vào thực tiễn, rút trích tri thức từ những dữ liệu.

Bài khóa luận tốt nghiệp sẽ được bắt đầu với cơ sở lý thuyết liên quan đến việc khai phá dữ liệu, các phương pháp tiếp cận bài toán phân tích dữ liệu, các quy trình nghiệp vụ xử lý dữ liệu và một số trường hợp gặp phải khi làm việc với dữ liệu lớn. Tiếp theo, trọng tâm sẽ là việc áp dụng một số mô hình Machine Learning phổ biến nhằm dự đoán, rút trích tri thức từ dữ liệu. Cuối cùng sẽ là các phương pháp để đánh giá độ chính xác cũng như hiệu suất của mô hình Machine Learning.

Chương 1
TỔNG QUAN VỀ PHƯƠNG PHÁP XỬ LÝ DỮ LIỆU LỚN

1.1. Tính thực tiễn

Trong những năm gần đây, với sự phát triển vượt bậc của Công Nghệ Thông Tin và Truyền Thông, một khối lượng dữ liệu khổng lồ được sản sinh ra hàng ngày. Việc đưa ra các công cụ khai thác hiệu quả trên khối lượng lớn dữ liệu ấy nhằm rút trích các tri thức phục vụ cho nhu cầu sử dụng của con người là vô cùng cấp thiết. Nó được ứng dụng hiệu quả ở một số ngành, lĩnh vực như:

- Ngành Marketing

Với việc đánh giá dữ liệu từ người dùng, các marketer có thể dễ dàng đưa ra các chiến dịch marketing cực kì hiệu quả, nhờ vào việc khai phá dữ liệu từ khách hàng với các đặc điểm như hành vi mua hàng trước đây, lịch sử mua hàng, thói quen người dùng, nhóm khách hàng....

- Chính Phủ

Các tổ chức chính phủ hoạt động về an ninh cộng đồng hoặc tiện ích xã hội sở hữu rất nhiều nguồn dữ liệu có thể khai thác insights. Ví dụ: khi phân tích dữ liệu cảm biến, chính phủ sẽ tăng mức độ hiệu quả của dịch vụ và tiết kiệm chi phí. Machine Learning còn hỗ trợ phát hiện gian lận và giảm thiểu khả năng trộm cắp danh tính.

- Các dịch vụ tài chính

Ngân hàng và các doanh nghiệp hoạt động trong lĩnh vực tài chính sử dụng việc khai thác dữ liệu với 2 mục đích chính: xác định insights trong dữ liệu và ngăn chặn lừa đảo. Insights sẽ cho biết được các cơ hội đầu tư hoặc thông báo đến nhà đầu tư thời điểm giao dịch hợp lý. Ngoài ra, dựa vào dữ liệu, các công ty, tổ chức cũng có thể tìm được những khách hàng đang có hồ sơ rủi ro cao hoặc sử dụng giám sát mạng để chỉ rõ những tín hiệu lừa đảo.

Như vậy, việc vận dụng khai phá dữ liệu lớn có thể giải quyết được rất nhiều các lĩnh vực của đời sống.

1.2. Mục đích đề tài

Việc nghiên cứu đề tài "Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn" trước hết là quá trình tìm hiểu, tổng hợp những kiến thức đã học để hoàn thành khóa luận tốt nghiệp đại học tại trường Đại học Thông Tin Liên Lạc. Tiếp theo là nghiên cứu khoa học, bổ sung kiến thức bản thân phục vụ cho việc học tập, làm việc của bản thân và rộng hơn nữa là cho các cá nhân tổ chức có nhu cầu nghiên cứu lĩnh vực này có thêm nguồn tài liệu nghiên cứu, định hình được một khái niệm về việc khai phá dữ liệu. Và mục đích cuối cùng là có thể tạo ra được sản phẩm nền tảng giải quyết các vấn đề thực tiễn đáp ứng nhu cầu của các cá nhân tổ chức. Mục đích cụ thể và trước mắt sẽ là tạo ra một báo cáo khoa học về quy trình xử lý và khai phá dữ liệu lớn.

1.3. Phạm vi đề tài

Đối với đề tài khóa luận tốt nghiệp "Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn" sẽ có các phạm vi nghiên cứu cũng như xây dựng đề tài trong giới hạn kiến thức và kinh phí của sinh viên cũng như là nội dung mà đề tài đề cập đến. Cụ thể như sau:

Thời gian nghiên cứu: Từ 3 tháng đến 4 tháng. Có kết hợp việc học và thực tập tại cơ quan.

Nội dung nghiên cứu:

- Kiến thức toán liên quan đến việc khai phá dữ liệu (đại số tuyến tính, giải tích, xác suất – thống kê, tối ưu hóa, Machine Learning, ngôn ngữ lập trình Python...).
- Quy trình khai phá dữ liệu.
- Kĩ thuật mô tả và xử lý dữ liệu.
- Kết hợp lý thuyết và thực tiễn tạo nên bài luận về đề tài "Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn".

1.4. Phương pháp nghiên cứu

Để thực hiện được đề tài này cần phải áp dụng các phương pháp nghiên cứu khoa học logic, thống nhất, hiệu quả để đạt năng xuất và chất lượng cao nhất cho đề tài này. Cụ thể, đầu tiên là tìm hiểu kiến thức từ các tài liệu, các kênh thông tin. Thứ hai là tổng hợp các kiến thức đã học có liên quan. Thứ ba là thí nghiệm xử lý với dữ liệu càng lớn càng tốt. Cụ thể hơn nữa:

Đối với việc thu thập thông tin:

- Thu thập thông tin từ các blog chuyên về Machine Learning như Machine Learning Cơ Bản, Ông Xuân Hồng.
- Nghiên cứu từ các khóa học online ở: Standford, Edx, Cousera, Udemy.
- Tham khảo từ một số đầu sách nổi tiếng về Machine Learning: Bishop, MachineLearningMastery.

Đối với việc xử lý thông tin:

- Nhận tất cả các thông tin liên quan nhưng chỉ xử lý và chắt lọc các thông tin liên quan trực tiếp hoặc gián tiếp gần với đề tài.
- Kiểm chứng lại thông tin trước khi sử dụng.
- Mô hình hóa các thông tin nhận được để chuyển thông tin thành dạng khoa học logic.

1.5. Bố cục luận văn

Bố cục luận văn được chia làm 4 phần tương ứng với 4 chương:

Chương 1: Tổng quan về phương pháp xử lý dữ liệu lớn

Tìm hiểu về tính thực tiễn của đề tài, tầm quan trọng của việc khai phá dữ liệu lớn hiện nay. Ngoài ra, nói lên mục đích và phạm vi đề tài. Cuối cùng là trình bày phương pháp nghiên cứu.

Chương 2: Cơ sở lý thuyết

Ở chương này, tìm hiểu về Machine Learning, quy trình khai phá dữ liệu, các phương pháp khai thác dữ liệu, mô hình chung cho các bài toán Machine

Learning và các tình huống hoặc trường hợp mắc phải khi làm việc với dữ liệu lớn, ví dụ như Overfitting, Multicollinearity.

Chương 3: Một số mô hình Machine Learning phổ biến

Nghiên cứu một số mô hình Machine Learning tiêu biểu và được áp dụng trong khóa luận tốt nghiệp. Cụ thể, một số mô hình như: Linear Regression, Gradient Decent, K-means, Logistic Regression, Support Vector Machine, Artificial Neural Network.

Chương 4: Đánh giá mô hình

Ở chương 4, sau khi sử dụng mô hình Machine Learning, ta sẽ sử dụng một số phương pháp như Accuracy, Confusion Matrix, True/False Positive/Negative, ROC, Area Under the Curve, Precision-Recall và RSS để đánh giá mô hình Machine Learning.

Chương 2
CỞ SỞ LÝ THUYẾT

2.1. Machine Learning

2.1.1. Định nghĩa

Machine Learning (máy học) 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.

2.1.2. Các nhóm giải thuật của Machine Learning

Các giải thuật học máy được phân ra làm 2 loại chính là:

- Học có giám sát (Supervised Learning)

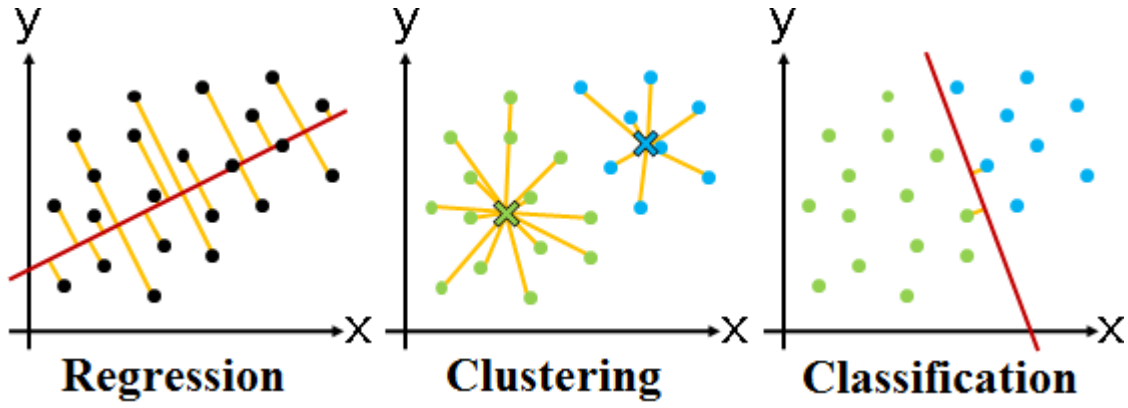
Là phương pháp sử dụng những dữ liệu đã được gán nhãn từ trước để suy luận ra quan hệ giữa đầu vào và đầu ra. Các dữ liệu này được gọi là dữ liệu huấn luyện và chúng là cặp các đầu vào - đầu ra. Học có giám sát sẽ xem xét các tập huấn luyện này để từ đó có thể đưa ra dự đoán đầu ra cho 1 đầu vào mới chưa gặp bao giờ. Ví dụ dự đoán giá nhà, phân loại email. Thuật toán Supervised Learning còn được tiếp tục chia nhỏ ra thành hai loại chính: Classification (Phân loại) và Regression (Hồi quy).

- Học phi giám sát (Unsupervised Learning)

Khác với học có giám sát, học phi giám sát sử dụng những dữ liệu chưa được gán nhãn từ trước để suy luận. Phương pháp này thường được sử dụng để tìm cấu trúc của tập dữ liệu. Tuy nhiên lại không có phương pháp đánh giá được cấu trúc tìm ra được là đúng hay sai. Ví dụ như phân cụm dữ liệu, triết xuất thành phần chính của một chất nào đó. Các bài toán Unsupervised learning được tiếp tục chia nhỏ thành hai loại: Clustering (phân nhóm) và Association.

Ngoài ra còn có 1 loại nữa là học tăng cường (Reinforcement Learning). Phương pháp học tăng cường tập trung vào việc làm sao để cho 1 tác tử trong môi trường có thế hành động sao cho lấy được phần thưởng nhiều nhất có thể. Khác

với học có giám sát nó không có cặp dữ liệu gán nhãn trước làm đầu vào và cũng không có đánh giá các hành động là đúng hay sai.



Hình 2.1. Phân nhóm Machine Learning dựa trên chức năng

2.1.3. Regression

Phân tích hồi quy là một phân tích thống kê để xác định xem các biến độc lập quy định các biến phụ thuộc như thế nào. Được dùng trong các bài toán dự đoán.

Ví dụ: dự đoán giá nhà dựa vào địa điểm, diện tích, số phòng, giá nhà các năm trước đây...

2.1.4. Classification

Một bài toán được gọi là classification nếu các label của dữ liệu đầu vào 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.

2.1.5. Clustering

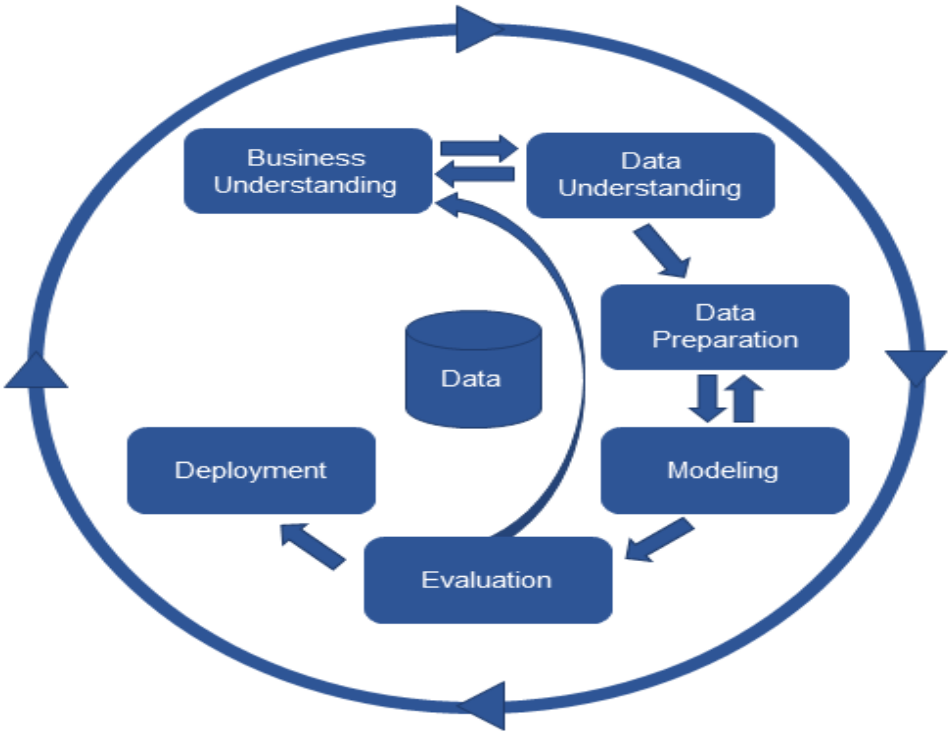
Một bài toán phân nhóm toàn bộ dữ liệu X thành các nhóm nhỏ dựa trên đặc điểm và đặc trưng giữa các dữ liệu trong mỗi nhóm.

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.

2.2. Quy trình khai phá dữ liệu

Quy trình tiêu chuẩn liên ngành cho khai thác dữ liệu, thường được biết đến với từ viết tắt CRISP-DM (Cross Industry Standard Process for Data Mining) là một quy trình lặp, có khả năng quay lui (backtracking) gồm 6 giai đoạn:



Hình 2.2. Quy trình CRISP-DM

2.2.1. Tìm hiểu nghiệp vụ (Business understanding)

- Hiểu mục tiêu nghiệp vụ.
- Đánh giá tình huống.
- Quy đổi từ mục tiêu nghiệp vụ sang mục tiêu khai phá dữ liệu.
- Xây dựng kế hoạch dự án.

2.2.2. Tìm hiểu dữ liệu (Data understanding)

- Xem xét các yêu cầu về dữ liệu.
- Thu thập, thăm dò và đánh giá chất lượng ban đầu.

2.2.3. Chuẩn bị dữ liệu (Data preparation)

- Lựa chọn dữ liệu.
- Thu thập dữ liệu.
- Tích hợp và định dạng dữ liệu (ETL).
- Làm sạch dữ liệu.
- Chuyển đổi dữ liệu.

2.2.4. Mô hình hoá (Modeling)

- Lựa chọn kỹ thuật mô hình thích hợp.
- Tách tập dữ liệu thành dữ liệu tập huấn (training data) và dữ liệu thử nghiệm (testing data) tập hợp con cho mục đích đánh giá.
- Phát triển và kiểm tra các thuật toán mô hình thay thế và cài đặt tham số.
- Tinh chỉnh cài đặt mô hình theo đánh giá ban đầu về hiệu suất của mô hình.

2.2.5. Đánh giá mô hình (Evaluation Model)

- Đánh giá mô hình trong bối cảnh ứng với các tiêu chí nghiệp vụ.
- Phê duyệt mẫu.

2.2.6. Triển khai (Deployment)

- Tạo báo cáo kết quả.
- Lập kế hoạch và phát triển quy trình triển khai.
- Triển khai mô hình.
- Phân phối kết quả mô hình và tích hợp trong hệ thống của tổ chức.
- Đánh giá dự án.

2.3. Các phương pháp khai thác dữ liệu

2.3.1. Thống kê mô tả (Descriptive Statistics)

Là các phương pháp liên quan đến việc thu thập số liệu, tóm tắt, trình bày, tính toán và mô tả các đặc trưng khác nhau để phản ánh một cách tổng quát đối tượng nghiên cứu.

- Các nguyên tắc định lượng mô tả các features chính của một tập dữ liệu. Về bản chất, nó mô tả một bộ dữ liệu.
- Quy trình mô tả và diễn dịch là các bước khác nhau.

2.3.2. Thống kê suy luận (Inferential Statistics)

Là bao gồm các phương pháp ước lượng các đặc trưng của tổng thể, phân tích mối liên hệ giữa các hiện tượng nghiên cứu, dự đoán hoặc ra quyết định trên cơ sở thu thập thông tin từ kết quả quan sát mẫu. Nghĩa là, thống kê từ một lượng sample (mẫu) để dự đoán về population (quần thể).

- Suy luận thường là mục tiêu của các mô hình thống kê (Statistical models).
- Suy luận phụ thuộc rất nhiều vào population và sample.

2.3.3. Phân tích dữ liệu thăm dò (Exploratory Data Analysis - EDA)

Là một cách tiếp cận phân tích các tập dữ liệu để tóm tắt các đặc điểm chính của chúng, tìm các mối quan hệ chưa được biết trước đó, thường với các phương pháp trực quan.

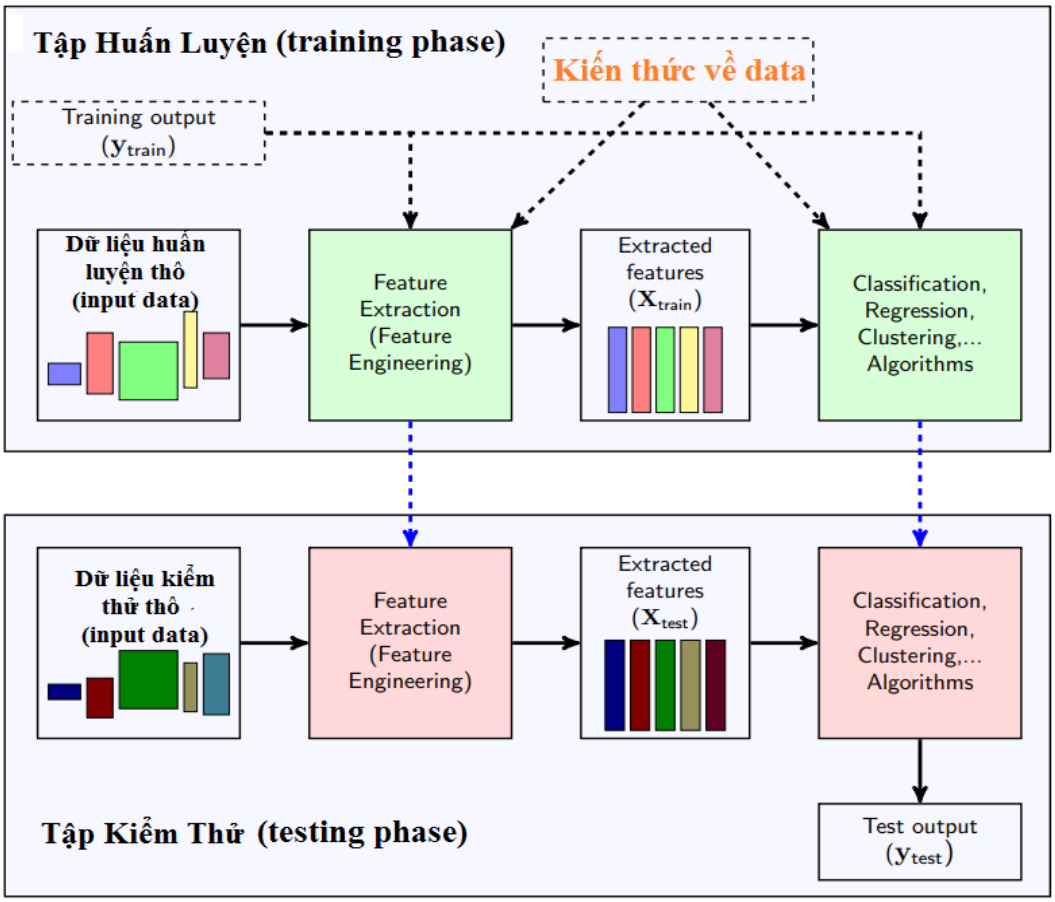
- Khám phá các model tốt cho việc khám phá các kết nối mới.
- Có ích cho việc định nghĩa các nghiên cứu và câu hỏi trong tương lai.
- Không nên sử dụng phân tích thăm dò để khái quát hoặc dự đoán.

2.3.4. Phân tích dự đoán (Predictive Analysis)

Phân tích dự đoán là nhánh của phân tích nâng cao được sử dụng để đưa ra dự đoán về các sự kiện không xác định trong tương lai. Phân tích dự đoán sử dụng nhiều kỹ thuật từ khai phá dữ liệu, thống kê, mô hình hóa, học máy và trí thông minh nhân tạo để phân tích dữ liệu hiện tại để đưa ra dự đoán về tương lai. Các mẫu tìm thấy trong dữ liệu lịch sử và giao dịch có thể được sử dụng để xác định các rủi ro và cơ hội cho tương lai. Các mô hình phân tích dự đoán nắm bắt các mối quan hệ giữa nhiều yếu tố để đánh giá rủi ro với một tập hợp các điều kiện cụ thể để gán điểm hoặc trọng số. Bằng cách áp dụng thành công các phân tích dự báo, các doanh nghiệp có thể giải thích một cách hiệu quả các dữ liệu lớn vì lợi ích của họ.

2.4. Mô hình chung cho các bài toán Machine Learning

Phần lớn các bài toán Machine Learning thường được thể hiện trong hình vẽ dưới đây:



Hình 2.3. Mô hình chung cho các bài toán Machine Learning

2.4.1. Training phase

2.4.1.1. Dữ liệu đầu vào (input data)

Mục đích của Feature Engineering là tạo ra một "Feature Extractor" biến dữ liệu thô ban đầu thành dữ liệu phù hợp với từng mục đích khác nhau.

2.4.1.2. Đầu ra (output)

Output của training set

- Trong các bài toán học không giám sát, ta không biết output nên hiển nhiên sẽ không có đầu vào này.
- Trong các bài toán học giám sát, có khi dữ liệu này cũng không được sử dụng.

