**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.

Trường Đại Học Thông Tin Liên Lạc

Tháng 06 năm 2018

Sinh Viên Thực Hiện

**Trịnh Đình Phúc**

DANH MỤC BẢNG BIỂU VÀ HÌNH VẼ

Bảng

Hình ảnh

[Hình 2.1: Một nút cổ chai có thể xảy ra khi có quá nhiều tranh chấp cho tài nguyên máy chủ nội bộ 5](#_Toc516701537)

[Hình 2.2: Cấu trúc peer-to-peer của Cassandra 6](#_Toc516701538)

[Hình 2.3: Phân tán dữ liệu trên cassandra 7](#_Toc516701539)

[Hình 2.4: Cụm Cassandra với 4 nút được phân bố trên ring 8](#_Toc516701540)

[Hình 2.5: Nút ảo 10](#_Toc516701541)

[Hình 2.6: Việc giao tiếp giữa các nút trong cassandra 11](#_Toc516701542)

[Hình 2.7: Mô hình dữ liệu của cassandra 13](#_Toc516701543)

[Hình 2.8: Quy trình CRISP-DM 14](#_Toc516701544)

[Hình 2.9: Mô hình chung cho các bài toán Machine Learning 18](#_Toc516701545)

[Hình 2.10: Sử dụng phương pháp PCA để giảm chiều dữ liệu 3D xuống 2D 22](#_Toc516701546)

[Hình 3.1: Hiển thị dữ liệu về chiều cao & cân nặng trên đồ thị 32](#_Toc516701547)

[Hình 3.2: Mô hình Linear Regression sau khi học xong 33](#_Toc516701548)

[Hình 3.3: Ảnh hưởng của Outliers 34](#_Toc516701549)

[Hình 3.4: Biểu diễn các điểm local và global 35](#_Toc516701550)

[Hình 3.5: Parapol của hàm f(x) 36](#_Toc516701551)

[Hình 3.6: Minh họa thuật toán Gradient Decent – Bước 1 38](#_Toc516701552)

[Hình 3.7: Minh họa thuật toán Gradient Decent – Bước 2 39](#_Toc516701553)

[Hình 3.8: Minh họa thuật toán Gradient Decent – Bước 3 39](#_Toc516701554)

[Hình 3.9: Minh họa thuật toán Gradient Decent – Bước 4 39](#_Toc516701555)

[40](#_Toc516701556)

[Hình 3.10: Minh họa thuật toán Gradient Decent – Bước 5 40](#_Toc516701557)

[Hình 3.11: Bài toán phân cụm với 3 cluster 43](#_Toc516701558)

[Hình 3.12: Kết quả thi dựa trên số giờ học 48](#_Toc516701559)

[Hình 3.13: Một số loại Activation Function 49](#_Toc516701560)

[Hình 3.14: Mô tả Linear Regression không phù hợp 49](#_Toc516701561)

[Hình 3.15: Kiến trúc tổng quát của một mạng Neuron nhân tạo 51](#_Toc516701562)

[Hình 3.16: Quá trình xử lý thông tin của một ANN 52](#_Toc516701563)

[Hình 3.17: Hàm tổng của một neuron 52](#_Toc516701564)

[Hình 3.18: Hàm tổng của nhiều neuron 53](#_Toc516701565)

[Hình 3.19: Hàm chuyển đổi giữa internal activation và output 53](#_Toc516701566)

[Hình 3.20: Quá trình học của ANN 55](#_Toc516701567)

[Hình 4.1: Ví dụ về đường ROC 61](#_Toc516701568)

[Hình 4.2: Cách tính Precision và Recall 62](#_Toc516701569)

BẢNG CHỮ VIẾT TẮT

MỤC LỤC

[MỞ ĐẦU 1](#_Toc516699494)

[Chương 1: TỔNG QUAN VỀ PHƯƠNG PHÁP XỬ LÝ DỮ LIỆU LỚN 2](#_Toc516699495)