Ví dụ: nếu dữ liệu đầu vào thô đã có cùng số chiều rồi nhưng số chiều quá lớn, ta muốn giảm số chiều của nó thì cách đơn giản nhất là chiếu vector đó xuống một không gian có số chiều nhỏ hơn bằng cách lấy một ma trận ngẫu nhiên nhân với nó. Ma trận này thường là ma trận béo (số hàng ít hơn số cột, tiếng Anh - fat matrices) để đảm bảo số chiều thu được nhỏ hơn số chiều ban đầu. Việc làm này mặc dù làm mất đi thông tin, tuy nhiên 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 đầu vào thô, ta sẽ có bài toán tìm ma trận chiếu để lượng thông tin mất đi là ít nhất.

Trong nhiều trường hợp, dữ liệu output của training set cũng được sử dụng để tạo ra Feature Extractor. Ví dụ: trong bài toán classification, ta không quan tâm nhiều đến việc mất thông tin hay không, ta chỉ quan tâm đến việc những thông tin còn lại có đặc trưng cho từng class hay không. Ví dụ: dữ liệu thô là các hình vuông và hình tam giác có màu đỏ và xanh. Trong bài toán phân loại đa giác, các output là tam giác và vuông, thì ta không quan tâm tới màu sắc mà chỉ quan tâm tới số cạnh của đa giác. Ngược lại, trong bài toán phân loại màu, các class là xanh và đỏ, ta không quan tâm tới số cạnh mà chỉ quan tâm đến màu sắc thôi.

- Main Algorithms (các thuật toán chính)

Khi có được extracted features rồi, ta sử dụng những thông tin này cùng với training output và prior knowledge để tạo ra các mô hình phù hợp.

Lưu ý: Khi xây dựng bộ feature extractor và main algorithms, ta không được sử dụng bất kỳ thông tin nào trong tập test data. Ta phải giả sử rằng những thông tin trong test data chưa được nhìn thấy bao giờ.

2.4.2. Testing phase

Bước này đơn giản hơn nhiều. Với dữ liệu đầu vào thô mới, ta sử dụng feature extractor đã tạo được ở trên (tất nhiên không được sử dụng output của nó vì output là cái ta đang đi tìm) để tạo ra feature vector tương ứng. Feature vector được đưa vào main algorithm đã được học ở training phase để dự đoán output.

2.5. Feature Engineering

Khi làm việc với các bài toán Machine Learning thực tế, nhìn chung chỉ có được dữ liệu thô (raw) chưa qua chỉnh sửa, chọn lọc. Ta cần phải tìm một phép biến đổi để loại ra những dữ liệu nhiễu (noise), và để đưa dữ liệu thô với số chiều khác nhau về cùng một chuẩn (cùng là các vector hoặc ma trận). Dữ liệu chuẩn mới này phải đảm bảo giữ được những thông tin đặc trưng (features) cho dữ liệu thô ban đầu. Không những thế, tùy vào từng bài toán, ta cần thiết kế những phép biến đổi để có những features phù hợp. Quá trình quan trọng này được gọi là Feature Extraction, hoặc Feature Engineering, tiếng việt gọi là trích chọn đặc trưng. Feature engineering cố gắng biểu diễn tốt nhất tập dữ liệu ban đầu sao cho tương thích với mô hình dự đoán đang sử dụng.

2.5.1. Feature Selection

Giả sử rằng các điểm dữ liệu có số features khác nhau (do kích thước dữ liệu khác nhau hay do một số feature mà điểm dữ liệu này có nhưng điểm dữ liệu kia lại không thu thập được) và số lượng features là cực lớn. Ta cần chọn ra một số lượng nhỏ hơn các feature phù hợp với bài toán. Một số cách trong Feature Selection như:

2.5.1.1. Cách 1: Loại bỏ các features có giá trị phương sai thấp

VarianceThreshold (ngưỡng phương sai) là một phương pháp cơ bản của Feature Selection. Nó loại bỏ tất cả các features có phương sai không đáp ứng giá trị ngưỡng. Theo mặc định, nó loại bỏ tất cả các tính năng zero-variance (phương sai bằng 0), tức là các features có cùng giá trị trong tất cả các sample.

$$\boldsymbol{Var(X) = p(1-p)}$$

Trong đó: **p** là số phần trăm muốn loại bỏ của một sample, như ví dụ dưới đây, ta chọn 80% = 0.8;

```
from sklearn.feature_selection import VarianceThreshold
import numpy as np
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1,
1]]
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
print("Ma trận ban đầu:\n{}\n\nMa trận sau khi
transform:\n{}\n".format(X, sel.fit_transform(X)))
def column(matrix, i):
    return [row[i] for row in matrix]
print("Giá trị của varianceThreshold là: {}".format(sel))
print("Giá trị variance của cột 0 là: {}\nGiá trị variance của cột 1
là: {}".format(np.var(column(X,0)),np.var(column(X,1))))
```

Kết quả:

```
Ma trận ban đầu:
[[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1,
1]]
Ma trận sau khi transform:
[[0 1] [1 0] [0 0] [1 1] [1 0] [1 1]]
#Ta có thể thấy VarianceThreshold đã xóa cột đầu tiên của ma
trận vì variance của cột 0 < giá trị VarianceThreshold
#để kiểm chứng, ta sẽ thử xem giá trị của cột 0 và cột 1 của ma
trận
Giá trị của varianceThreshold là:
VarianceThreshold(threshold=0.15999999999999998)
Giá trị variance của cột 0 là: 0.13888888888888892
Giá trị variance của cột 1 là: 0.22222222222222224
```

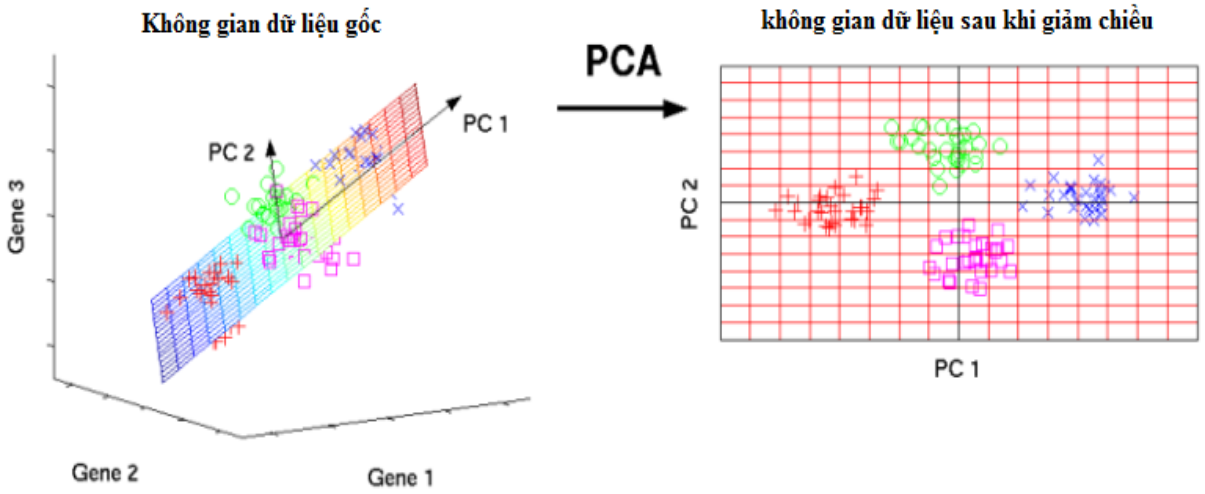
2.5.1.2. Dimentionality Reduction

Một phương pháp khác là làm giảm số chiều của dữ liệu để giảm bộ nhớ và khối lượng tính toán. Việc giảm số chiều này có thể được thực hiện bằng nhiều cách, trong đó random projection là cách đơn giản nhất. Tức chọn một ma trận chiếu (projection matrix) ngẫu nhiên rồi nhân nó với từng điểm dữ liệu (giả sử dữ liệu ở dạng vector cột) để được các vector có số chiều thấp hơn.

Ví dụ: vector ban đầu có số chiều là 784, chọn ma trận chiếu có kích thước (100x784), khi đó nếu nhân ma trận chéo này với vector ban đầu, ta sẽ được một vector mới có số chiều là 100, nhỏ hơn số chiều ban đầu rất nhiều. Lúc này, có thể ta không có tên gọi cho mỗi feature nữa vì các feature ở vector ban đầu đã được trộn lẫn với nhau theo một tỉ lệ nào đó rồi lưu và vector mới này. Mỗi thành phần của vector mới này được coi là một feature (không tên).

Việc chọn một ma trận chiếu ngẫu nhiên đôi khi mang lại kết quả tệ không mong muốn vì thông tin bị mất đi quá nhiều.

Một phương pháp được sử dụng nhiều để hạn chế lượng thông tin mất đi có tên là Principle Component Analysis:



Hình 2.4. Sử dụng phương pháp PCA để giảm chiều dữ liệu 3D xuống 2D

2.5.2. Bag-of-Word & TF-IDF

Bag-of-word (BoW) là phương pháp đưa các từ, các câu, đoạn văn ở dạng text trong các văn bản về một vector mà mỗi phần tử là một số. Ý tưởng của BoW là phân tích và phân nhóm dựa theo "Túi đựng từ". Với dữ liệu kiểm thử mới, tiến hành tìm ra số lần từng từ của dữ liệu kiểm thử xuất hiện trong "túi". Tuy nhiên BoW vẫn tồn tại khuyết điểm, nên TF-IDF là phương pháp khắc phục. Có thể ứng dụng BoW + TF-IDF vào việc tìm kiếm, phân loại tài liệu, lọc mail spam xác định ý định của người dùng.

Giải pháp phổ biến là sử dụng một phương pháp thống kê có tên là Term Frequency– Inverse Document Frequency (TF-IDF), giá trị TF-IDF của một từ là một con số thu được qua thống kê thể hiện mức độ quan trọng của từ này trong một văn bản, mà bản thân văn bản đang xét nằm trong một tập hợp các văn bản.

Đầu tiên, TF là tần số xuất hiện của 1 từ trong 1 văn bản tính như sau:

$$f_{t,d} / \sum_{t' \in d}^{n} f_{t',d}$$

Trong đó:

- $f_{t,d}$ số lần xuất hiện từ t trong văn bản d;
- Mẫu số là tổng số từ trong văn bản d;

Tiếp theo, IDF là tần số nghịch của 1 từ trong tập văn bản.

Mục đích của việc tính IDF là giảm giá trị của các từ thường xuyên xuất hiện như "is", "the" Do các từ này không mang nhiều ý nghĩa trong việc phân loại văn bản.

cái điểm idf này càng nhỏ khi mà cái từ " NLP" này xuất hiện nhiều trong N-documents, còn trong 1 văn bản nó có xuất hiện 1000 lần nhưng những văn

$$idf(t, D) = log\ \frac{N}{|\{d \in D: t \in D\}|}$$

Trong đó:

- **N:** là tổng các văn bản trong một tập văn bản **N = |D|**;
- $|\{d \in D: t \in D\}|$ số lượng văn bản khi mà t xuất hiện;

Cuối cùng, TF-IDF những từ có giá trị TF-IDF cao là những từ xuất hiện nhiều trong văn bản này, và xuất hiện ít trong các văn bản khác. Việc này giúp lọc ra những từ phổ biến và giữ lại những từ có giá trị cao (từ khoá của văn bản đó). Được tính bởi công thức:

$$tfidf(t, d, D) = tf(t, d).idf(t, D)$$

Áp dụng:

Khởi tạo 2 văn bản, tính số lần xuất hiện của mỗi từ trong văn bản:

```
# Khởi tạo 2 văn bản text1, text2
text1 = "Phúc thích xem phim , Trâm cũng thích xem phim"
text2 = "Ngoài ra , Phúc còn thích bơi lội"
bowA = text1.split(" ") # tách từ ở văn bản 1
bowB = text2.split(" ") # tách từ ở văn bản 2

#Tạo một dictionary
word_dict = set(bowA).union(set(bowB))
wordDictA = dict.fromkeys(word_dict, 0)
wordDictB = dict.fromkeys(word_dict, 0)

#Đếm số lượng từ
for word in bowA:
    wordDictA[word]+=1
for word in bowB:
    wordDictB[word]+=1

print("Các từ trong 2 văn bản là:\n {}".format(word_dict))
print("Số từ xuất hiện trong văn bản 1 là:\n {}\n\nSố từ xuất hiện
trong văn bản 2 là:\n {}".format(wordDictA,wordDictB))
```

Kết quả:

```
Các từ trong 2 văn bản là:
 {'thích', 'phim', ',', 'lội', 'cũng', 'ra', 'Trâm', 'còn', 'Ngoài',
'Phúc', 'xem', 'bơi'}
Số từ xuất hiện trong văn bản 1 là:
 {'thích': 2, 'phim': 2, ',': 1, 'lội': 0, 'cũng': 1, 'ra': 0,
'Trâm': 1, 'còn': 0, 'Ngoài': 0, 'Phúc': 1, 'xem': 2, 'bơi': 0}
Số từ xuất hiện trong văn bản 2 là:
 {'thích': 1, 'phim': 0, ',': 1, 'lội': 1, 'cũng': 0, 'ra': 1,
'Trâm': 0, 'còn': 1, 'Ngoài': 1, 'Phúc': 1, 'xem': 0, 'bơi': 1}
```

Tính TF:

```
def compute_TF(word_dict, bow):
    tf_dict = {}
    bow_count = len(bow)
    for word, count in word_dict.items():
        tf_dict[word] = count / float(bow_count)
    return tf_dict
print("\nKết quả TF:\n văn bản 1: {}\n văn bản 2 {}"
.format(compute_TF(wordDictA,bowA),compute_TF(wordDictB,bowB)))
```

Kết quả:

```
Kết quả TF:
 văn bản 1: {'xem': 0.2, 'thích': 0.2, 'còn': 0.0, 'Ngoài': 0.0,
'ra': 0.0, 'Trâm': 0.1, ',': 0.1, 'bơi': 0.0, 'lội': 0.0, 'Phúc':
0.1, 'phim': 0.2, 'cũng': 0.1}
 văn bản 2: {'xem': 0.0, 'thích': 0.125, 'còn': 0.125, 'Ngoài':
0.125, 'ra': 0.125, 'Trâm': 0.0, ',': 0.125, 'bơi': 0.125, 'lội':
0.125, 'Phúc': 0.125, 'phim': 0.0, 'cũng': 0.0}
```

Tính IDF:

```
def compute_IDF(doc_list):
    import math #import thư viện math
    idf_dict = {} #tạo một dictionary rỗng
    N = len(doc_list) #gán độ dài của list cho biến N
    idf_dict = dict.fromkeys(doc_list[0].keys(), 0) #tạo dictionary lưu
các keys với value = 0
    #lọc ra thành 1 list gồm các từ xuất hiện >=1 lần
    for doc in doc_list:
        for word, count in doc.items():
            if count > 0:
                idf_dict[word] += 1
    for word, count in idf_dict.items():
        idf_dict[word] = math.log(N / float(count))
    return idf_dict
print("Kết Quả IDF:\n {}".format(compute_IDF([wordDictA, wordDictB])))
```

```
Kết Quả IDF:
{'cũng': 0.6931471805599453, 'Phúc': 0.0, 'bơi': 0.6931471805599453,
'Trâm': 0.6931471805599453, 'ra': 0.6931471805599453, 'phim':
0.6931471805599453, 'lội': 0.6931471805599453, 'còn':
0.6931471805599453, 'xem': 0.6931471805599453, 'Ngoài':
0.6931471805599453, ',': 0.0, 'thích': 0.0}
```

Từ kết quả trên có thể nhìn thấy những từ có trọng số càng cao thì những từ đó càng có giá trị phân loại và ngược lại ví dụ như từ "thích", dấu “,” xuất hiện nhiều nên sẽ không có giá trị phân loại các văn bản với nhau. Tuy nhiên bộ train data lần này ít nên hiệu quả không rõ rệt, nếu thử trên lượng data lớn chắc chắn sẽ rất hiệu quả.

2.5.3. Feature Scaling & Normalization

Các điểm dữ liệu đôi khi được đo đạc với những đơn vị khác nhau, m và feet chẳng hạn. Hoặc có hai thành phần (của vector dữ liệu) chênh lệch nhau quá lớn, một thành phần có khoảng giá trị từ **0** đến **1000**, thành phần kia chỉ có khoảng giá trị từ **0** đến **1** chẳng hạn. Lúc này, ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

Rescaling

Phương pháp đơn giản nhất là đưa tất cả các thành phần về cùng một khoảng, **[0,1]** hoặc **[-1,1]**. Nếu muốn đưa một thành phần (feature) về khoảng **[0,1]** công thức sẽ là:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

Trong đó:

- x là giá trị ban đầu;
- x' là giá trị sau khi chuẩn hóa;
- $min(x), max(x)$ được tính trên toàn bộ dữ liệu training data ở cùng một thành phần. Việc này được thực hiện trên từng thành phần của vector dữ liệu **x**;

Ví dụ xử dụng phương pháp Rescaling để chuẩn hóa:

```
import numpy as np
#thư viện sklearn để lấy hàm min_max_scaler
from sklearn import preprocessing
#Tạo một ma trận tên X_Train
X_train = np.array([[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]])
#gán biến min_max_scaler từ hàm MinMaxScaler() cho khoảng [0,1]
#gán biến max_abs_scaler từ hàm MaxAbsScaler() cho khoảng [-1,1]
min_max_scaler = preprocessing.MinMaxScaler()
max_abs_scaler = preprocessing.MinMaxScaler()
#Thực hiện chuyển đổi về khoảng [0,1] và [-1,1] bằng hàm
fit_transform
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
#in ra màn hình ma trận sau khi scaled!
print("Ma trận ban đầu:\n{}\n\nMa trận sau khi scaling:\nĐối với
khoảng [0,1]:\n{}\n\n""Đối với khoảng [-1,1]:\n{}"
.format(X_train,X_train_minmax,X_train_maxabs))
```

Kết quả:

```
Ma trận ban đầu:
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]
Ma trận sau khi scale:
Đối với khoảng [0,1]:
[[ 0.5         0.          1.        ]
 [ 1.          0.5         0.33333333]
 [ 0.          1.          0.        ]]
Đối với khoảng [-1,1]:
[[ 0.5 -1.   1. ]
 [ 1.   0.   0. ]
 [ 0.   1.  -0.5]]
```

Standardization

Một phương pháp nữa cũng hay được sử dụng là giả sử mỗi thành phần đều có phân phối chuẩn với kỳ vọng là 0 và phương sai là 1. Khi đó, công thức chuẩn hóa sẽ là:

$$x' = \frac{x - \bar{a}}{\sigma}$$

Trong đó: $\bar{a}, \sigma$ lần lượt là kỳ vọng và phương sai (standard deviation) của thành phần đó trên toàn bộ training data;

Ví dụ xử dụng phương pháp Standardization để chuẩn hóa:

```
#thêm thư viện xử lý ma trận
import numpy as np
#thư viện sklearn để lấy hàm min_max_scaler
from sklearn import preprocessing
#Tạo một ma trận tên X_Train
X_train = np.array([[1.,-1.,2.],[2.,0.,0.],[0.,1.,-1.]])
#gán biến scaler từ hàm StandardScaler() sau khi fit ma trận
scaler = preprocessing.StandardScaler().fit(X_train)
print("Ma trận ban đầu:\n{}\n\nMa trận sau khi scaling:\n{}"
.format(X_train,scaler.transform(X_train)))
```