[1.1. Tính thực tiễn 2](#_Toc516699496)

[1.2. Mục đích đề tài 3](#_Toc516699497)

[1.3. Phạm vi đề tài 3](#_Toc516699498)

[1.4. Trọng tâm đề tài 4](#_Toc516699499)

[1.5. Phương pháp nghiên cứu 4](#_Toc516699500)

[Chương 2: CỞ SỞ LÝ THUYẾT 5](#_Toc516699501)

[2.1. Apache Cassandra 5](#_Toc516699502)

[2.1.1. Cấu trúc của Cassandra 6](#_Toc516699503)

[2.1.2. Phân tán dữ liệu trong Cassandra 7](#_Toc516699504)

[2.1.3. Nhân bản dữ liệu trong Cassandra 10](#_Toc516699505)

[2.1.4. Giao tiếp giữa các nút trong Cassandra 11](#_Toc516699506)

[2.1.5. Mô hình dữ liệu của Cassandras 13](#_Toc516699507)

[2.2. Quy trình khai phá dữ liệu 14](#_Toc516699508)

[2.2.1. Tìm hiểu nghiệp vụ (Business understanding) 14](#_Toc516699509)

[2.2.2. Tìm hiểu dữ liệu (Data understanding) 14](#_Toc516699510)

[2.2.3. Chuẩn bị dữ liệu (Data preparation) 14](#_Toc516699511)

[2.2.4. Mô hình hoá (Modeling) 15](#_Toc516699512)

[2.2.5. Đánh giá mô hình (Evaluation Model) 15](#_Toc516699513)

[2.2.6. Triển khai (Deployment) 15](#_Toc516699514)

[2.3. Các phương pháp khai thác dữ liệu 16](#_Toc516699518)

[2.3.1. Thống kê mô tả (Descriptive Statistics) 16](#_Toc516699519)

[2.3.2. Thống kê suy luận (Inferential Statistics) 16](#_Toc516699520)

[2.3.3. Phân tích dữ liệu thăm dò (Exploratory Data Analysis - EDA) 16](#_Toc516699521)

[2.3.4. Phân tích dự đoán (Predictive Analysis) 17](#_Toc516699522)

[2.3.5. Phân tích nguyên nhân (Causal analysis) 17](#_Toc516699523)

[2.3.6. Phân tích cơ chế (Mechanistic analysis) 17](#_Toc516699524)

[2.4. Mô hình chung cho các bài toán Machine Learning 18](#_Toc516699525)

[2.4.1. Training phase 18](#_Toc516699526)

[2.4.2. Testing phase 20](#_Toc516699527)

[2.5. Feature Engineering 20](#_Toc516699528)

[2.5.1. Feature Selection 20](#_Toc516699529)

[2.5.2. Bag-of-Word 23](#_Toc516699530)

[2.5.3. Feature Scaling & Normalization 25](#_Toc516699531)

[Chương 3: MỘT SỐ MÔ HÌNH MACHINE LEARNING 28](#_Toc516699532)

[3.1. Linear Regression/Ordinary least squares 28](#_Toc516699534)

[3.1.1. Giới thiệu 28](#_Toc516699535)

[3.1.2. Dạng của Linear Regression 29](#_Toc516699536)

[3.1.3. Sai số dự đoán 29](#_Toc516699537)

[3.1.4. Hàm mất mát (Cost Function /Lost Function) 29](#_Toc516699538)

[3.1.5. Nghiệm trên bài toán Linear Regression 30](#_Toc516699539)

[3.1.6. Ví dụ trên Python 31](#_Toc516699540)

[3.1.7. Hạn chế của Linear Regression 34](#_Toc516699541)

[3.2. Gradient Decent 35](#_Toc516699542)

[3.2.1. Gradient Decent cho hàm một biến 36](#_Toc516699543)

[3.2.2. Gradient Decent cho hàm nhiều biến 39](#_Toc516699544)