Kết quả:

```
Ma trận ban đầu:
[[ 1. -1.  2.]
 [ 2.  0.  0.]
 [ 0.  1. -1.]]
Ma trận sau khi scaling:
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

Scaling to unit length

Một lựa chọn khác nữa cũng được sử dụng rộng rãi là chuẩn hóa các thành phần của mỗi vector dữ liệu sao cho toàn bộ vector có độ lớn (Euclid, tức norm 2) bằng 1. Việc này có thể được thực hiện bằng:

$$x' = \frac{x}{||x||_2}$$

Trong đó:

- $||x||_2$ là norm 2;
- x là dữ liệu muốn chuẩn hóa;
- x' là dữ liệu sau khi đã được chuẩn hóa khi dùng norm 2;

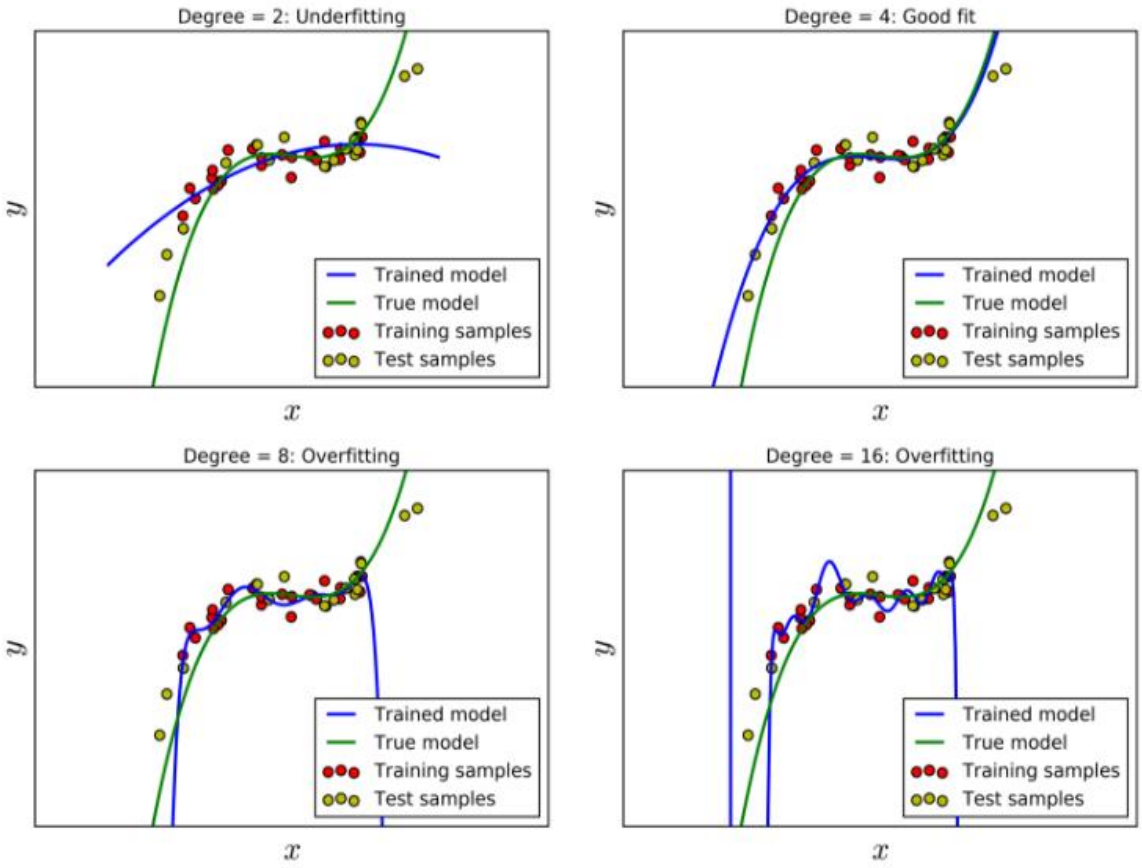
2.6. Overfitting

2.6.1. Giới thiệu

Overfitting là hiện tượng mô hình tìm được quá khớp với dữ liệu training. Việc quá khớp này có thể dẫn đến việc dự đoán nhầm nhiễu, và chất lượng mô hình không còn tốt trên dữ liệu test nữa. Dữ liệu test được giả sử là không được biết trước, và không được sử dụng để xây dựng các mô hình Machine Learning.

2.6.2. Ví dụ

Có 50 điểm dữ liệu được tạo bằng một đa thức bậc ba cộng thêm nhiễu. Tập dữ liệu này được chia làm hai, 30 điểm dữ liệu màu đỏ cho training data, 20 điểm dữ liệu màu vàng cho test data. Đồ thị của đa thức bậc ba này được cho bởi đường màu xanh lá cây. Bài toán của ta là giả sử ta không biết mô hình ban đầu mà chỉ biết các điểm dữ liệu, hãy tìm một mô hình "tốt" để mô tả dữ liệu đã cho.



Hình 2.5. Underfitting và Overfitting với Polynominal Regression

Bài toán này hoàn toàn có thể được giải quyết bằng Linear Regression với dữ liệu mở rộng cho một cặp điểm $(\boldsymbol{x}, \boldsymbol{y})$ là $(\boldsymbol{x}, \boldsymbol{y})$ với $\boldsymbol{x} = [\mathbf{1}, \boldsymbol{x}, \boldsymbol{x^2}, \boldsymbol{x^3}, \dots, \boldsymbol{x^d}]$ cho đa thức bậc $\boldsymbol{d}$. Điều quan trọng là ta cần tìm bậc $\boldsymbol{d}$ của đa thức cần tìm.

Trong ví dụ trên, độ phức tạp của mô hình có thể được coi là bậc của đa thức cần tìm. Rõ ràng là một đa thức bậc không vượt quá 29 có thể fit được hoàn toàn với 30 điểm trong training data. Cùng xét vài giá trị $\boldsymbol{d} = \mathbf{2}, \mathbf{4}, \mathbf{8}, \mathbf{16}$. Với $\boldsymbol{d} = \mathbf{2}$, mô hình không thực sự tốt vì mô hình dự đoán quá khác so với mô hình thực. Trong trường hợp này, ta nói mô hình bị underfitting. Với $\boldsymbol{d} = \mathbf{8}$, với các điểm dữ liệu trong khoảng của training data, mô hình dự đoán và mô hình thực là khá giống nhau. Tuy nhiên, về phía phải, đa thức bậc 8 cho kết quả hoàn toàn ngược với xu hướng của dữ liệu. Điều tương tự xảy ra trong trường hợp $\boldsymbol{d} = \mathbf{16}$. Đa thức bậc 16 này quá fit dữ liệu trong khoảng đang xét, và quá fit, tức không được mượt trong khoảng dữ liệu training. Việc quá fit trong trường hợp bậc 16 không tốt vì mô hình đang cố gắng mô tả nhiễu hơn là dữ liệu. Hai trường hợp đa thức bậc cao này được gọi là Overfitting.

Về cơ bản, overfitting xảy ra khi mô hình quá phức tạp để mô phỏng training data. Điều này đặc biệt xảy ra khi lượng dữ liệu training quá nhỏ trong khi độ phức tạp của mô hình quá cao.

Một mô hình được coi là tốt (fit) nếu cả train error và test error đều thấp. Nếu train error thấp nhưng test error cao, ta nói mô hình bị overfitting. Nếu train error cao và test error cao, ta nói mô hình bị underfitting. Nếu train error cao nhưng test error thấp hiện tượng này phải rất may mắn mới xảy ra, hoặc có chỉ khi tập dữ liệu test quá nhỏ.

2.6.3. Một số phương pháp tránh overfitting

Validation/Hold-out

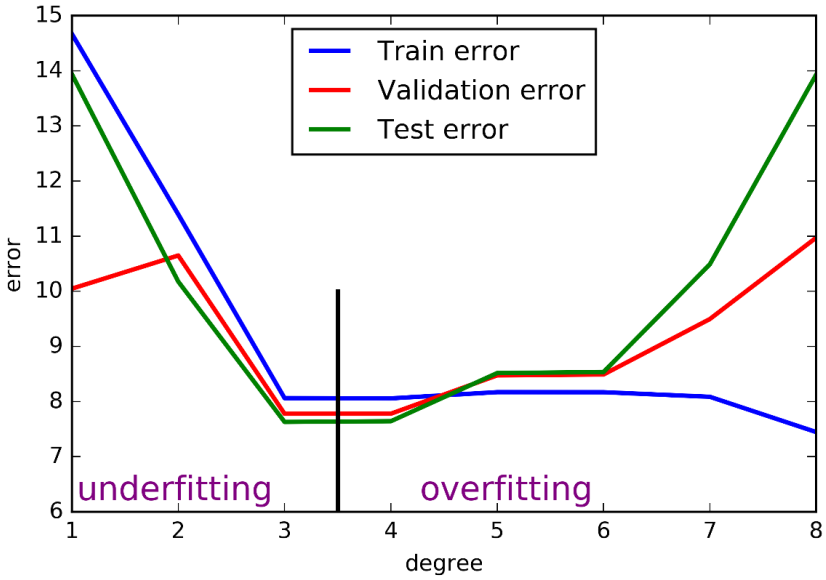
Phương pháp đơn giản nhất là trích từ tập training data ra một tập con nhỏ và thực hiện việc đánh giá mô hình trên tập con nhỏ này. Tập con nhỏ được trích ra từ training set này được gọi là validation set. Lúc này, training set là phần còn

lại của training set ban đầu. Train error được tính trên training set mới này, và có một khái niệm nữa được định nghĩa tương tự như trên validation error, tức error được tính trên tập validation.

Với khái niệm này, ta tìm mô hình sao cho cả train eror và validation error đều nhỏ, qua đó có thể dự đoán được rằng test error cũng nhỏ. Phương pháp thường được sử dụng là sử dụng nhiều mô hình khác nhau. Mô hình nào cho validation error nhỏ nhất sẽ là mô hình tốt.

Thông thường, ta bắt đầu từ mô hình đơn giản, sau đó tăng dần độ phức tạp của mô hình. Tới khi nào validation error có chiều hướng tăng lên thì chọn mô hình ngay trước đó. Chú ý rằng mô hình càng phức tạp, train error có xu hướng càng nhỏ đi.

Hình dưới đây mô tả ví dụ phía trên với bậc của đa thức tăng từ 1 đến 8. Tập validation bao gồm 10 điểm được lấy ra từ tập training ban đầu.



Hình 2.6. Bậc của đa thức

Ta hãy tạm chỉ xét hai đường màu lam và đỏ, tương ứng với train error và validation error. Khi bậc của đa thức tăng lên, train error có xu hướng giảm. Điều này dễ hiểu vì đa thức bậc càng cao, dữ liệu càng được fit. Quan sát đường màu đỏ, khi bậc của đa thức là 3 hoặc 4 thì validation error thấp, sau đó tăng dần lên. Dựa vào validation error, ta có thể xác định được bậc cần chọn là 3 hoặc 4. Quan

sát tiếp đường màu lục, tương ứng với test error, thật là trùng hợp, với bậc bằng 3 hoặc 4, test error cũng đạt giá trị nhỏ nhất, sau đó tăng dần lên. Vậy cách làm này ở đây đã tỏ ra hiệu quả.

Việc không sử dụng test data khi lựa chọn mô hình ở trên nhưng vẫn có được kết quả khả quan vì ta giả sử rằng validation data và test data có chung một đặc điểm nào đó. Và khi cả hai đều là unseen data, error trên hai tập này sẽ tương đối giống nhau.

Cross-validation

Trong nhiều trường hợp, ta có rất hạn chế số lượng dữ liệu để xây dựng mô hình. Nếu lấy quá nhiều dữ liệu trong tập training ra làm dữ liệu validation, phần dữ liệu còn lại của tập training là không đủ để xây dựng mô hình. Lúc này, tập validation phải thật nhỏ để giữ được lượng dữ liệu cho training đủ lớn. Tuy nhiên, một vấn đề khác nảy sinh. Khi tập validation quá nhỏ, hiện tượng overfitting lại có thể xảy ra với tập training còn lại. Lúc này ta sẽ sử dụng cross-validation (kiểm định chéo).

Cross validation là một cải tiến của validation với lượng dữ liệu trong tập validation là nhỏ nhưng chất lượng mô hình được đánh giá trên nhiều tập validation khác nhau. Một cách thường đường sử dụng là chia tập training ra $\boldsymbol{k}$ tập con không có phần tử chung, có kích thước gần bằng nhau. Tại mỗi lần kiểm thử, được gọi là run, một trong số $\boldsymbol{k}$ tập con được lấy ra làm validata set. Mô hình sẽ được xây dựng dựa vào hợp của $\boldsymbol{k-1}$ tập con còn lại. Mô hình cuối được xác định dựa trên trung bình của các train error và validation error. Cách làm này còn có tên gọi là k-fold cross validation.

Regularization

Một nhược điểm lớn của cross-validation là số lượng training runs tỉ lệ thuận với $\boldsymbol{k}$. Điều đáng nói là mô hình polynomial (đa thức) như trên chỉ có một tham số cần xác định là bậc của đa thức.

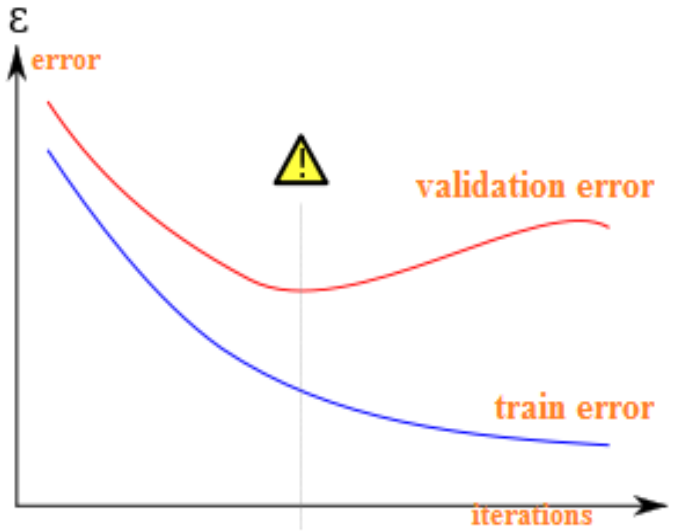
Trong các bài toán Machine Learning, lượng tham số cần xác định thường lớn hơn nhiều, và khoảng giá trị của mỗi tham số cũng rộng hơn nhiều, chưa kể

đến việc có những tham số có thể là số thực. Như vậy, việc chỉ xây dựng một mô hình thôi cũng là đã rất phức tạp rồi. Có một cách giúp số mô hình cần huấn luyện giảm đi nhiều, thậm chí chỉ một mô hình. Cách này có tên gọi chung là regularization.

Regularization, một cách cơ bản, là thay đổi mô hình một chút để tránh overfitting trong khi vẫn giữ được tính tổng quát của nó (tính tổng quát là tính mô tả được nhiều dữ liệu, trong cả tập training và test). Một cách cụ thể hơn, ta sẽ tìm cách di chuyển nghiệm của bài toán tối ưu hàm mất mát tới một điểm gần nó. Hướng di chuyển sẽ là hướng làm cho mô hình ít phức tạp hơn mặc dù giá trị của hàm mất mát có tăng lên một chút. Một kĩ thuật đơn giản là early stopping.

- **Early Stopping**

Trong nhiều bài toán Machine Learning, cần sử dụng các thuật toán lặp để tìm ra nghiệm, ví dụ như Gradient Descent. Nhìn chung, hàm mất mát giảm dần khi số vòng lặp tăng lên. Early stopping tức dừng thuật toán trước khi hàm mất mát đạt giá trị quá nhỏ, giúp tránh overfitting. Một kỹ thuật thường được sử dụng là tách từ training set ra một tập validation set như trên. Sau một (hoặc một số, ví dụ 50) vòng lặp, ta tính cả train error và validation error, đến khi validation error có chiều hướng tăng lên thì dừng lại, và quay lại sử dụng mô hình tương ứng với điểm và validation error đạt giá trị nhỏ.



Hình 2.7. Mô hình được xác định tại vòng lặp mà validation error đạt giá trị nhỏ nhất

- **Thêm số hạng vào hàm mất mát**

Kỹ thuật regularization phổ biến nhất là thêm vào hàm mất mát một số hạng nữa. Số hạng này thường dùng để đánh giá độ phức tạp của mô hình. Số hạng này càng lớn, thì mô hình càng phức tạp. Hàm mất mát mới này thường được gọi là regularized loss function, thường được định nghĩa như sau:

$$\boldsymbol{j_{reg}(\theta) = J(\theta) + \lambda R(\theta)}$$

Trong đó:

- $\boldsymbol{\theta}$ được dùng để ký hiệu các biến trong mô hình;
- $\boldsymbol{J(\theta)}$ ký hiệu cho hàm mất mát (loss function);
- $\boldsymbol{R(\theta)}$ là số hạng regularization;
- $\boldsymbol{\lambda}$ thường là một số dương để cân bằng giữa hai đại lượng ở vế phải;

2.7. Multicollinearity (Đa cộng tuyến)

2.7.1. Giới thiệu

Trong mô hình hồi quy (Linear Regression), nếu các biến độc lập có quan hệ chặt với nhau, các biến độc lập có mối quan hệ tuyến tính, nghĩa là các biến độc lập có tương quan chặt, mạnh với nhau thì sẽ có hiện tượng đa cộng tuyến, đó là hiện tượng các biến độc lập trong mô hình phụ thuộc lẫn nhau và thể hiện được dưới dạng hàm số. Ví dụ có hai biến độc lập A và B, khi A tăng thì B tăng, A giảm thì B giảm…. thì đó là một dấu hiệu của đa cộng tuyến. Nói một cách khác là hai biến độc lập có quan hệ rất mạnh với nhau, đúng ra hai biến này nó phải là 1 biến nhưng thực tế trong mô hình nhà nghiên cứu lại tách làm 2 biến. Hiện tượng đa cộng tuyến vi phạm giả định của mô hình hồi qui tuyến tính cổ điển là các biến độc lập không có mối quan hệ tuyến tính với nhau.

Đa cộng tuyến hoàn hảo:

X2	X3	X4
10	50	52
15	75	75
18	90	97
24	120	129

Bảng 2.1. Số giờ học và kết quả

Nhận Xét: X2 và **X3** có mối quan hệ tuyến tính chính xác: **X3** = 5**X2**

2.7.2. Cách phát hiện trường hợp đa cộng tuyến

Có 2 cách:

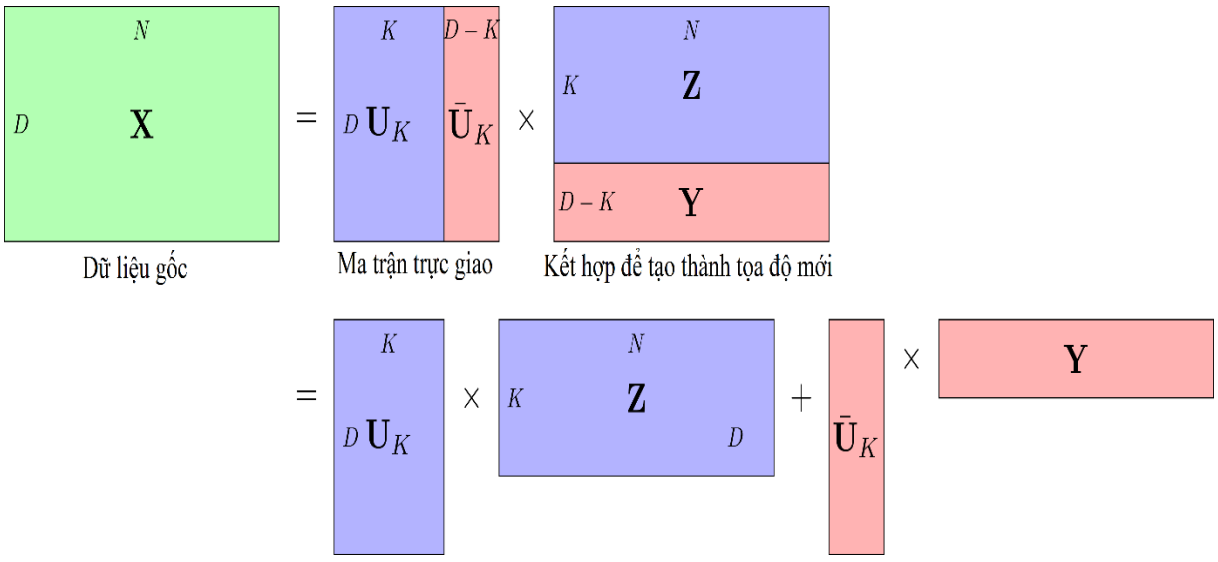
- Dựa vào hệ số phóng đại phương sai VIF (Variance Inflation Factor).
- Dựa vào hệ số tương quan, có hay không tương quan tuyến tính mạnh giữa các biến độc lập. Cách làm: xây dựng ma trận hệ số tương quan cặp giữa các biến độc lập và quan sát để nhận diện độ mạnh của các tương quan giữa từng cặp biến số độc lập.

2.7.3. Hệ quả của đa cộng tuyến:

- Không ước lượng được mô hình
- Gặp các cảnh báo trong khi xây dựng mô hình Machine Learning

2.7.4. Cách giải quyết đa cộng tuyến:

Dùng Principal Component Analysis (PCA), dùng kiến thức Eigenvector (vector riêng) và Eigenvalue (trị riêng) với ý tưởng chính là Tìm một hệ trực chuẩn mới sao cho trong hệ này, các thành phần quan trọng nhất nằm trong ***K*** thành phần đầu tiên.



Hình 2.8. Cách thức hoạt động của PCA

Dữ liệu ban đầu có thể tính được xấp xỉ theo dữ liệu mới như sau:

$$\boldsymbol{x} \approx \boldsymbol{U_k Z} + \boldsymbol{x}$$

Trong đó:

- $\boldsymbol{U_k}$ là ma trận có các cột tạo thành một hệ trực giao, được xây dựng với K vector riêng ứng với K trị riêng;
- $\boldsymbol{Z} = \boldsymbol{U_K^T \widehat{X}}$ với $\widehat{\boldsymbol{X}}$ là dữ liệu ban đầu đã chuẩn hóa xuống không gian con tìm được;
- $\boldsymbol{x}$ là vector kì vọng của toàn bộ dữ liệu;

Chương 3
MỘT SỐ MÔ HÌNH MACHINE LEARNING PHỔ BIẾN

3.1. Linear Regression/Ordinary least squares

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.

3.1.1. Giới thiệu

Một căn nhà rộng $\boldsymbol{x_1}\ \boldsymbol{m^2}$, có $\boldsymbol{x_2}$ phòng ngủ và cách trung tâm thành phố $\boldsymbol{x_3}\ \mathbf{km}$ có giá là bao nhiêu. Giả sử 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, có thể dự đoán được giá của căn nhà đó không? Nếu có thì hàm dự đoán $\boldsymbol{y = f(x)}$ sẽ có dạng như thế nào. Ở đây $\boldsymbol{x = [x_1, x_2, x_3]}$ là một vector hàng chứa thông tin đầu vào, $\boldsymbol{y}$ là một số vô hướng (scalar) biểu diễn đầu ra (tức giá của căn nhà trong ví dụ này).

Một cách đơn giản nhất, có thể thấy rằng:

- Diện tích nhà càng lớn thì giá nhà càng cao.
- Số lượng phòng ngủ càng lớn thì giá nhà càng cao.
- 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:

$$\boldsymbol{y \approx f(x) = \hat{y}}$$

$$\boldsymbol{f(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0}$$

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

3.1.2. Dạng của Linear Regression

$$\boldsymbol{y \approx \bar{x}\, w = \hat{y}}$$

Trong đó:

- $\mathbf{w} = \boldsymbol{\{w_0, w_1, w_2, w_3\}}$ là vector (cột) cần phải tối ưu;
- $\boldsymbol{\bar{x} = \{1, x_1, x_2, x_3\}}$ là vector (hàng) dữ liệu đầu vào, số $\mathbf{1}$ được đặt ở đầu để phép tính đơn giản và thuận tiện hơn cho việc tính toán;

3.1.3. Sai số dự đoán

Ta mong muốn rằng sự sai khác $\boldsymbol{e}$ giữa giá trị thực $\boldsymbol{y}$ và giá trị dự đoán $\boldsymbol{\hat{y}}$ là nhỏ nhất. Nói cách khác, muốn giá trị sau đây càng nhỏ càng tốt:

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

Trong đó: hệ số $\boldsymbol{\frac{1}{2}}$ để thuận tiện cho việc tính toán (khi tính đạo hàm thì số $\boldsymbol{\frac{1}{2}}$ sẽ bị triệt tiêu). Còn $\boldsymbol{e^2}$ vì $\boldsymbol{e^2 = y - \hat{y}}$, việc ta lấy bình phương cũng là để thuận tiện trong việc đạo hàm;

3.1.4. Hàm mất mát (Cost Function /Lost Function)

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

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

Hàm số $\boldsymbol{L(w)}$ được gọi là **hàm mất mát** (cost function/loss function) của bài toán Linear Regression. Ta luôn mong muốn rằng sự mất mát (sai số) là nhỏ nhất, điều đó đồng nghĩa với việc tìm vector hệ số $\boldsymbol{w}$ sao cho giá trị của hàm mất

mát này càng nhỏ càng tốt. Giá trị của $\boldsymbol{w}$ làm cho hàm mất mát đạt giá trị nhỏ nhất được gọi là *điểm tối ưu* (optimal point).

$$\boldsymbol{w^* = argmin_w\, L(w)}$$

Trước khi đi tìm lời giải, ta đơn giản hóa phép toán trong phương trình hàm mất mát $(\mathbf{1})$. Đặt $\boldsymbol{y = [y_1; y_2; \dots; y_N]}$ là một vector cột chứa tất cả các *output* của *training data*. $\boldsymbol{\bar{X} = [\bar{x}_1; \bar{x}_2; \dots; \bar{x}_N]}$ 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 $\boldsymbol{L(w)}$ được viết dưới dạng ma trận đơn giản hơn:

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

$$\boldsymbol{= \frac{1}{2}|| \mathbf{y} - \bar{X}\mathbf{w}||_2^2} \quad (\mathbf{2})$$

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

3.1.5. Nghiệm trên bài toán Linear Regression

Để tìm nghiệm cho bài toán tối ưu là giải phương trình đạo hàm (gradient) bằng $\mathbf{0}$.

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

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

Phương trình đạo hàm bằng $\mathbf{0}$ tương đương với:

$$\boldsymbol{\bar{X}^T\bar{X}w = \bar{X}^T y \triangleq \mathbf{b}} \quad (\mathbf{3})$$

Ký hiệu $\mathbf{\bar{X}^T y \triangleq b}$ tức là đặt $\mathbf{\bar{X}^T y}$ bằng $\mathbf{b}$

Nếu ma trận vuông $\mathbf{A = \bar{X}^T\bar{X}}$ khả nghịch (non-singular hay invertible) thì phương trình $(\mathbf{3})$ có nghiệm duy nhất: $\mathbf{w = A^{-1}b}$.

Nếu ma trận vuông $\boldsymbol{A}$ không khả nghịch (có định thức bằng 0), thì phương trình $(\mathbf{3})$ vô nghiệm hoặc vô số nghiệm. Khi đó ta sẽ sử dụng khái niệm **pseudo inverse** (giả nghịch đảo) $\boldsymbol{A^{\tau}}$ **(A dagger)**. Là trường hợp tổng quát của nghịch đảo khi ma trận không khả nghịch, hoặc không vuông.

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

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

3.1.6. Ví dụ trên Python

Ta sẽ lấy ví dụ đơn giản về giải về việc giải bài toán Linear Regression trong Python. Ta cũng sẽ so sánh nghiệm của bài toán khi giải theo phương trình $(\mathbf{4})$ và nghiệm tìm được khi dùng thư viện ***scikit-learn*** của Python. Để đơn giản ta sẽ dữ liệu với đầu vào chỉ có 1 chiều (1 giá trị).

Bài toán dự đoán cân nặng của một bạn trong lớp ĐHCN1A dựa vào chiều cao của bạn ấy (Trên thực tế, tất nhiên là không, vì cân nặng còn phụ thuộc vào nhiều yếu tố khác nữa, thể tích chẳng hạn).

Họ và Tên	Chiều Cao	Cân Nặng
Trịnh Đình Phúc	168	50
Nguyễn Đình Thái	169	55
Nguyễn Đức Tiến	180	62
Trần Trung Hiếu	176	64
Trịnh Văn Bình	161	53
Lê Thị Hoài Thuận	163	46
Trần Thanh Phương	172	65
Lê Huy Hoàng	170	58
Bùi Lý Hải Đăng	173	73
Huỳnh Diệp Phụng	163	55
Lê Hùng Phú	173	74
Huỳnh Quốc Đạt	169	52
Phạm Hữu Danh	172	71
Nguyễn Duy Thanh Tùng	172	**?**

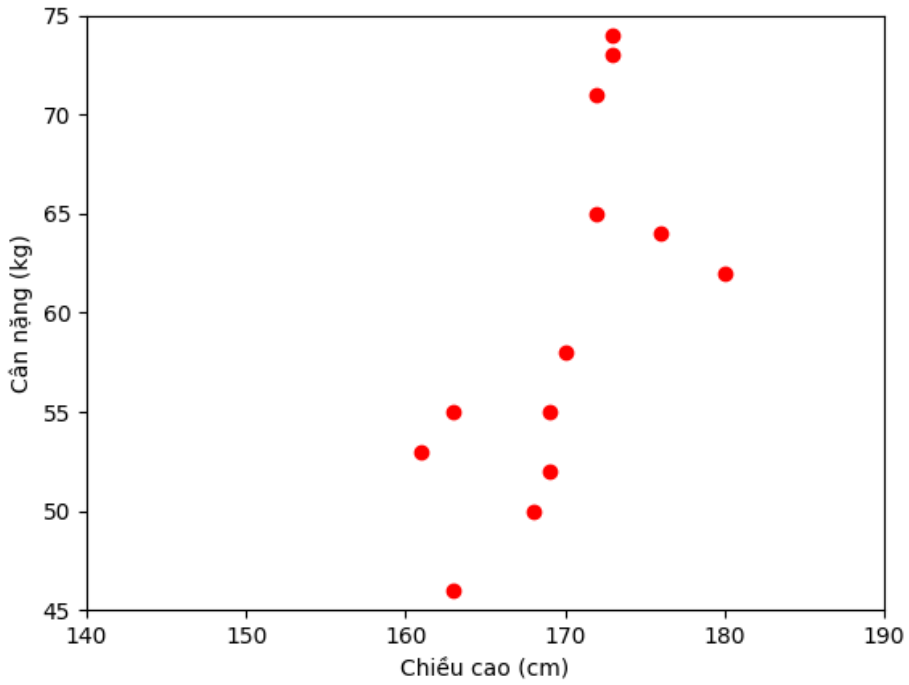
Bảng 3.1. Cân nặng và chiều cao lớp ĐHCN1A

Để kiểm tra độ chính xác của model tìm được, ta sẽ giữ lại hàng tên "Nguyễn Duy Thanh Tùng" để kiểm thử, các hàng còn lại được sử dụng để huấn luyện (train) model.

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.
import numpy as np
import matplotlib.pyplot as plt
# chiều cao (cm)
X = np.array([[168,169,180,176,161,163,172,170,173,163,173,169,172]]).T
# cân nặng (kg)
y = np.array([[ 50 ,55 ,62, 64, 53 , 46, 65, 58, 73, 55, 74, 52, 71]]).T
# Visualize data
plt.plot(X, y, 'ro')
plt.axis([140, 190, 45, 75])
plt.xlabel('Chiều cao (cm)')
plt.ylabel('Cân nặng (kg)')
plt.show()
```

Kết quả:



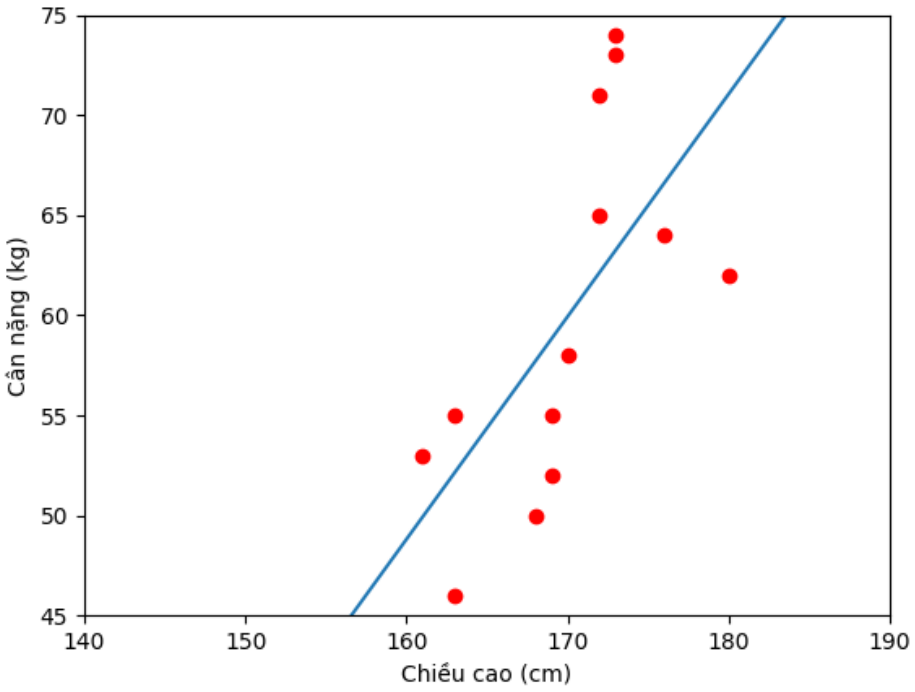
Hình 3.1. Hiển thị dữ liệu về chiều cao & cân nặng trên đồ thị

Nghiệm theo công thức:

Công thức: **Cân_nặng = w_1*(chiều_cao) + w_0**

Tiếp theo, ta sẽ tính toán các hệ số **w_1** và **w_0** dựa vào công thức (**4**)

```
# xây dựng X feature & reshape
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1)
# tính các trọng số A,b,w
A = np.dot(Xbar.T, Xbar) #np.dot tích trong 2 ma trận
b = np.dot(Xbar.T, y)
w = np.dot(np.linalg.pinv(A), b)  #pseudo-inverse
# chuẩn bị cho fitting line
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(145,190,2) #chia ra 2 khoảng từ 145 đến 190 (vì chiều
cao nằm trong khoảng đó)
y0 = w_0 + w_1*x0
print("Phương trình sau khi học được là: y ={}x {}".format(w_1,w_0))
```



Hình 3.2. Mô hình Linear Regression sau khi học xong

Nhận xét: từ đồ thị trên ta thấy đường màu xanh nằm khá gần với các đường màu đỏ, tức mô hình Linear Regression hoạt động tốt với tập dữ liệu training. Tiếp theo, ta sẽ dự đoán cân nặng của bạn "Nguyễn Duy Thanh Tùng".

```
y1 = w_1*172 + w_0
print("Phương trình sau khi học được là: y ={}x {}".format(w_1,w_0))
print( u'Chiều cao của Thanh Tùng là 172, cân nặng dự đoán: %.2f
(kg), cân nặng thực tế là: 60 (kg)'  %(y1) )
```

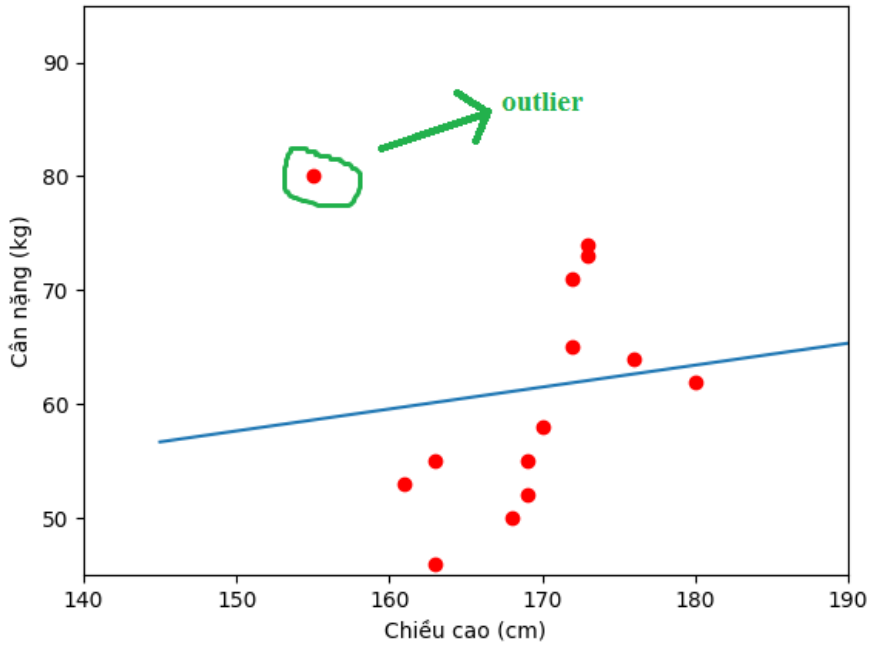
Kết quả:

```
Chiều cao của Thanh Tùng là 172, cân nặng dự đoán: 62.16 (kg),
cân nặng thực tế là: 60 (kg)
```

Nhận xét: kết quả dự đoán tương đối chính xác.

3.1.7. 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 (155 cm, 80kg) thì kết quả sẽ sai khác đi rất nhiều. Xem hình dưới đây:



Hình 3.3: Ảnh hưởng của Outliers

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).

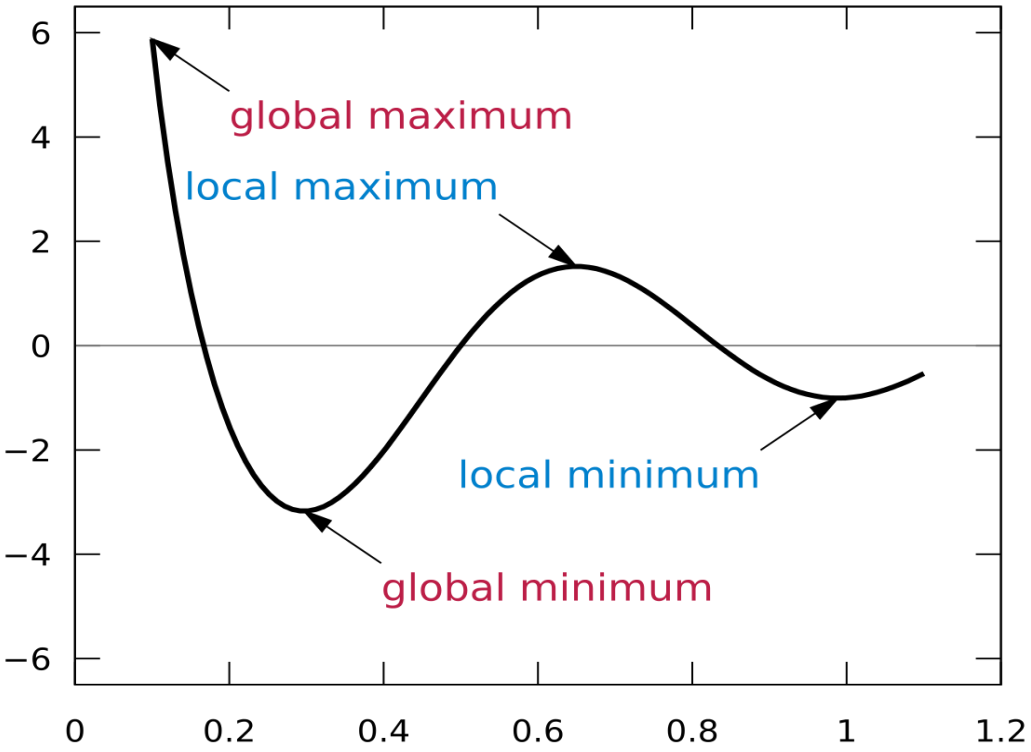
Hạn chế thứ hai của Linear Regression là nó không biểu diễn được các mô hình phức tạp. Mặc dù trong phần trên, ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa outcome và input không nhất thiết phải là tuyến tính,

nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế, những mô hình có các hàm như $\boldsymbol{x^2}, \boldsymbol{sin(x)}, \boldsymbol{log(x)}$.

3.2. Gradient Decent

3.2.1. Giới thiệu

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc đôi khi là lớn nhất) của một hàm số nào đó. Ví dụ như các hàm mất mát trong Linear Regression và K-means Clustering. Nhìn chung, việc tìm global minimum của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm local minimum, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.



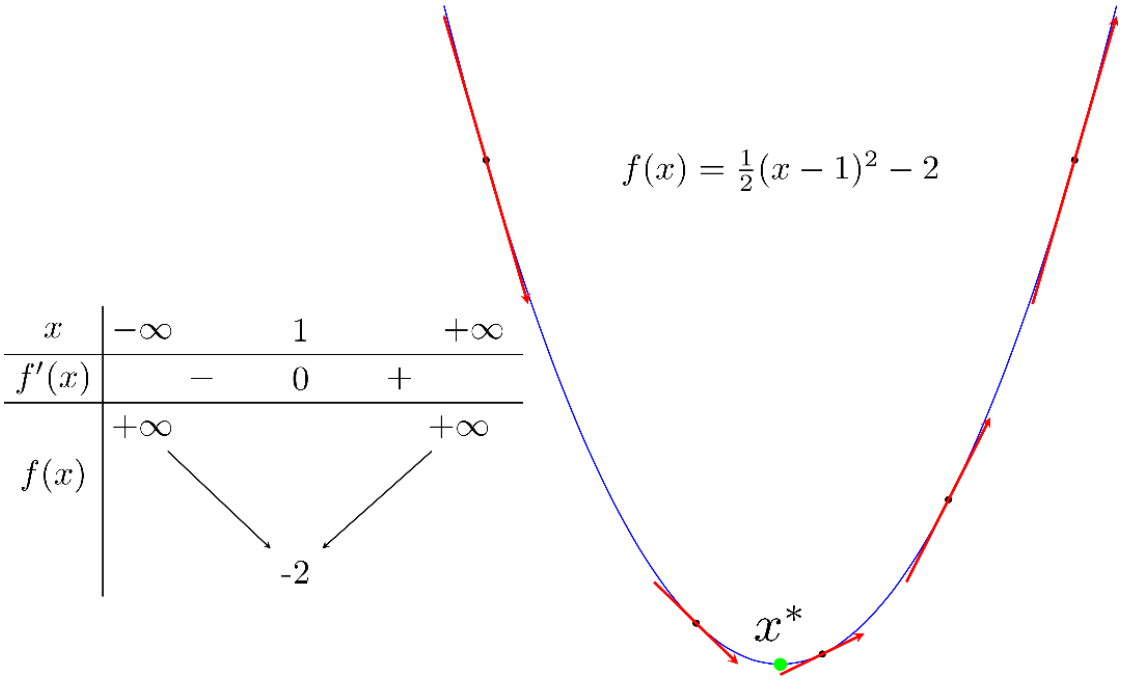
Hình 3.4. Biểu diễn các điểm local và global

Các điểm local minimum là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có

giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (GD) – Đạo hàm ngược và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

3.2.2. Gradient Decent cho hàm một biến



Hình 3.5. Parapol của hàm f(x)

Giả sử $\boldsymbol{x_t}$ là điểm ta tìm được sau vòng lặp **t**. Ta cần tìm một thuật toán để đưa $\boldsymbol{x_t}$ về càng gần $\boldsymbol{x}^*$ càng tốt.

Từ hình trên ta thấy: Nếu đạo hàm của hàm số tại $\boldsymbol{x_t}$: $\boldsymbol{f'(x_t) > 0}$ thì $\boldsymbol{x_t}$ nằm về bên phải so với $\boldsymbol{x}^*$ (và ngược lại). Để điểm tiếp theo $\boldsymbol{x_{t+1}}$ gần với $\boldsymbol{x}^*$hơn, ta cần di chuyển $\boldsymbol{x_t}$ về phía bên trái, tức về phía âm. Nói cách khác, ta cần di chuyển ngược dấu với đạo hàm

$$\boldsymbol{x_{t+1} = x_t + \Delta}$$

Trong đó:

- Δ là một đại lượng ngược dấu với đạo hàm $\mathbf{f'(x_t)}$;
- $\mathbf{x_t}$ càng xa $\mathbf{x}^*$ về phía bên phải thì $\mathbf{f'(x_t)}$ càng lớn hơn 0 (và ngược lại);

Vậy, lượng di chuyển Δ, tỉ lệ thuận với $-\mathbf{f'(x_t)}$.

Từ 2 nhận xét trên cho ta một cách cập nhật đơn giản:

$$\boldsymbol{x_{t+1} = x_t - \eta f'(x_t)}$$

Trong đó: $\boldsymbol{\eta}$ (đọc là eta) là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc ta phải đi ngược với đạo hàm;

3.2.3. Ví dụ trên python

Xét hàm số $\boldsymbol{f(x) = x^2 + 5sin(x)}$ với đạo hàm $\boldsymbol{f'(x) = 2x + 5cos(x)}$**.** Giả sử bắt đầu từ một điểm $\boldsymbol{x_0}$ nào đó, tại vòng lặp thứ $\boldsymbol{t}$**,** ta sẽ cập nhật như sau:

$$\boldsymbol{x_{t+1} = x_t - \eta(2x_t + 5cos(x_t))}$$

Các hàm số:

- **Grad_Fx** để tính đạo hàm;
- **Cost** để tính giá trị của hàm số. Hàm này không sử dụng trong thuật toán nhưng thường được dùng để kiểm tra việc tính đạo hàm của đúng không hoặc để xem giá trị của hàm số có giảm theo mỗi vòng lặp hay không;
- Hàm **Gradient_Decent** là phần chính thực hiện thuật toán Gradient Desent nêu phía trên. Đầu vào của hàm số này là learning rate ($\boldsymbol{\eta}$) và điểm bắt đầu. Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ;

Để mô tả thuật toán Gradient Decent với hàm số $\boldsymbol{f(x)}$**,** ta sẽ khởi tạo các điểm $\boldsymbol{x_0}$ khác nhau là

- $\boldsymbol{x_0 = -5}$
- $\boldsymbol{x_0 = 5}$

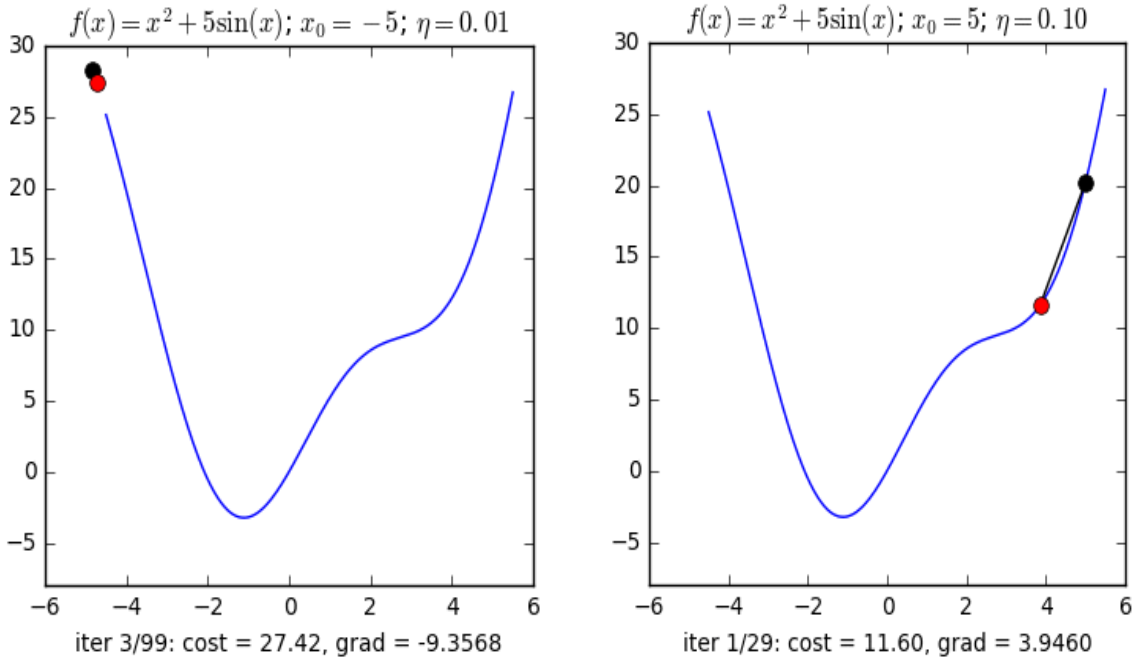
```
import numpy as np

def grad_Fx(x):
    return 2*x+ 5*np.cos(x)
def cost(x):
    return x**2 + 5*np.sin(x)
def Gradient_Decent(eta, x0):
    x = [x0]
    for it in range(100):
        x_new = x[-1] - eta*grad_Fx(x[-1])
        if abs(grad_Fx(x_new)) < 1e-3: #Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ (0.001)
            break
        x.append(x_new)
    return (x, it)
(x1, it1) = Gradient_Decent(.1, -5) # đầu vào, learning rate = 0.1, điểm bắt đầu bằng -5
(x2, it2) = Gradient_Decent(.1,  5) # đầu vào, learning rate = 0.1, điểm bắt đầu bằng 5
print('với x1 = %f, cost = %f, hội tụ sau %d bước lặp'%(x1[-1], cost(x1[-1]), it1))
print('với x2 = %f, cost = %f, hội tụ sau %d bước lặp'%(x2[-1], cost(x2[-1]), it2))
```