[3.2.3. Stopping Criteria (điều kiện dừng) 39](#_Toc516699545)

[3.2.4. Biến thể của Gradient Descent 40](#_Toc516699546)

[3.2.5. Thứ tự lựa chọn điểm dữ liệu 41](#_Toc516699547)

[3.3. K-Means 41](#_Toc516699548)

[3.3.1. Giới thiệu 41](#_Toc516699549)

[3.3.2. Ví dụ 41](#_Toc516699550)

[3.3.3. Phân tích mặt toán học 42](#_Toc516699551)

[3.3.4. Thuật toán 43](#_Toc516699552)

[3.3.5. Hàm mất mát và bài toán tối ưu 43](#_Toc516699553)

[3.3.6. Thuật toán tối ưu hàm mất mát 44](#_Toc516699554)

[3.4. Logistic Regression 45](#_Toc516699555)

[3.4.1. Giới thiệu 45](#_Toc516699556)

[3.4.2. Ví dụ 46](#_Toc516699557)

[3.4.3. Mô hình Logistic Regression 47](#_Toc516699558)

[3.4.4. Sigmoid Function 48](#_Toc516699559)

[3.4.5. Tanh Function (Hyperpolic Tangent) 48](#_Toc516699560)

[3.5. Artificial Neural Network (ANN) 48](#_Toc516699561)

[3.5.1. Kiến trúc tổng quát của một ANN 48](#_Toc516699562)

[3.5.2. Quá trình xử lý thông tin của một ANN 49](#_Toc516699563)

[3.5.3. Transfer function 51](#_Toc516699564)

[3.5.4. Quá trình học (Learning Processing) của ANN 52](#_Toc516699565)

[3.5.5. Nguyên tắc huấn luyện (Training Protocols) 53](#_Toc516699566)

[3.6. Support Vector Machines 53](#_Toc516699567)

[Chương 4: ĐÁNH GIÁ MÔ HÌNH (MODEL EVALUATION) 54](#_Toc516699568)

[4.1. Giới Thiệu 54](#_Toc516699569)

[4.2. Accuracy 54](#_Toc516699570)

[4.3. Confusion Matrix (CM) 54](#_Toc516699571)

[4.4. True/False Positive/Negative 56](#_Toc516699572)

[4.5. Receiver operating characteristic curve 58](#_Toc516699573)

[4.6. Area Under the Curve 59](#_Toc516699574)

[4.7. Precision và Recall 60](#_Toc516699575)

[4.8. Residual sum of squares (RSS) 61](#_Toc516699576)

[KẾT LUẬN 62](#_Toc516699578)

MỞ ĐẦU

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

* 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 tiềm ẩn phục vụ cho nhu cầu sử dụng của con người là vô cùng cấp thiết.

* **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 có thể giải quyết được các nhu cầu trên.

* 1. Mục đích đề tài

Việc nghiên cứu đề tài “Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn” trước hết là 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 như 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 sẽ là tạo ra một báo cáo khoa học về quy trình dụng phương pháp xử lý dữ liệu lớn.

* 1. 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).
* Hệ cơ sở dữ liệu Cassandra.
* Quy trình khai phá 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. 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.

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.

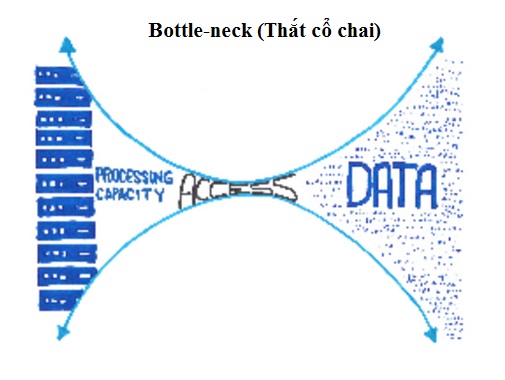
Đối với việc xử lý thông tin:

* Nhận tất cả các thông tin liên quan nhưng chỉ xử lý và chắc 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.