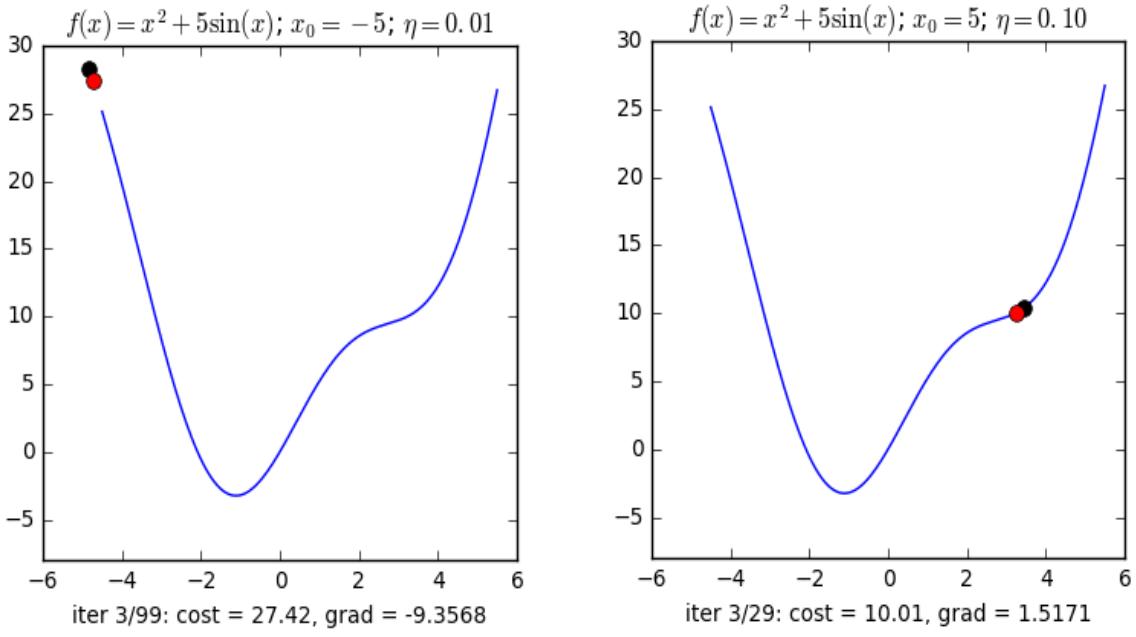
Kết quả:

```
với x1 = -1.110667, cost = -3.246394, hội tụ sau 11 bước lặp
với x2 = -1.110341, cost = -3.246394, hội tụ sau 29 bước lặp
```

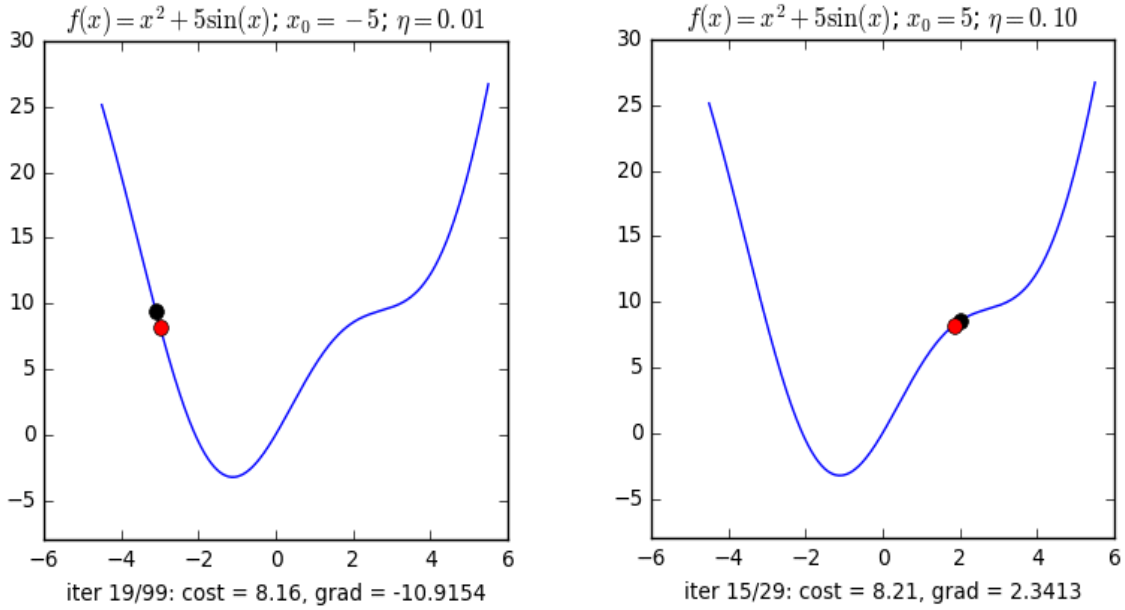
Vậy là với các điểm ban đầu khác nhau, thuật toán của ta tìm được nghiệm gần giống nhau, mặc dù với tốc độ hội tụ khác nhau. Dưới đây là hình ảnh minh họa thuật toán GD cho bài toán này.



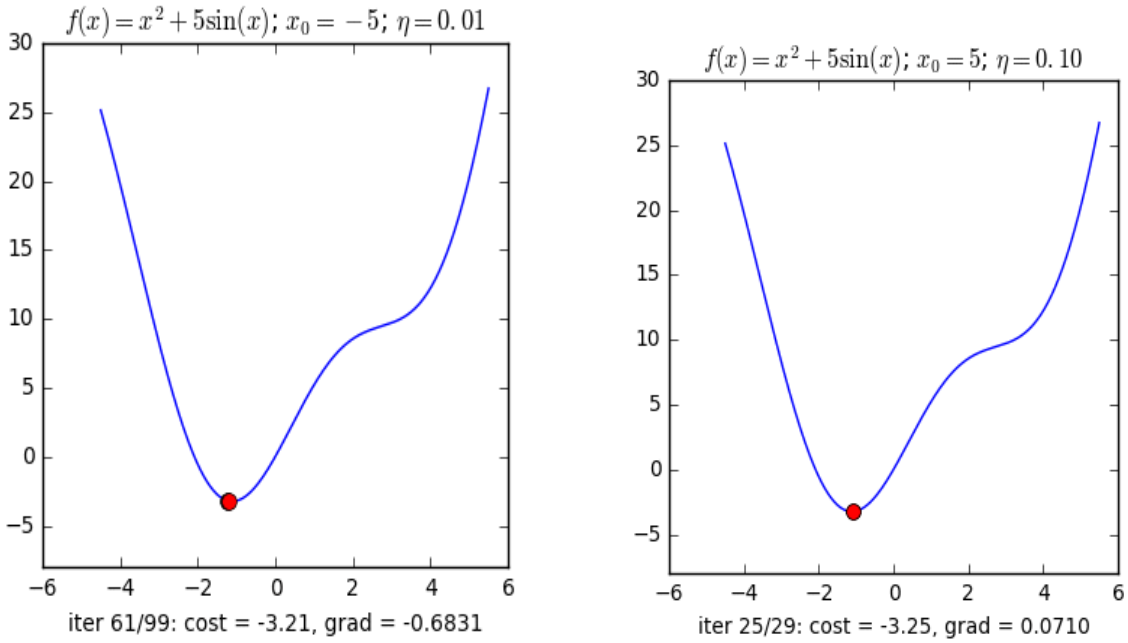
Hình 3.6. Minh họa thuật toán Gradient Decent – Bước 1



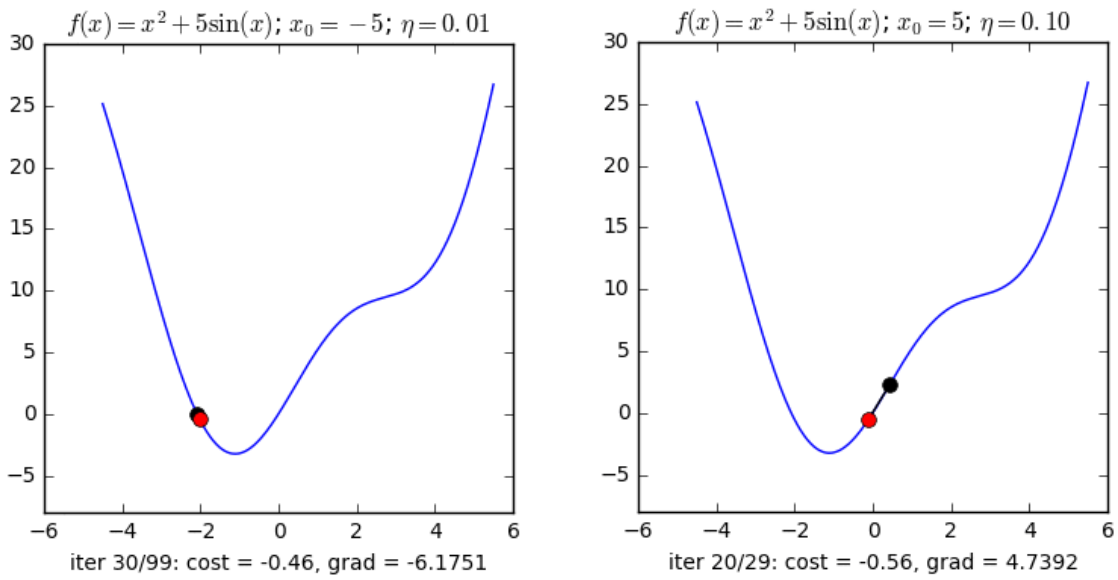
Hình 3.7. Minh họa thuật toán Gradient Decent – Bước 2



Hình 3.8. Minh họa thuật toán Gradient Decent – Bước 3



Hình 3.9. Minh họa thuật toán Gradient Decent – Bước 4



Hình 3.10. Minh họa thuật toán Gradient Decent – Bước 5

Từ hình minh họa trên ta thấy rằng ở hình bên trái, tương ứng với $\boldsymbol{x_0 = -5}$, nghiệm hội tụ nhanh hơn, vì điểm ban đầu $\boldsymbol{x_0}$ gần với nghiệm $\boldsymbol{x^* \approx -1}$ hơn. Hơn nữa, với $\boldsymbol{x_0 = -5}$ ở hình bên phải, đường đi của nghiệm có chứa một khu vực có đạo hàm khá nhỏ gần điểm có hoành độ bằng 2. Điều này khiến cho thuật toán "la cà" ở đây khá lâu. Khi vượt qua được điểm này thì mọi việc diễn ra rất tốt đẹp.

Nhận xét:

- Với các điểm ban đầu khác nhau, thuật toán tìm được nghiệm gần giống nhau, mặc dù với tốc độ hội tụ khác nhau.
- Tốc độ hội tụ của GD không những phụ thuộc vào điểm khởi tạo ban đầu mà còn phụ thuộc vào learning rate.

Việc lựa chọn learning rate rất quan trọng trong các bài toán thực tế. Việc lựa chọn giá trị này phụ thuộc nhiều vào từng bài toán và phải làm một vài thí nghiệm để chọn ra giá trị tốt nhất. Ngoài ra, tùy vào một số bài toán, GD có thể làm việc hiệu quả hơn bằng cách chọn ra learning rate phù hợp hoặc chọn learning rate khác nhau ở mỗi vòng lặp

3.2.4. Gradient Decent cho hàm nhiều biến

Giả sử ta cần tìm global minimum cho hàm $\boldsymbol{f(\theta)}$ trong đó $\boldsymbol{\theta}$ (theta) là một vector, thường được dùng để ký hiệu tập hợp các tham số của một mô hình cần tối ưu (trong Linear Regression thì các tham số chính là hệ số $\boldsymbol{w}$). Đạo hàm của hàm số đó tại một điểm $\boldsymbol{\theta}$ bất kỳ được ký hiệu là $\boldsymbol{\nabla_\theta f(\theta)}$ (hình tam giác ngược đọc là *nabla*). Tương tự như hàm 1 biến, thuật toán GD cho hàm nhiều biến cũng bắt đầu bằng một điểm dự đoán $\boldsymbol{\theta_0}$, sau đó, ở vòng lặp thứ $\boldsymbol{t}$, quy tắc cập nhật là:

$$\boldsymbol{\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)}$$

3.2.5. Stopping Criteria (điều kiện dừng)

Khi nào thì ta biết thuật toán đã hội tụ và dừng lại?

Trong thực nghiệm, có một vài phương pháp như dưới đây:

- Giới hạn số vòng lặp: đây là phương pháp phổ biến nhất và cũng để đảm bảo rằng chương trình chạy không quá lâu. Tuy nhiên, một nhược điểm của cách làm này là có thể thuật toán dừng lại trước khi đủ gần với nghiệm.
- So sánh gradient của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Phương pháp này cũng có một nhược điểm lớn là việc tính đạo hàm đôi khi trở nên quá phức tạp (ví dụ như khi có quá nhiều dữ liệu).
- So sánh giá trị của hàm mất mát của nghiệm tại hai lần cập nhật liên tiếp, khi nào giá trị này đủ nhỏ thì dừng lại. Nhược điểm của phương pháp này là nếu tại một thời điểm, đồ thị hàm số có dạng bằng phẳng tại một khu vực nhưng khu vực đó không chứa điểm local minimum (khu vực này thường được gọi là saddle points), thuật toán cũng dừng lại trước khi đạt giá trị mong muốn.
- Trong SGD và mini-batch GD, cách thường dùng là so sánh nghiệm sau một vài lần cập nhật.

3.2.6. Biến thể của Gradient Descent

3.2.6.1. Batch Gradient Descent

Thuật toán Gradient Descent còn được gọi là Batch Gradient Descent. Batch ở đây được hiểu là tất cả, tức khi cập nhật $\boldsymbol{\theta} = \boldsymbol{w}$, ta sử dụng tất cả các điểm dữ liệu $\boldsymbol{x_i}$.

Cách làm này có một vài hạn chế đối với cơ sở dữ liệu quá lớn. Việc phải tính toán lại đạo hàm với tất cả các điểm này sau mỗi vòng lặp trở nên cồng kềnh và không hiệu quả.

3.2.6.2. Stochastic Gradient Descent(SGD)

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu $\boldsymbol{x_i}$ rồi cập nhật $\boldsymbol{\theta}$ dựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu, sau đó lặp lại quá trình trên. Thuật toán rất đơn giản này trên thực tế lại làm việc rất hiệu quả.

Mỗi lần duyệt một lượt qua tất cả các điểm trên toàn bộ dữ liệu được gọi là một epoch. Với GD thông thường thì mỗi epoch ứng với 1 lần cập nhật $\boldsymbol{\theta}$. Với SGD thì mỗi epoch ứng với $\boldsymbol{N}$ lần cập nhật $\boldsymbol{\theta}$ với $\boldsymbol{N}$ lần cập nhật $\boldsymbol{\theta}$ với $\boldsymbol{N}$ là số điểm dữ liệu. Nhìn vào một mặt, việc cập nhật từng điểm một như thế này có thể làm giảm đi tốc độ thực hiện 1 epoch. Nhưng nhìn vào một mặt khác, SGD chỉ yêu cầu một lượng epoch rất nhỏ (thường là 10 cho lần đầu tiên, sau đó khi có dữ liệu mới thì chỉ cần chạy dưới một epoch là đã có nghiệm tốt). 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, tức online learning.

3.2.7. Thứ tự lựa chọn điểm dữ liệu

Một điểm cần lưu ý đó là: sau mỗi epoch, ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên. Việc này cũng ảnh hưởng tới hiệu năng của SGD.

Một cách toán học, quy tắc cập nhật của SGD là:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\eta \nabla_{\theta} J(\theta; x_i; y_i)}$$

Trong đó: $\mathbf{J(\theta; x_i; y_i)}$ là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là $(\mathbf{x_i; y_i})$;

3.3. K-Means

3.3.1. Giới thiệu

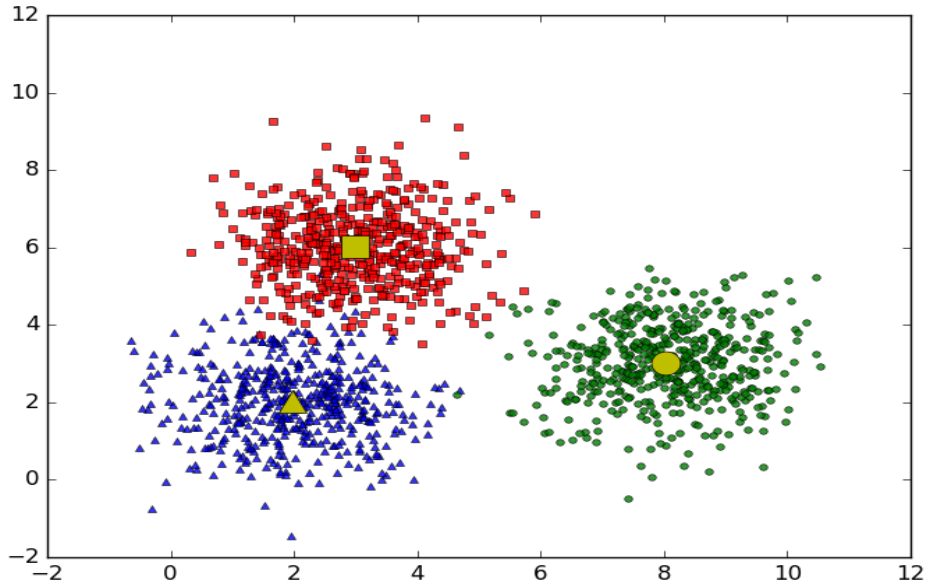
Trong thuật toán K-means clustering, ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.

3.3.2. 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. Lúc đó ta có thể sử dụng các phương pháp phân cụm, ví dụ như K-means.

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) lại thành một cụm.



Hình 3.11. Bài toán phân cụm với 3 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.

3.3.3. Phân tích mặt toán học

Giả sử có $\boldsymbol{N}$ điểm dữ liệu là $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_N}] \in \boldsymbol{R^{d \times D}}$ và $\boldsymbol{K < N}$ là số cluster muốn phân chia. Ta cần tìm các center $\boldsymbol{m_1}, \boldsymbol{m_2}, \dots, \boldsymbol{m_K} \in \boldsymbol{R^{d \times 1}}$ và label của mỗi điểm dữ liệu.

Với mỗi điểm dữ liệu $\boldsymbol{x_i}$ đặt $\boldsymbol{y_i} = [\boldsymbol{y_{i1}}, \boldsymbol{y_{i2}}, \dots, \boldsymbol{y_{ik}}]$ là label vector của nó, trong đó nếu $\boldsymbol{x_i}$ được phân vào cluster $\boldsymbol{k}$ thì $\boldsymbol{y_{ik} = 1}$ và $\boldsymbol{y_{ij} = 0, \forall j \neq k}$. Điều này có nghĩa là có đúng một phần tử của vector $\boldsymbol{y_i}$ là bằng 1 (tương ứng với cluster của $\boldsymbol{x_i}$) các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là $[\boldsymbol{1}, \boldsymbol{0}, \boldsymbol{0}, \dots, \boldsymbol{0}]$ thì nó thuộc vào cluster 1, là $[\boldsymbol{0}, \boldsymbol{1}, \boldsymbol{0}, \dots, \boldsymbol{0}]$ thì nó thuộc vào cluster 2,...Cách mã hóa label của dữ liệu như thế này được gọi là biểu diễn one-hot-encoding.

Ràng buộc của $\boldsymbol{y_i}$ có thể viết dưới dạng toán học như sau:

$$\boldsymbol{y_{ik}} = \{\boldsymbol{0}, \boldsymbol{1}\}, \sum_{\boldsymbol{k=1}}^{\boldsymbol{k}} \boldsymbol{y_{ik}} = \boldsymbol{1} \quad (\boldsymbol{5})$$

3.3.4. Thuật toán

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

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

Bước 1: Chọn $\boldsymbol{K}$ điểm bất kỳ làm các ***center*** ban đầu.

Bước 2: Phân mỗi điểm dữ liệu vào ***cluster*** có ***center*** gần nó nhất.

Bước 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.

Bước 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.

Bước 5: Quay lại bước 2.

3.3.5. Hàm mất mát và bài toán tối ưu

Nếu ta coi center $\boldsymbol{m_k}$ là center của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi $\boldsymbol{m_k}$ thì một điểm dữ liệu $\boldsymbol{x_i}$ được phân vào cluster $\boldsymbol{k}$ sẽ bị sai số là $\boldsymbol{(x_i - m_k)}$. Ta mong muốn sai số này có giá trị tuyệt đối nhỏ nhất nên sai số này có giá trị tuyệt đối nhỏ nhất nên ta sẽ tìm cách để đại lượng sau có giá trị nhỏ nhất:

$$||\boldsymbol{x_i - m_k}||_2^2$$

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

$$\boldsymbol{y_{ik}||x_i - m_k||_2^2 = \sum_{j=1}^{k} y_{ij}||x_i - m_j||_2^2}$$

Sai số cho toàn dữ liệu sẽ là:

$$\boldsymbol{L(Y, M) = \sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}||x_i - m_j||_2^2}$$

Trong đó: $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, \dots, \mathbf{y_N}], \mathbf{M} = [\mathbf{m_1}, \mathbf{m_2}, \dots, \mathbf{m_k}]$ lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. $\mathbf{L(Y, M)}$ là hàm số mất mát với ràng buộc như trong phương trình $(\mathbf{5})$;

Tóm lại, cần tối ưu bài toán sau:

$$\boldsymbol{Y, M = argmin_{Y,M} \sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij}||x_i - m_j||_2^2} \quad (\mathbf{6})$$

$$\boldsymbol{s.t: y_{ij} = \{0, 1\}\ \forall i, j;\ \sum_{j=1}^{K} y_{ij} = 1\ \forall i}$$

3.3.6. Thuật toán tối ưu hàm mất mát

Một cách đơn giản để giải bài toán $(\mathbf{6})$ là xen kẽ giải $\boldsymbol{Y}$ và $\boldsymbol{M}$ 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. Ta sẽ lần lượt giải quyết hai bài toán sau đây:

3.3.6.1. Cố định M, tìm Y

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

$$\boldsymbol{y_j} = \boldsymbol{argmin_{y_j}} \sum_{j=1}^{K} \boldsymbol{y_{ij}} \left|\left| \boldsymbol{x_i} - \boldsymbol{m_j} \right|\right|_2^2 \quad (\mathbf{7})$$

$$\boldsymbol{s.t}: \boldsymbol{y_{ij}} = \{\mathbf{0}, \mathbf{1}\}\ \forall \boldsymbol{j};\ \sum_{\boldsymbol{j=1}}^{\boldsymbol{K}} \boldsymbol{y_{ij}} = \mathbf{1}$$

Chỉ có một phần tử của vector label $\boldsymbol{y_i}$ bằng 1 nên bài toán **(7)** có thể tiếp tục viết dưới dạng đơn giản hơn:

$$\boldsymbol{j} = \boldsymbol{argmin_j} \left|\left| \boldsymbol{x_i} - \boldsymbol{m_j} \right|\right|_2^2$$

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

3.3.6.2. Cố định Y, tìm M

Giá sử ta đã xác định được label vector vector cho từng điểm dữ liệu, bài toán tìm center cho mỗi cluster được rút gọn lại thành:

$$\boldsymbol{m_j} = \boldsymbol{argmin_{m_j}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{y_{ij}} \left|\left| \boldsymbol{x_i} - \boldsymbol{m_j} \right|\right|_2^2$$

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 $\boldsymbol{m_j}$ nên sẽ tìm được giá trị nhỏ nhất và điểm tối ưu tương ứng.

Đặt $\boldsymbol{l(m_j)}$ là hàm bên trong dấu $\boldsymbol{argmin}$, ta có đạo hàm:

$$\frac{\boldsymbol{\partial l(m_j)}}{\boldsymbol{\partial m_j}} = \boldsymbol{2\sum_{i=1}^{N} y_{ij}(m_j - x_i)}$$

Giải phương trình đạo hàm bằng 0 ta có:

$$\boldsymbol{m_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij}\, x_i \;\Rightarrow\; m_j = \frac{\sum_{i=1}^{N} y_{ij} x_i}{\sum_{i=1}^{N} y_{ij}}}$$

Ta thấy $\boldsymbol{m_j}$ là trung bình cộng của các điểm trong cluster $\boldsymbol{j}$

3.4. Logistic Regression

3.4.1. Giới thiệu

Mục tiêu của hồi qui Logistic là nghiên cứu mối tương quan giữa một (hay nhiều) yếu tố nguy cơ (risk factor) và đối tượng phân tích (outcome). Chẳn hạn như đối với nghiên cứu mối tương quan giữa thói quen hút thuốc lá và nguy cơ mắc ung thư phổi thì yếu tố nguy cơ ở đây là thói quen hút thuốc lá và đối tượng phân tích ở đây là nguy cơ mắc ung thư phổi. Trong hồi qui logistic thì các đối tượng nghiên cứu thường được thể hiện qua các biến số nhị phân (binary) như xảy ra/ không xảy ra; chết/sống; có/không,… còn các yếu tố nguy cơ có thể được thể hiện qua các biến số liên tục (tuổi, huyết áp,…) hoặc các biến nhị phân (giới tính) hay các biến thứ bậc (thu nhập : cao, trung bình, thấp).

3.4.2. Ví dụ

Một nhóm 20 sinh viên dành thời gian trong khoảng từ 0 đến 6 giờ cho việc ôn thi. Thời gian ôn thi này ảnh hưởng đến xác suất sinh viên vượt qua kỳ thi như thế nào?

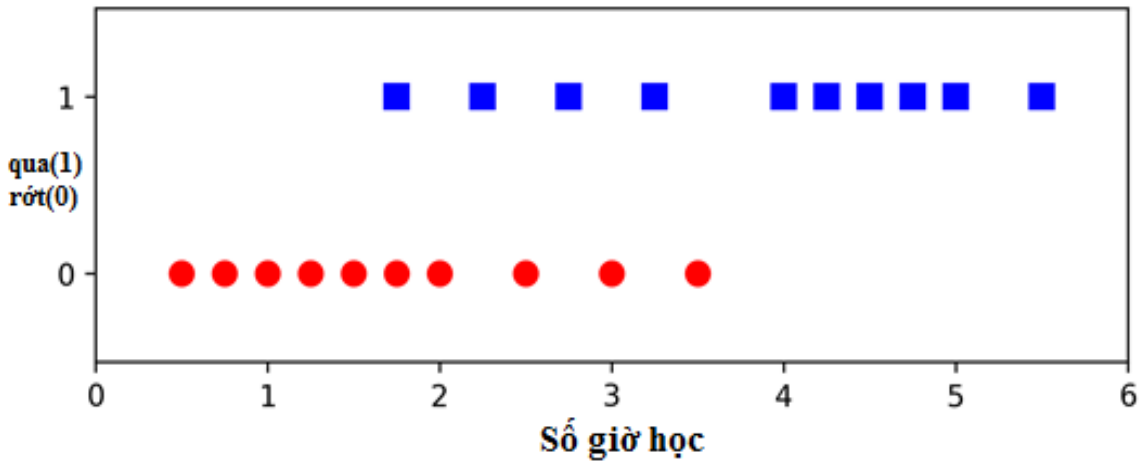
Kết quả thu được như sau:

Số giờ học	Kết quả	Số giờ học	Kết quả
0.5	0	2.75	1
0.75	0	3	0
1	0	3.25	1
1.25	0	3.5	0
1.5	0	4	1
1.75	0	4.25	1
1.75	1	4.5	1
2	0	4.75	1
2.25	1	5	1
2.5	0	5.5	1

Bảng 3.2. Số giờ học và kết quả

Mặc dù có một chút bất công khi học 3.5 giờ thì trượt, còn học 1.75 giờ thì lại đỗ, nhìn chung, học càng nhiều thì khả năng đỗ càng cao.

Biểu diễn các điểm này trên đồ thị:



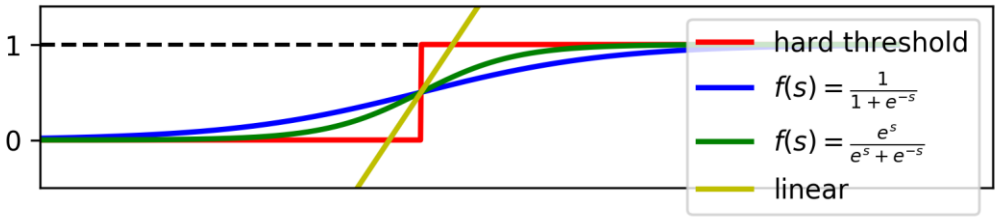
Hình 3.12. Kết quả thi dựa trên số giờ học

3.4.3. Mô hình Logistic Regression

Đầu ra dự đoán của logistic regression thường được viết chung dưới dạng:

$$f(x) = \theta(w^T x)$$

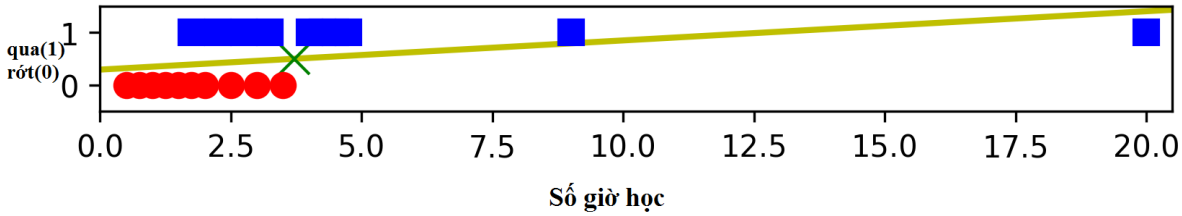
Trong đó: **θ** được gọi là logistic function. Một số activation cho mô hình tuyến tính được cho trong hình dưới đây:



Hình 3.13. Một số loại Activation Function

- **Đường màu vàng**:

Biểu diễn linear regression. Đường này không bị chặn nên không phù hợp cho bài toán này. Khi áp dụng mô hình linear regression như hình dưới đây và lấy mốc 0.5 để phân lớp, toàn bộ sinh viên thi trượt vẫn được dự đoán là trượt, nhưng rất nhiều sinh viên thi đỗ cũng được dự đoán là trượt (nếu ta coi điểm x màu xanh lục là ngưỡng cứng để đưa ra kết luận). Rõ ràng đây là một mô hình không tốt.



Hình 3.14. Mô tả Linear Regression không phù hợp

- **Đường màu đỏ:**

Rõ ràng là khi sử dụng ngưỡng cứng ở đây vì dữ liệu ở đây không tách biệt tuyến tính (linearly separable).

- **Các đường màu xanh:**

Hai đường màu xanh lá và xanh dương hợp với bài toán của ta hơn, chúng có 1 vài tính chất quan trọng sau:

1. Là hàm số liên tục nhận giá trị thực, bị chặn trong khoảng $(\mathbf{0}, \mathbf{1})$.
2. Nếu coi điểm có tung độ là 1/2 làm điểm phân chia thì các điểm càng xa điểm này về phía bên trái có giá trị càng gần 0. Ngược lại, các điểm càng xa điểm này về phía phải có giá trị càng gần 1. Điều này khớp với nhận xét rằng học càng nhiều thì xác suất đỗ càng cao và ngược lại.

3. Mượt (smooth) nên có đạo hàm mọi nơi, có thể được lợi trong việc tối ưu.

3.4.4. Sigmoid Function

$$f(s) = \frac{1}{1 + e^{-s}} \triangleq \sigma(s)$$

Đây là hàm được sử dụng nhiều nhất, vì nó bị chặn trong khoảng $(0, 1)$.

3.4.5. Tanh Function (Hyperpolic Tangent)

$$tanh(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}}$$

Hàm số này nhận giá trị trong khoảng $(-1, 1)$, với khoảng $(0, 1)$ hàm có dạng:

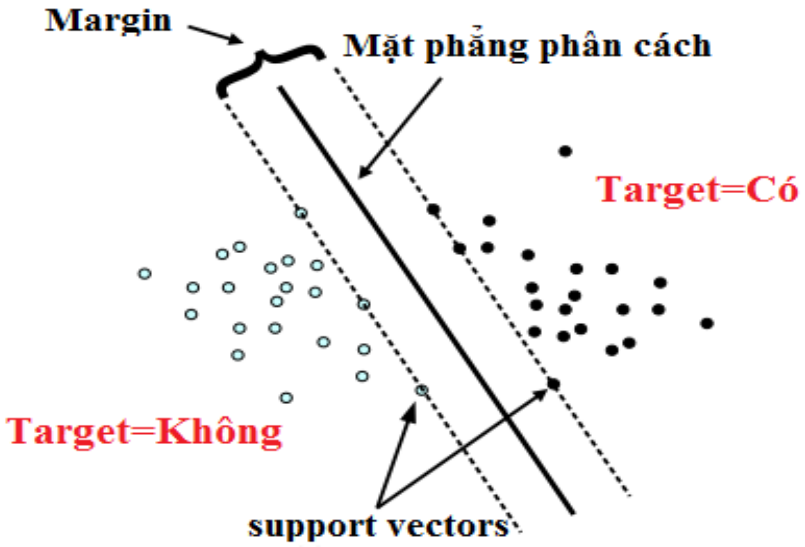
$$tanh(s) = 2\sigma(2s) - 1$$

3.5. Support Vector Machine (SVM)

3.5.1. Giới thiệu

SVM là một thuật toán Supervised, chủ yếu sử dụng cho việc bài toán Classification. Trong thuật toán này, ta vẽ đồ thị dữ liệu là các điểm trong n chiều (ở đây n là số lượng các feature) với giá trị của mỗi feature sẽ là một phần liên kết. Sau đó thực hiện tìm "đường bay" phân chia các lớp.

Đường bay, nó chỉ hiểu đơn giản là 1 đường thẳng có thể phân chia các lớp ra thành hai phần riêng biệt. Support Vectors hiểu một cách đơn giản là các đối tượng trên đồ thị tọa độ quan sát, Support Vector Machine là một biên giới để chia hai lớp tốt nhất.



Hình 3.15. Thuật toán phân lớp SVM

3.5.2. Khoảng cách từ một điểm tới một siêu phẳng

Trong không gian 2 chiều ta biết rằng khoảng cách từ một điểm có tọa độ $(\boldsymbol{y_0, y_0})$ tới đường thẳng có phương trình $\boldsymbol{w_x x + w_x y + b = 0}$ được xác định bởi:

$$\frac{|\boldsymbol{w_1 x_0 + w_2 y_0 + b}|}{\sqrt{\boldsymbol{w_1}^2 + \boldsymbol{w_2}^2}}$$

Hơn nữa, nếu ta bỏ dấu trị tuyệt đối ở tử số, có thể xác định được điểm đó nằm về phía nào của đường thẳng hay mặt phẳng đang xét. Những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu dương nằm về cùng 1 phía, những điểm làm cho biểu thức trong dấu giá trị tuyệt đối mang dấu âm nằm về phía còn lại. Những điểm nằm trên đường thẳng/măt phẳng sẽ làm cho tử số có giá trị bằng 0, tức khoảng cách bằng 0.

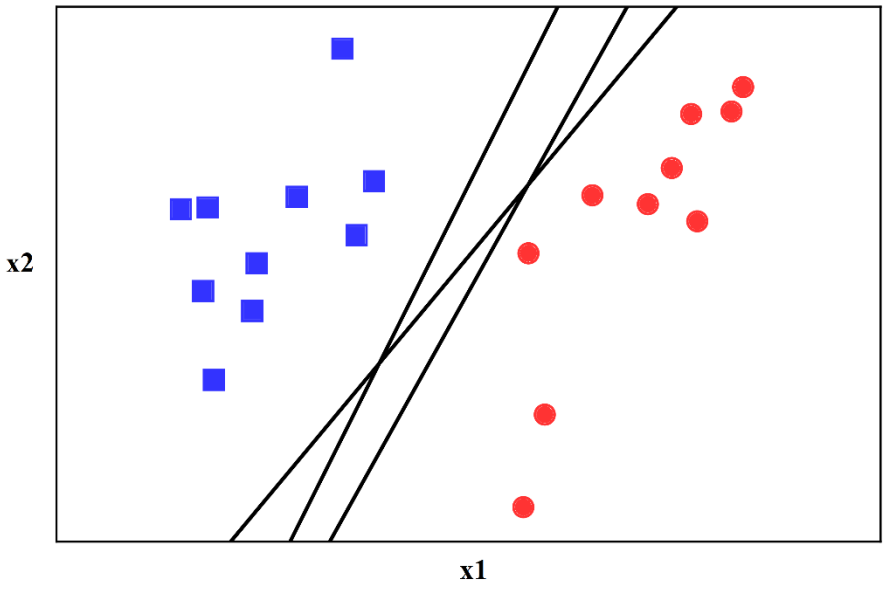
Việc này có thể được tổng quát lên không gian nhiều chiều: Khoảng cách từ một điểm (vector) có toạ độ $\boldsymbol{x_0}$ tới siêu mặt phẳng (hyperplane) có phương trình $\boldsymbol{w^T x_0 + b = 0}$ được xác định bởi:

$$\frac{|\boldsymbol{w^T x_0 + b}|}{||\boldsymbol{x}||_2}$$

Với $||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^{d} \boldsymbol{w_i^2}}$ và $\boldsymbol{d}$ là số chiều của không gian.

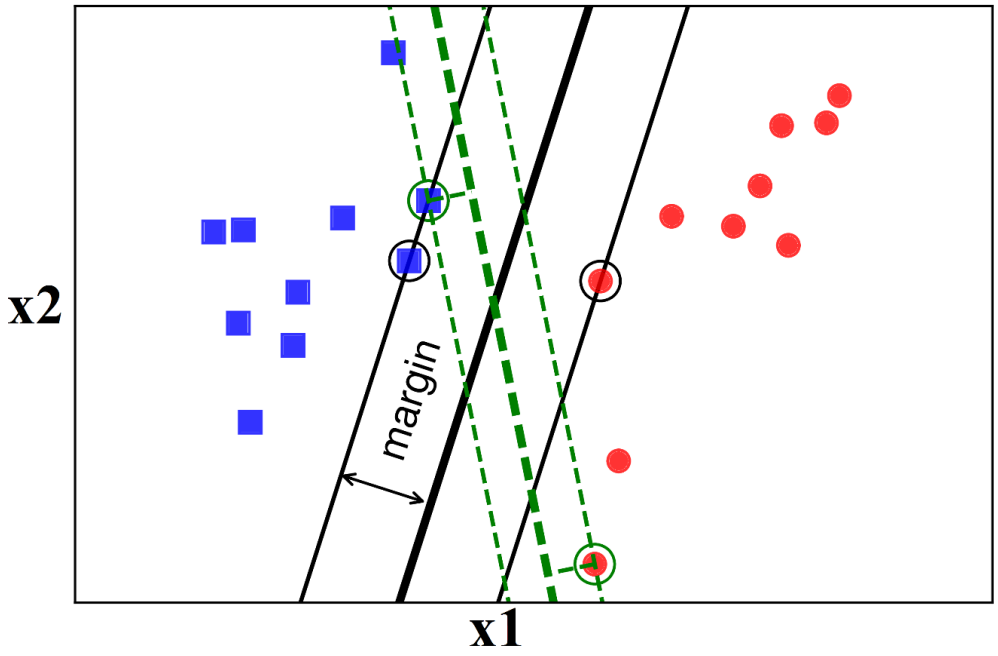
3.5.3. Phân chia hai class

Trong thuật toán Perceptron Learning Algorithm (PLA). Giả sử rằng có hai class khác nhau được mô tả bởi các điểm trong không gian nhiều chiều, hai class này linearly separable, tức tồn tại một siêu phẳng phân chia chính xác hai class đó. Hãy tìm một siêu mặt phẳng phân chia hai classes đó, tức tất cả các điểm thuộc một class nằm về cùng một phía của siêu mặt phẳng đó và ngược phía với toàn bộ các điểm thuộc class còn lại. Thuật toán PLA có thể làm được việc này nhưng nó có thể cho vô số nghiệm như dưới đây:



Hình 3.16. Các mặt phân cách hai classes linearly separable

Trong vô số các mặt phân chia đó, đâu là mặt phân chia tốt nhất theo một tiêu chuẩn nào đó. Trong ba đường thẳng minh họa có hai đường thẳng khá lệch về phía class hình tròn đỏ. Điều này có thể khiến cho lớp màu đỏ "không vui" vì lãnh thổ bị lấn nhiều quá. Cần tìm một tiêu chuẩn để đo sự "hạnh phúc" của mỗi class. Hãy xem Hình 3.17 dưới đây:



Hình 3.17. Margin của hai classes là bằng nhau và lớn nhất có thể

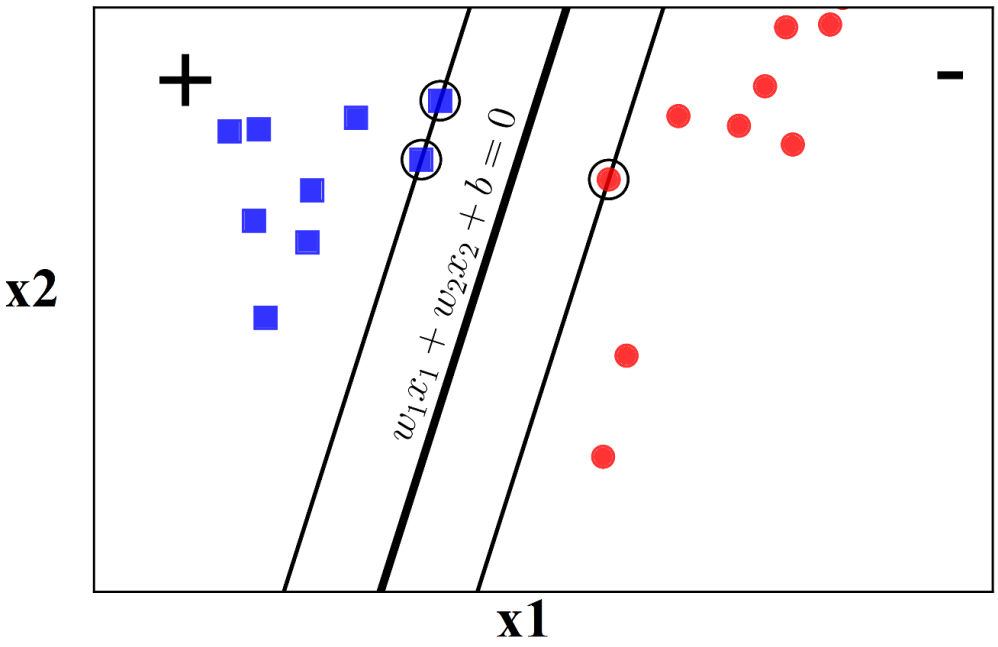
Ta cần một đường phân chia sao cho khoảng cách từ điểm gần nhất của mỗi class (các điểm được khoanh tròn) tới đường phân chia là như nhau, như thế thì mới công bằng. Khoảng cách cân bằng tính từ đường phân chia được gọi là margin.

Bài toán tối ưu trong Support Vector Machine (SVM) chính là bài toán đi tìm đường phân chia sao cho margin là lớn nhất. Đây cũng là lý do vì sao SVM còn được gọi là Maximum Margin Classifier.

3.5.4. Tối ưu cho SVM

Giả sử rằng các cặp dữ liệu của training set $\mathbf{x}, \mathbf{y}$ đi từ $\mathbf{1} \rightarrow \mathbf{n}$ là $(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_N}, \mathbf{y_N}), \dots, (\mathbf{x_N}, \mathbf{y_N})$ với vector $\mathbf{x_i} \boldsymbol{\epsilon} \mathbf{R^d}$ thể hiện đầu vào của một điểm dữ liệu và $\mathbf{y_i}$ là nhãn của điểm dữ liệu đó, $\mathbf{d}$ là số chiều của dữ liệu và $\mathbf{N}$ là số điểm dữ liệu. Giả sử rằng nhãn của mỗi điểm dữ liệu được xác định bởi $\mathbf{y_i} = \mathbf{1}$ ($\mathbf{class\ 1}$) hoặc $\mathbf{y_i} = \mathbf{-1}$ ($\mathbf{class\ 2}$).