Chương 2:  
CỞ SỞ LÝ THUYẾT

* 1. Apache Cassandra

**Cassandra là một cơ sở dữ liệu hướng cột, phân tán mã nguồn mở được thiết kế để xử lý một khối lượng lớn dữ liệu giàn trải trên nhiều node mà vẫn đảm bảo tính sẵn sàng cao (Highly Availability), khả năng mở rộng hay thu giảm số node linh hoạt (Elastic Scalability) và chấp nhận một số lỗi (Fault Tolerant). Nó được phát triển bởi Facebook và vẫn còn tiếp tục phát triển và sử dụng cho mạng xã hội lớn nhất thới giới này. Cassandra được coi là sự kết hợp của Dynamo của Amazon và BigTable của Google.**

**Các nút máy chủ trong cụm Cassandra là đồng nhất theo thiết kế ngang hàng (peer-to-peer), không có bất cứ thành phần nào trong hệ thống là điểm hỏng thắt cổ chai (bottle-neck).**

Hình 2.1: Một nút cổ chai có thể xảy ra khi có quá nhiều tranh chấp cho tài nguyên máy chủ nội bộ

* + 1. **Cấu trúc của Cassandra**

Hình 2.2: Cấu trúc peer-to-peer của Cassandra

Thiết kế của Cassandra là thiết kế phân tán trên hàng ngàn máy chủ mà không có bất cứ điểm chết tập trung nào. Cassandra có thiết kế dựa trên kiến trúc mạng ngang hàng (Peer-to-Peer) tất cả các nút máy chủ trong hệ thống đều có vai trò như nhau và không có nút máy chủ nào đóng vai trò là máy chủ trung tâm mà việc hỏng hóc của máy chủ này có thể kéo theo đánh sập hoàn toàn hệ thống như các kiến trúc chủ - khách truyền thống. Các nút máy chủ của Cassandra là độc lập và tham gia vào kết nối với các nút máy chủ khác trong hệ thống. Mỗi nút đều có thể xử lý các thao tác ghi và đọc dữ liệu, không phân biệt là dữ liệu được lưu trữ một cách vật lý trên máy chủ nào trong hệ thống. Khi một nút trong hệ thống bị hỏng hóc và dừng hoạt động, các thao tác đọc ghi dữ liệu có thể được xử lý bởi các nút khác trong hệ thống. Quá trình này hoàn toàn trong suốt với ứng dụng cho phép ẩn đi hỏng hóc của hệ thống đối với các ứng dụng đó. Trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Cơ chế này được thực thi trong quá trình đọc dữ liệu (read repair) thay vì đồng bộ ngay trong thao tác ghi dữ liệu, điều này cho phép tăng hiệu năng cho thao tác ghi dữ liệu.

* + 1. Phân tán dữ liệu trong Cassandra

Cassandra sử dụng cơ chế hàm băm nhất quán phân tán (Distributed consistent hashing) tổ chức các nút máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo hàm băm nhất quán. Mỗi vòng tròn được coi là một Datacenter.



Hình 2.3: Phân tán dữ liệu trên cassandra

Các nút trong một cụm Cassandra sẽ được phân bố trên một vòng tròn gọi là ring (như hình trên). Mỗi nút sẽ được gán với 1 giá trị key, Cassandra dùng 127 bit để tạo ra key này. Mỗi nút trong ring sẽ quản lý một phạm vi giá trị của các key. Phạm vi của key được xác định trải đều từ giá trị của chính nút đó nắm giữ, đi ngược lại chiều kim đồng hồ cho đến khi gặp nút đầu tiên thì dừng lại. Đối chiếu lên hình, ta sẽ thấy rằng phạm vi các key mà nút T-1 quản lý nằm trong vùng (T-0; T-1]. Khi một bản ghi được ghi vào cụm Cassandra. Trường khóa của bản ghi đó sẽ được đi qua một hàm băm nhất quán, trả về một giá trị key 127bit, giá trị key này nằm trong vùng kiểm soát của nút nào thì bản ghi đó sẽ được ghi vào nút đấy. Ví dụ ta có giá trị trên các trường name được băm ra như bảng sau:

|  |  |
| --- | --- |
| Name | Hash Value |
| A | -2245452657672322382 |
| B | 7723358928203680754 |
| C | -6756552657672322382 |
| D | 1168658928203680754 |