Ta có hình mô tả như dưới đây:



Hình 3.18. Phân tích bài toán SVM

Dựa vào hình trên ta có thể thấy khoảng cách cách đều từ 2 mặt phẳng, chấm đỏ và chấm xanh là bằng nhau tính từ đường thẳng $\boldsymbol{w_1 x_1 + w_2 x_2 + b = 0}$.

Giả sử rằng các điểm vuông xanh thuộc class **1**, các điểm tròn đỏ thuộc class **-1** và mặt $\boldsymbol{w^T x + b = w_1 x_1 + w_2 x_2 + b = 0}$ là mặt phân chia giữa hai classes (Hình 3.18). Hơn nữa, class **1** nằm về phía dương, class **-1** nằm về phía âm của mặt phân chia. Nếu ngược lại, ta chỉ cần đổi dấu của **w** và $\boldsymbol{b}$.

Ta quan sát thấy một điểm quan trọng sau đây: với cặp dữ liệu $(\boldsymbol{x_N}, \boldsymbol{y_N})$ bất kì, khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{\boldsymbol{y_n(w^T x_n + b)}}{\boldsymbol{||w||_2}}$$

Điều này có thể dễ nhận thấy vì theo giả sử ở trên, $\boldsymbol{y_n}$ luôn cùng dấu với phía $\boldsymbol{x_n}$. Từ đó suy ra $\boldsymbol{y_n}$ cùng dấu với $\boldsymbol{w^T x_n + b}$, và tử số luôn là 1 số không âm.

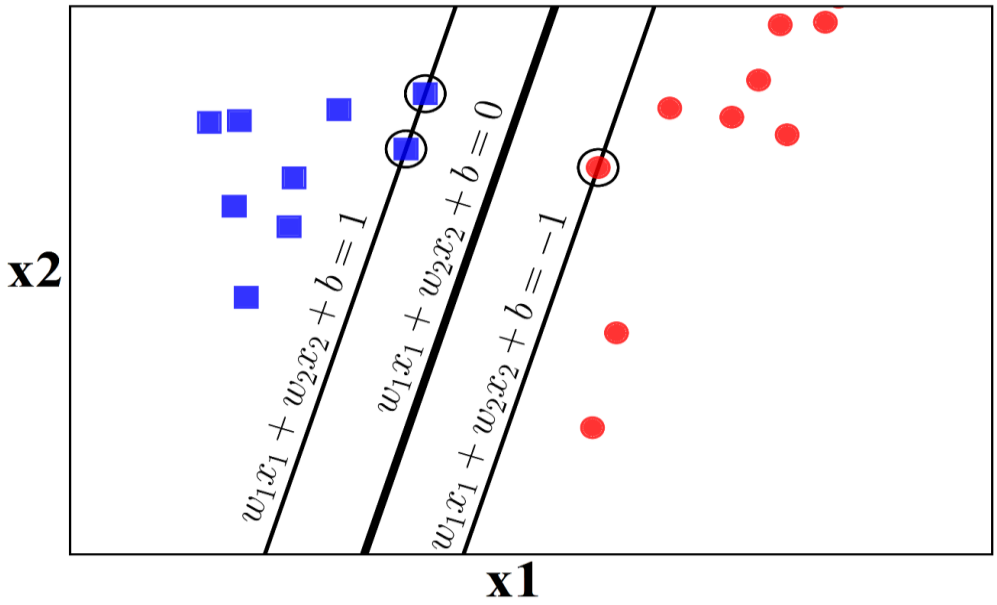
Với mặt phần chia như trên, margin được tính là khoảng cách gần nhất từ 1 điểm tới mặt đó (bất kể điểm nào trong hai classes):

$$\boldsymbol{margin = min_n \frac{y_n(w^T x_n + b)}{||w||_2}}$$

Bài toán tối ưu trong SVM chính là bài toán tìm $\boldsymbol{w}$ và **b** sao cho margin này đạt giá trị lớn nhất:

$$\boldsymbol{y_n(w^T x_n + b) = 1}$$

Với những điểm nằm gần mặt phân chia nhất như dưới đây:



Hình 3.19. Các điểm gần mặt phân cách nhất của hai classes được khoanh tròn

Như vậy, với mọi **n** ta có:

$$y_n(w^T x_n + b) \geq 1$$

Vậy bài toán tối ưu có dạng:

$$(w, b) = argmin_{w,b} \frac{1}{2} ||w||_2^2$$

$$subject\ to: 1 - y_n(w^T x_n + b) \leq 0, \forall n = 1, 2, \dots, N$$

Xác định class cho một điểm dữ liệu mới: Sau khi tìm được mặt phân cách $w^T x + b = 0$, class của bất kì một điểm nào sẽ được xác định đơn giản bằng cách:

$$class(x) = sgn(w^T x + b)$$

Trong đó: hàm sgn là hàm xác định dấu, nhận giá trị **1** nếu đối số là không âm và **-1** nếu ngược lại;

3.5.5. Ví dụ trên Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(22)
#Đầu tiên ta sẽ tạo dữ liệu giả sao cho dữ liệu linearly
separable
means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N) # class
1
X1 = np.random.multivariate_normal(means[1], cov, N) # class
-1
X = np.concatenate((X0.T, X1.T), axis = 1) # nối toàn bộ dữ
liệu
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))),
axis = 1) # nhãn
# Giải bài toán đối ngẫu Lagrange bằng sử dụng CVXOPT trong
thư viện sklearn
from cvxopt import matrix, solvers
# build K
V = np.concatenate((X0.T, -X1.T), axis = 1)
K = matrix(V.T.dot(V))
p = matrix(-np.ones((2*N, 1)))
# build A, b, G, h
G = matrix(-np.eye(2*N))
h = matrix(np.zeros((2*N, 1)))
A = matrix(y)
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(K, p, G, h, A, b)
```

Kết quả:

```
lambda =
 [[  8.54018321e-01   2.89132533e-10   1.37095535e+00
6.36030818e-10
    4.04317408e-10   8.82390106e-10   6.35001881e-10
5.49567576e-10
    8.33359230e-10   1.20982928e-10   6.86678649e-10
1.25039745e-10
    2.22497367e+00   4.05417905e-09   1.26763684e-10
1.99008949e-10
    2.13742578e-10   1.51537487e-10   3.75329509e-10
3.56161975e-10]]
```

Nhận xét: Ta nhận thấy rằng hầu hết các giá trị của lambda đều rất nhỏ, tới $\mathbf{10^9}, \mathbf{10^{10}}$. Đây chính là các giá trị bằng 0 nhưng vì sai số tính toán nên nó khác 0 một chút. Chỉ có 3 giá trị khác 0, ta dự đoán là sẽ có 3 điểm là support vectors.

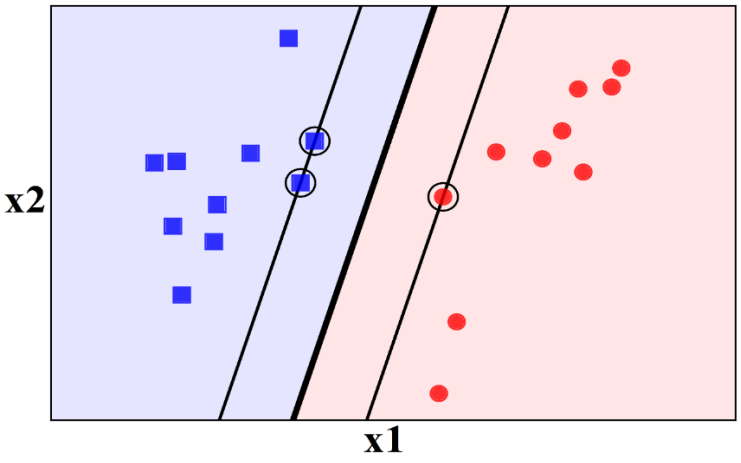
Ta đi tìm support set **S** rồi tìm nghiệm của bài toán:

```
# Khởi tạo số epsilon để so sánh lấy 3 nghiệm trên tập nghiệp
epsilon = 1e-6
S = np.where(l > epsilon)[0]
VS = V[:, S]
XS = X[:, S]
yS = y[:, S]
lS = l[S]
# calculate w and b
w = VS.dot(lS)
b = np.mean(yS.T - w.T.dot(XS))
print('giá trị của w = ', w.T)
print('giá trị của b = ', b)
```

Kết quả:

```
w =  [[-2.00984381  0.64068336]]
b =  4.66856063387
```

Minh họa kết quả:

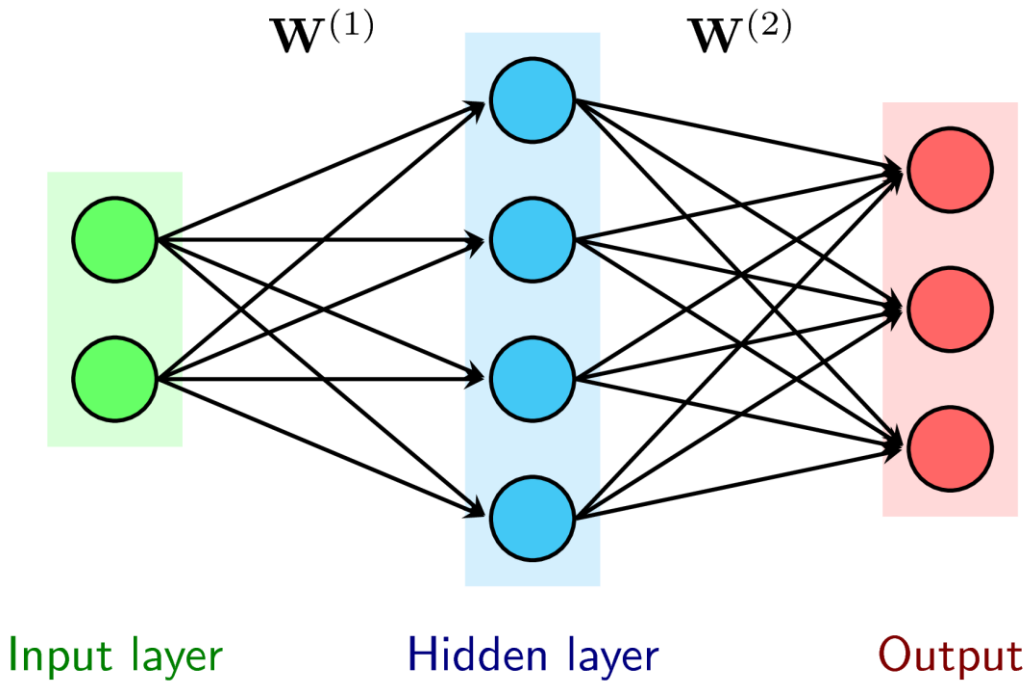


Hình 3.20. Minh họa nghiệm tìm được

3.6. Artificial Neural Network (ANN)

3.6.1. Kiến trúc tổng quát của một ANN

Artificial Neural Network - ANN (mạng nơ-ron nhân tạo) là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thống thần kinh của sinh vật, bao gồm số lượng lớn các Neuron được gắn kết để xử lý thông tin. ANN giống như bộ não con người, được học bởi kinh nghiệm (thông qua huấn luyện), có khả năng lưu giữ những kinh nghiệm hiểu biết (tri thức) và sử dụng những tri thức đó trong việc dự đoán các dữ liệu chưa biết (unseen data).

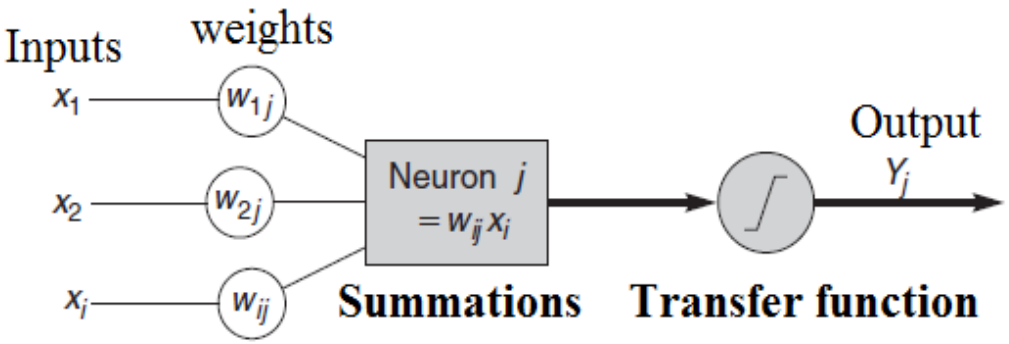


Hình 3.21. Kiến trúc tổng quát của một mạng Neuron nhân tạo

Kiến trúc chung của một ANN gồm 3 thành phần: Input Layer, Hidden Layer và Output Layer.

Trong đó, lớp ẩn (Hidden Layer) gồm các Neuron, nhận dữ liệu input từ các Neuron ở lớp (Layer) trước đó và chuyển đổi các input này cho các lớp xử lý tiếp trong một ANN có thể có nhiều Hidden Layer. Mỗi Neuron nhận các dữ liệu vào (Inputs) xử lý chúng và cho ra một kết quả (Output) duy nhất.

3.6.2. Quá trình xử lý thông tin của một ANN



Hình 3.22. Quá trình xử lý thông tin của một ANN

Inputs: Mỗi Input tương ứng với 1 thuộc tính (attribute) của dữ liệu (patterns).

Output: Kết quả của một ANN là một giải pháp cho một vấn đề.

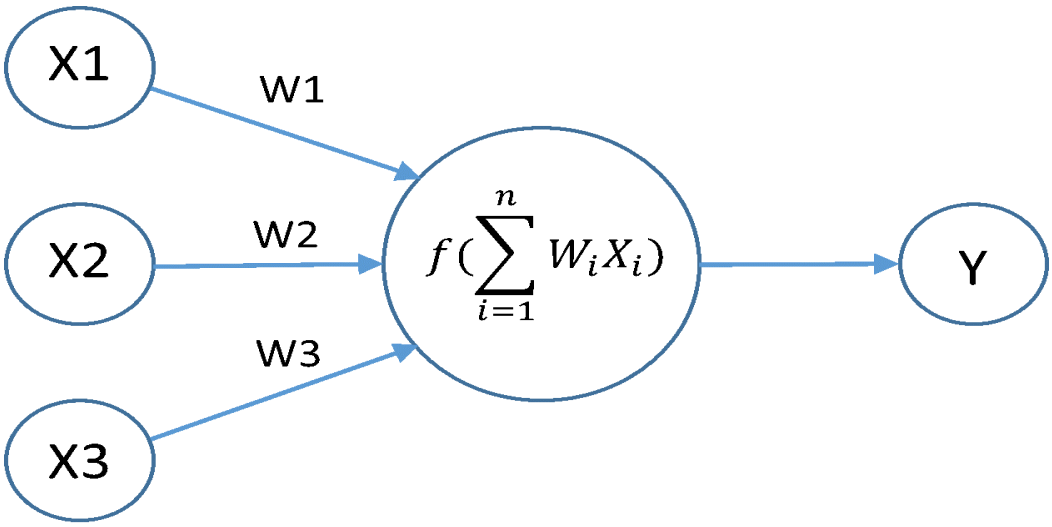
Connection Weights (Trọng số liên kết): Đây là thành phần rất quan trọng của một ANN, nó thể hiện mức độ quan trọng (độ mạnh) của dữ liệu đầu vào đối với quá trình xử lý thông tin (quá trình chuyển đổi dữ liệu từ Layer này sang layer khác). Quá trình học (Learning Processing) của ANN thực ra là quá trình điều chỉnh các trọng số (Weight) của các input data để có được kết quả mong muốn.

Summation Function (Hàm tổng): Tính tổng trọng số của tất cả các input được đưa vào mỗi Neuron. Hàm tổng của một Neuron đối với **n** input được tính theo công thức sau:

$$Y = \sum_{i=1}^{n} X_i W_i$$

Trong đó:

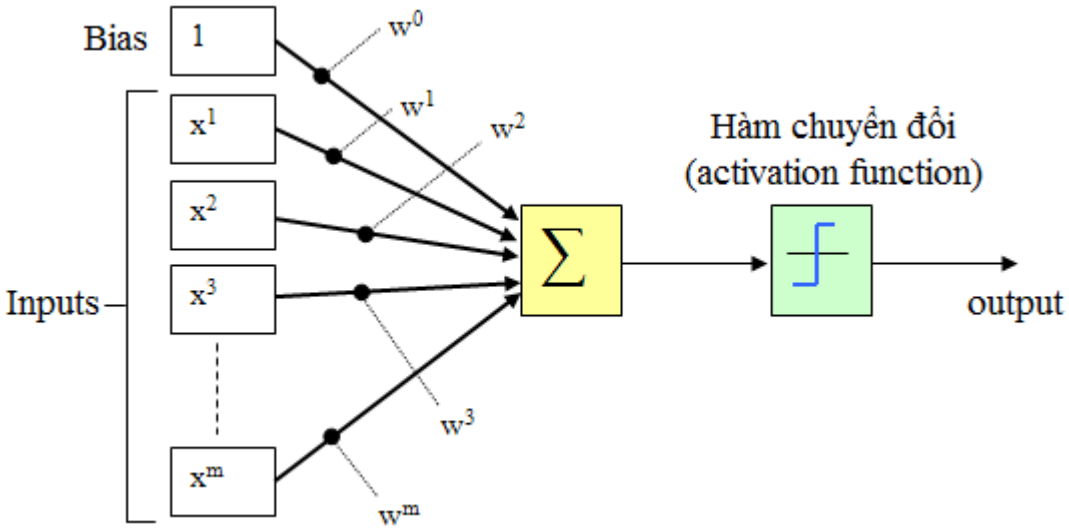
- X_i là giá trị đầu vào;
- W_i là trọng số liên kết của mỗi giá trị đầu vào tương ứng;



Hình 3.23. Hàm tổng của một neuron

3.6.3. Transfer function

Hàm tổng (Summing Function) của một Neuron cho biết khả năng kích hoạt (Activation) của neuron đó còn gọi là internal activation. Các Neuron này có thể sinh ra một output hoặc không trong ANN (nói cách khác rằng có thể output của 1 Neuron có thể được chuyển đến layer tiếp trong mạng Neuron theo hoặc không). Mối quan hệ giữa Internal Activation và output được thể hiện bằng hàm chuyển đổi (Transfer Function).



Hình 3.24. Hàm chuyển đổi giữa internal activation và output

Việc lựa chọn Transfer Function có tác động lớn đến kết quả của ANN. Hàm chuyển đổi phi tuyến được sử dụng phổ biến trong ANN là sigmoid (logical activation) function.

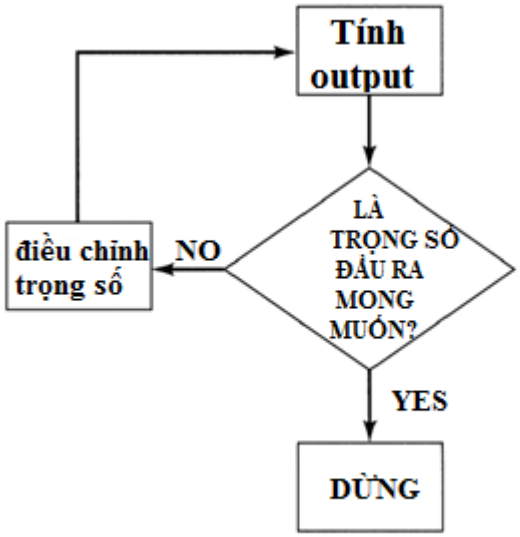
$$Y_T = \frac{1}{1 + e^{-Y}}$$

Trong đó:

- $\mathbf{Y_T}$ là hàm chuyển đổi;
- **Y** là hàm tổng;

Kết quả xử lý tại các Neuron (Output) đôi khi rất lớn, vì vậy transfer function được sử dụng để xử lý output này trước khi chuyển đến layer tiếp theo. Đôi khi thay vì sử dụng Transfer Function người ta sử dụng giá trị ngưỡng (Threshold value) để kiểm soát các output của các neuron tại một layer nào đó trước khi chuyển các output này đến các Layer tiếp theo. Quá trình học (Learning Processing) của ANN

Quá trình Training được lặp lại cho đến kết quả (output) của ANN đạt được giá trị mong muốn (Desired value) đã biết. Điển hình cho kỹ thuật này là mạng Neuron lan truyền ngược (Backpropagation).



Hình 3.25. Quá trình học của ANN

Quá trình training được diễn ra như sau:

Bước 1: Tính giá trị output .

Bước 2: So sánh output với giá trị mong muốn (desired value).

Bước 3: Nếu chưa đạt được giá trị mong muốn thì hiệu chỉnh trọng số (weights) và tính lại output

3.6.4. Nguyên tắc huấn luyện (Training Protocols)

Mạng Neuron có 3 cách huấn luyện chính đó là batch training, stochastic training và on-line training. Đối với on-line training thì các trọng số của mạng (weights) được cập nhật ngay lập tức sau khi một input pattern được đưa vào mạng. Stohastic training cũng giống như on-line training nhưng việc chọn các input patterns để đưa vào mạng từ training set được thực hiện ngẫu nhiên (random). Batch training thì tất cả các input patterns được đưa vào mạng cùng lúc và sau đó cập nhật các trọng số mạng đồng thời. Ưu điểm của on-line training là tiết kiệm bộ nhớ vì không cần lưu lại số lượng lớn các input patterns trong bộ nhớ.

Trong quá trình huấn luyện mạng, thuật ngữ “epoch” được dùng để mô tả quá trình khi tất cả các input patterns của training set được đưa để huấn luyện mạng. Nói cách khách 1 epoch được hoàn thành khi tất cả các dữ liệu trong training set được đưa vào huấn luyện mạng. Vì vậy số lượng “epoch” xác định số lần mạng được huấn luyện (hay số lần đưa tất cả các dữ liệu trong training set vào mạng).

Chương 4
ĐÁNH GIÁ MÔ HÌNH (MODEL EVALUATION)

4.1. Giới Thiệu

Khi xây dựng một mô hình Machine Learning, Ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình. Có rất nhiều cách đánh giá một mô hình phân lớp. Tuỳ vào những bài toán khác nhau mà ta sử dụng các phương pháp khác nhau. Các phương pháp thường được sử dụng là: Accuracy score, Confusion matrix, ROC curve, Area Under the Curve, Precision and Recall, F1 score.

4.2. Accuracy

Cách đơn giản và hay được sử dụng nhất là accuracy (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

4.3. Confusion Matrix (CM)

Cách tính sử dụng Accuracy chỉ cho ta biết được bao nhiêu phần trăm lượng dữ liệu được phân loại đúng mà không chỉ ra được cụ thể mỗi loại được phân loại như thế nào, lớp nào được phân loại đúng nhiều nhất, và dữ liệu thuộc lớp nào thường bị phân loại nhầm vào lớp khác. Để có thể đánh giá được các giá trị này, ta sử dụng một ma trận được gọi là confusion matrix.

Về cơ bản, confusion matrix thể hiện có bao nhiêu điểm dữ liệu thực sự thuộc vào một class, và được dự đoán là rơi vào một class. Để hiểu rõ hơn, hãy xem bảng dưới đây:

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Bảng 4.1. Confusion Matrix

Có tổng cộng 165 điểm dữ liệu. Ta xét ma trận tạo bởi các giá trị tại vùng 2x2 trung tâm của bảng. Ma trận thu được được gọi là Confusion Matrix. Nó là một ma trận vuông với kích thước mỗi chiều bằng số lượng lớp dữ liệu. Giá trị tại hàng thứ i, cột thứ j là số lượng điểm lẽ ra thuộc vào class i nhưng lại được dự đoán là thuộc vào class j. Như vậy, nhìn vào hàng thứ nhất (0), ta có thể thấy được rằng trong số 60 điểm thực sự thuộc lớp 0, chỉ có một điểm được phân loại đúng, điểm còn lại bị phân loại nhầm vào lớp 1.

Ta có thể suy ra ngay rằng tổng các phần tử trong toàn ma trận này chính là số điểm trong tập kiểm thử. Các phần tử trên đường chéo của ma trận là số điểm được phân loại đúng của mỗi lớp dữ liệu. Từ đây có thể suy ra accuracy chính bằng tổng các phần tử trên đường chéo chia cho tổng các phần tử của toàn ma trận. Đoạn code dưới đây mô tả cách tính confusion matrix:

```
def confusion_matrix(y_true,y_pred): # xây dựng hàm tính ma trận
    N= np.unique(y_true).shape[0]  #tính số lớp của ma trận
    cm = np.zeros((N,N)) #khởi tạo ma trận 0 NxN lớp
    for i in range(y_true.shape[0]):#lấy từng phần tử trong mtran
        cm[y_true[i],y_pred[i]] +=1
    return cm
y_true = np.array([0, 0, 0, 0, 1, 1, 1, 2, 2, 2])
y_pred = np.array([0, 1, 0, 2, 1, 1, 0, 2, 1, 2])
cnf_matrix = confusion_matrix(y_true, y_pred) #gọi hàm
print('Confusion matrix là :',cnf_matrix)
print('\nĐộ chính xác Accuracy:
{}%'.format(int(np.diagonal(cnf_matrix).sum()/cnf_matrix.sum()*10
0)))
```

Kết quả:

```
Confusion matrix là:
[[ 2.  1.  1.]
 [ 1.  2.  0.]
 [ 0.  1.  2.]]
Độ chính xác Accuracy: 60%
```

Cách biểu diễn trên đây của confusion matrix còn được gọi là unnormalized confusion matrix, tức CM chưa chuẩn hoá. Để có cái nhìn rõ hơn, ta có thể dùng normalized confuion matrix, tức CM được chuẩn hoá. Để có normalized confusion matrix, ta lấy mỗi hàng của unnormalized confusion matrix sẽ được chia cho tổng các phần tử trên hàng đó. Như vậy, ta có nhận xét rằng tổng các phần tử trên một hàng của normalized confusion matrix luôn bằng 1. Điều này thường không đúng trên mỗi cột. Dưới đây là cách tính normalized confusion matrix:

```
normalized_confusion_matrix =
cnf_matrix/cnf_matrix.sum(axis=1,keepdims=True)
print("Normalized confusion matrix là:\n
{}".format(normalized_confusion_matrix))
```

Kết quả:

```
Normalized confusion matrix là:
 [[ 0.5         0.25        0.25      ]
 [ 0.33333333  0.66666667  0.        ]
 [ 0.          0.33333333  0.66666667]]
```

4.4. True/False Positive/Negative

Cách đánh giá này thường được áp dụng cho các bài toán phân lớp có hai lớp dữ liệu. Cụ thể hơn, trong hai lớp dữ liệu này có một lớp nghiêm trọng hơn lớp kia và cần được dự đoán chính xác. Ví dụ: trong bài toán xác định có bệnh ung thư hay không thì việc không bị sót (miss) quan trọng hơn là việc chẩn đoán nhầm âm tính thành dương tính. Trong bài toán xác định có mìn dưới lòng đất hay không thì việc bỏ sót nghiêm trọng hơn việc báo động nhầm rất nhiều. Hay trong

bài toán lọc email rác thì việc cho nhầm email quan trọng vào thùng rác nghiêm trọng hơn việc xác định một email rác là email thường.

Trong những bài toán này, người ta thường định nghĩa lớp dữ liệu quan trọng hơn cần được xác định đúng là lớp Positive (P-dương tính), lớp còn lại được gọi là Negative (N-âm tính). Ta định nghĩa True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) dựa trên confusion matrix chưa chuẩn hoá như sau:

	Predicted as Positive	**Predicted as Negative**
Actual: Positive	True Positive (TP)	False Negative (FN)
Actual: Negative	False Positive (FP)	True Negative (TN)

Bảng 4.2. True/False Positive/Negative

Người ta thường quan tâm đến **TPR, FNR, FPR, TNR** (R - Rate) dựa trên normalized confusion matrix như sau:

	Predicted as Positive	**Predicted as Negative**
Actual: Positive	TPR = TP/(TP + FN)	FNR = FN/(TP + FN)
Actual: Negative	FPR = FP/(FP + TN)	TNR = TN/(FP + TN)

Bảng 4.3. Normalized Confusion Matrix

False Positive Rate còn được gọi là **False Alarm Rate** (tỉ lệ báo động nhầm), **False Negative Rate** còn được gọi là **Miss Detection Rate** (tỉ lệ bỏ sót). Trong bài toán dò mìn, thà báo nhầm còn hơn bỏ sót, tức là ta có thể chấp nhận **False Alarm Rate** cao để đạt được **Miss Detection Rate** thấp.

Chú ý:

- Việc biết một cột của **confusion matrix** này sẽ suy ra được cột còn lại vì tổng các hàng luôn bằng 1 và chỉ có hai lớp dữ liệu.
- Với các bài toán có nhiều lớp dữ liệu, ta có thể xây dựng bảng **True/False Positive/Negative** Negative.cho mỗi lớp nếu coi lớp đó là lớp Positive, các lớp còn lại gộp chung thành lớp

4.5. Receiver operating characteristic curve (ROC)

Nếu bây giờ ta coi lớp **1** là lớp Positive, lớp **0** là lớp Negative, làm thế nào để tăng mức độ báo nhầm (FPR) để giảm mức độ bỏ sót (FNR). Chú ý rằng tăng FNR đồng nghĩa với việc giảm TPR vì tổng của chúng luôn bằng **1**.

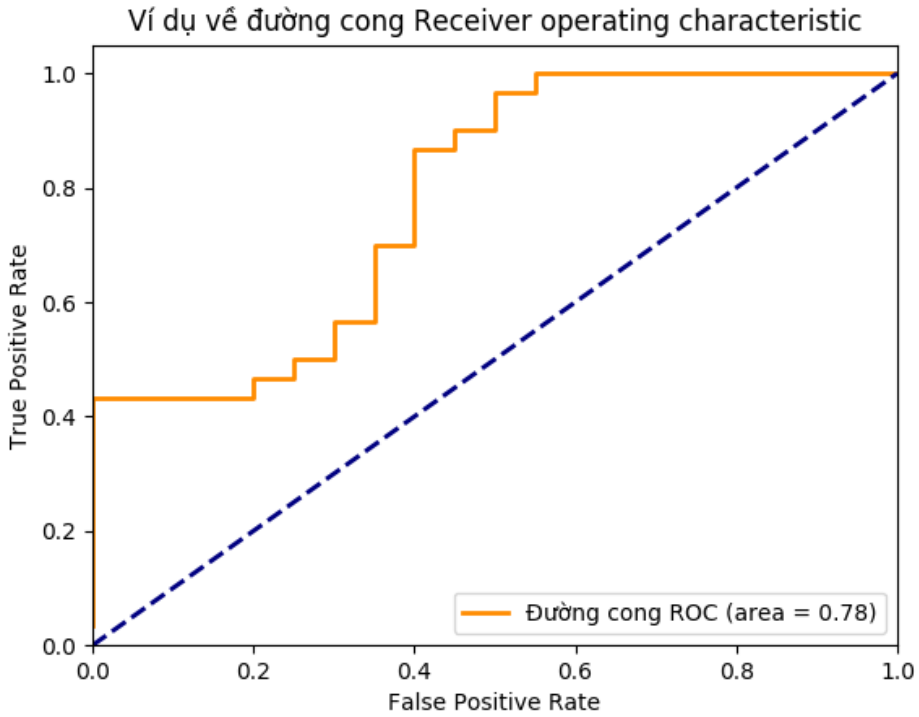
Một kỹ thuật đơn giản là ta thay giá trị threshold từ **0.5** xuống một số nhỏ hơn. Chẳng hạn nếu chọn threshold = **0.3**, thì mọi điểm được dự đoán có xác suất đầu ra lớn hơn **0.3** sẽ được dự đoán là thuộc lớp Positive. Nói cách khác, tỉ lệ các điểm được phân loại là Positive sẽ tăng lên, kéo theo cả False Positive Rate và True Positive Rate cùng tăng lên (cột thứ nhất trong ma trận tăng lên). Từ đây suy ra cả FNR và TNR đều giảm.

Ngược lại, nếu ta muốn bỏ sót còn hơn báo nhầm, tất nhiên là ở mức độ nào đó, như bài toán xác định email rác chẳng hạn, ta cần tăng threshold lên một số lớn hơn 0.5. Khi đó, hầu hết các điểm dữ liệu sẽ được dự đoán thuộc lớp 0, tức Negative, và cả TNF và FNR đều tăng lên, tức TPR và FPR giảm xuống.

Như vậy, ứng với mỗi giá trị của threshold, ta sẽ thu được một cặp (FPR, TPR). Biểu diễn các điểm (FPR, TPR) trên đồ thị khi thay đổi threshold từ 0 tới 1 ta sẽ thu được một đường được gọi là Receiver Operating Characteristic curve hay ROC curve.

Chú ý rằng khoảng giá trị của threshold không nhất thiết từ **0** tới **1** trong các bài toán tổng quát. Khoảng giá trị này cần được đảm bảo có trường hợp TPR/FPR nhận giá trị lớn nhất hay nhỏ nhất mà nó có thể đạt được.

Dưới đây là hình ảnh mô tả đường cong Receiver Operating Characteristic:



Hình 4.1. Ví dụ về đường ROC

4.6. Area Under the Curve

Dựa trên đường cong ROC, ta có thể chỉ ra rằng một mô hình có hiệu quả hay không. Một mô hình hiệu quả khi có FPR thấp và TPR cao, tức tồn tại một điểm trên đường cong ROC gần với điểm có toạ độ **(0, 1)** trên đồ thị (góc trên bên trái). Curve (đường cong) càng gần thì mô hình càng hiệu quả.

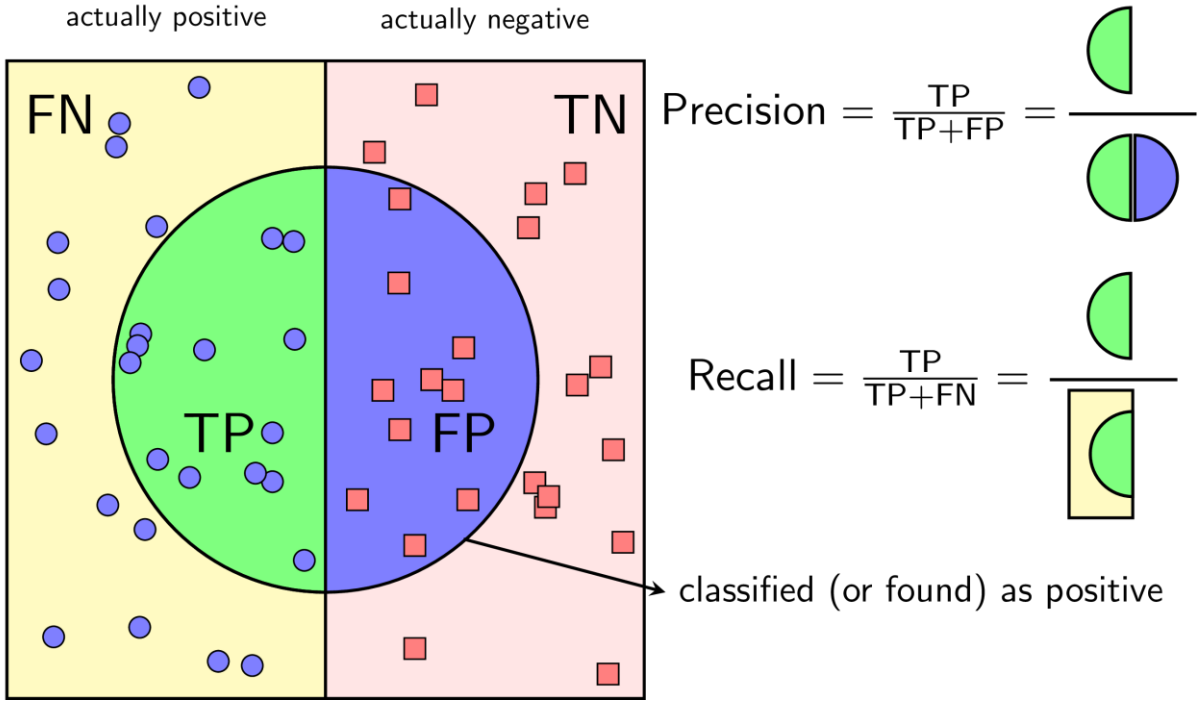
Có một thông số nữa dùng để đánh giá được gọi là Area Under the Curve hay AUC. Đại lượng này chính là diện tích nằm dưới ROC curve màu cam. Giá trị này là một số dương nhỏ hơn hoặc bằng 1. Giá trị này càng lớn thì mô hình càng tốt.

4.7. Precision và Recall

Với bài toán phân loại mà tập dữ liệu của các lớp là chênh lệch nhau rất nhiều, có một phép đó hiệu quả thường được sử dụng là Precision-Recall.

Trước hết xét bài toán phân loại nhị phân. Ta cũng coi một trong hai lớp là positive, lớp còn lại là negative.

Xét hình 3 dưới đây:



Hình 4.2. Cách tính Precision và Recall

Với một cách xác định một lớp là positive, Precision được định nghĩa là tỉ lệ số điểm true positive trong số những điểm được phân loại là positive (TP + FP).

Recall được định nghĩa là tỉ lệ số điểm true positive trong số những điểm thực sự là positive (TP + FN).

Một cách toán học, Precison và Recall là hai phân số có tử số bằng nhau nhưng mẫu số khác nhau:

$$\boldsymbol{Precision = \frac{TP}{TP + FP}}$$

$$\boldsymbol{Recall = \frac{TP}{TP + FN}}$$

Có thể nhận thấy rằng TPR và Recall là hai đại lượng bằng nhau. Ngoài ra, cả Precision và Recall đều là các số không âm nhỏ hơn hoặc bằng một.

Precision cao đồng nghĩa với việc độ chính xác của các điểm tìm được là cao. Recall cao đồng nghĩa với việc True Positive Rate cao, tức tỉ lệ bỏ sót các điểm thực sự positive là thấp.

4.8. Residual sum of squares (RSS)

Trong thống kê, residual sum of squares (RSS) hoặc the sum of squared residuals (SSR) hoặc the sum of squared errors of prediction (SSE), là tổng số dư bình phương (độ lệch giữa giá trị dự đoán và giá trị thực tế của dữ liệu). Đây là thước đo sự khác biệt giữa dữ liệu thực và mô hình ước tính. Nếu giá trị RSS nhỏ thì có nghĩa là độ **fit** (khít) giữa mô hình dự đoán và mô hình ước tính cao. Nó được dùng như một tiêu chí tối ưu trong việc lựa chọn tham số và lựa chọn mô hình.

$$\boldsymbol{RSS} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{y_i} - \boldsymbol{f(x_i)})^{\boldsymbol{2}}$$

Trong đó:

- $\boldsymbol{y_i}$ là giá trị của biến thứ i được dự đoán;
- $\boldsymbol{x_i}$ là giá trị thứ i của biến giải thích;
- $\boldsymbol{f(x_i)}$ là giá trị được dự đoán của $\boldsymbol{y_i}$;

KẾT LUẬN

Sau khi đã áp dụng quy trình khai phá dữ liệu sử dụng Machine Learning vào dữ liệu thực tiễn với các phần đã hoàn thành như:

- Ứng dụng quy trình khai phá dữ liệu và rút trích các tri thức giá trị của dữ liệu đối với 3 dạng bài toán: Regression, Classification và Clustering.
- Thực hiện Model Tuning để cải thiện độ chính xác mô hình.
- Thực hiện đánh giá mô hình qua nhiều phương pháp.
- Xây dựng Data Story một cách mạch lạc và trực quan.

Trong quá trình thực hiện, đã có một số nhận xét chung về ưu và nhược điểm về mô hình Machine Learning như sau:

	Ưu điểm	**Nhược điểm**
Bài toán Regression	Cho kết quả dự đoán cao	Rất nhạy cảm với nhiễu, dễ bị Overfitting
Bài toán Classification	Cho kết quả phân loại tốt	Thời gian huấn luyện lâu (SVM)
Bài toán Clustering	Thời gian huấn luyện nhanh	Khó khăn trong việc Feature Selection

Dựa vào việc đánh giá ưu và nhược điểm, ta có thể sử dụng phương hướng phát triển mới như:

Sử dụng các phương pháp và mô hình Machine Learning mới để đem lại độ chính xác cao hơn. Ví dụ như các thuật toán Ensemble như Boosting, AdaBoost, Random Forest. Đổi lại, mô hình đòi hỏi phần cứng hệ thống cao hơn, thời gian train lâu hơn.

TÀI LIỆU THAM KHẢO

1. Ông Xuân Hồng, *Blog Machine Learning Ông Xuân Hồng.*

2. Vũ Hữu Tiệp, *Blog Machine Learning Cơ Bản.*

3. Andrew Ng (2014), *Machine Learning Course*, khóa học online trả phí.

4. Andrew Ng (2018), *Deep Learning Course*, khóa học online trả phí.

5. Christopher Bishop (2006), *Pattern Recognition and Machine Learning*, Sách điện tử.

6. Grant Sanderson (2012), *3Blue1Brown Course*, khóa học online miễn phí trên Youtube.

7. Jason Brownlee (2016), *Machine Learning Mastery*, sách điện tử.

8. Kirill Eremenko, *Machine Learning AZ™: Hands-On Python & R In Data Science – Udemy, Super Data Science Podcast.*

PHỤ LỤC CÁC THUẬT NGỮ

Thuật ngữ	Mô tả
Boosting	Là thuật toán học quần thể bằng cách xây dựng nhiều thuật toán học cùng lúc (ví dụ như cây quyết định) và kết hợp chúng lại.
Data Story	Là cách mà dẫn dắt người đọc quá trình rút trích insight từ bước ban đầu như cách tiếp cận,khám phá data cho đến các kĩ thuật xử lý data, đưa vào model.
Ensemble	Phương pháp ensemble là mô hình được tổng hợp từ nhiều mô hình con (weaker model) được huấn luyện độc lập. Kết quả dự đoán cuối cùng dựa trên kết quả "bỏ phiếu" của từng mô hình con đó cho kết quả đầu ra.
Euclid	Đo khoảng cách giữa hai điểm trong không gian.
Insight	Trong marketing, Customer Insight là những suy nghĩ, mong muốn ẩn sâu bên trong ảnh hưởng đến quyết định mua hàng của Customers/Consumers.
Linearly separable	Tách biệt tuyến tính, nếu tồn tại một đường phẳng phân chia hai class thì ta gọi hai class đó là linearly separable.
Marketer	Những người làm công việc marketing trong doanh nghiệp được gọi là Marketer.
Machine Learning	Machine Learning (máy học) 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.
Model Tuning	Là tinh chỉnh mô hình, hiểu được các thông số (Parameter) của mô hình và tinh chỉnh các thông số này ra sao sẽ làm thay đổi đáng kể độ chính xác của mô hình dự đoán. Kĩ năng này đòi hỏi phải có hiểu biết thấu đáo về các thuật toán machine learning.
Norm	Để xác định khoảng cách giữa hai vector $\boldsymbol{y}$ và $\boldsymbol{z}$, người ta thường áp dụng một hàm số lên vector hiệu $\boldsymbol{x} = \boldsymbol{y} - \boldsymbol{z}$.
Training	Quá trình huấn luyện mô hình Machine Learning.

TRƯỜNG ĐH THÔNG TIN LIÊN LẠC **CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**
KHOA ..... **Độc lập - Tự do - Hạnh phúc**

PHIẾU NHẬN XÉT KHÓA LUẬN TỐT NGHIỆP

(Dành cho giảng viên hướng dẫn)

I. Thông tin chung

- Họ và tên sinh viên: ..... MSSV: .....
- Lớp: ……..…
- Tên đề tài: …….
- Họ và tên người hướng dẫn: …….

II. Nhận xét về khóa luận

2.1. Nhận xét về hình thức: ……

…….

…….

2.2. Tính cấp thiết của đề tài: …….

…….

…….

2.3. Mục tiêu và nội dung: ……..

…….

…….

…….

2.4. Tổng quan tài liệu và tài liệu tham khảo: ……

…….

…….

2.5. Phương pháp nghiên cứu: ……

…….

…….

…….

2.6. Kết quả đạt được:…….

…….

…….

…….

2.7. Kết luận và đề nghị: …….

…….

…….

…….

2.8. Tính sáng tạo và ứng dụng:……..

…….

……

2.9. Các vấn đề cần bổ sung, chỉnh sửa: …….

…….

……

III. Phần nhận xét tinh thần và thái độ làm việc của sinh viên

…….

……

IV. Đánh giá

1. Đánh giá chung

2. Đề nghị Được bảo vệ: ☐

Không được bảo vệ: ☐

Cán bộ hướng dẫn
(Ký, ghi rõ họ tên)