Và ta có một cụm Cassandra với 4 nút được phân bố trên **ring** như hình sau:



Hình 2.4: Cụm Cassandra với 4 nút được phân bố trên ring

Với Cassandra, chúng ta có hai chiến lược phân mảnh (xác định vị trí của từng nút trong ***ring***).

**Random partitioning:** Đây là chiến lược mặc định và được đề xuất của Cassandra, vị trí của các node được xác định hoàn toàn thông qua mảng băm MD5. Phạm vi khóa nằm trong khoảng từ **0** tới .

**Ordered partitioning:** Chiến thuật phân mảnh đảm bảo các nút được sắp xếp theo thứ tự và phạm vi key mà mỗi nút sở hữu là như nhau.

Với chiến lược phân mảnh thứ nhất, nếu như các giá trị băm xuất ra giúp cho việc đặt các nút trong vòng phù hợp thì tất cả các bản ghi sẽ được phân bố đều trên toàn cụm. Việc thêm hay bớt mỗi nút ra khỏi cụm cũng dễ dàng hơn do không phải phân bố lại vị trí các nút khác.

Với chiến lược phân mảnh thứ hai, khi mà các nút được phân bố đều vả phạm vi quản lý **key** là như nhau, những điều đó lại mang lại những bất lợi khá rõ ràng: Khó cân bằng trong cụm: Mỗi khi thêm hay bớt một nút khỏi cụm, người quản trị sẽ phải tự tái cân bằng cụm lại một cách thủ công để đảm bảo các nút phân bố đều. Nếu dữ liệu được ghi tuần tự, có thể xảy ra trường hợp hàng loạt dữ liệu được ghi vào một nút. Gây mất cân bằng trong cụm.

**Nhận xét:** Với cả hai chiến lược phân mảnh trên, vẫn để lộ ra những điểm yếu, khi số lượng nút trong vòng quá ít, hoặc các nút phân bố không đều theo giá trị băm của các bảng ghi đưa vào, rất dễ đưa đến hiện tượng mất cân bằng, quá tải trong cụm. Ngoài ra, khi thêm hay xóa một nút khỏi vòng, thì sẽ phải mất công tái cân bằng lại cụm.

**Nút ảo:** Để giải quyết nhược điểm của 2 chiến lược phân mảnh trên ta có một giải pháp đó là sử dụng ***nút ảo***. Nút ảo trông giống như một thành phần của vòng tròn trong hệ thống, nhưng bản chất nút ảo chỉ là ánh xạ của một nút vật lý đến một địa chỉ khác trong vòng. Khi dữ liệu đi vào vùng quản lý của nút ảo, nó sẽ được đưa về lưu trữ tại nút vật lý của nút ảo đó. Mỗi nút vật lý khi tham gia vào vòng sẽ được gán một vị trí của chính nút đó và gán thêm một số lượng các vị trí khác (được coi như là nút ảo của nút đó). Cassandra cấu hình mặc định mỗi một nút tham gia vòng sẽ được gán 256 nút ảo trong vòng.



Hình 2.5: Nút ảo

Hình trên thể hiện một vòng trong có 4 nút vật lý, mỗi nút được gán thêm 7 nút ảo, như vậy tổng cộng trên vòng tròn sẽ có 32 phân vùng key. Vậy tác dụng của nút ảo là gì, khi việc phân tán đều các nút ảo ra khắp vòng, số lượng nút tăng lên khiến cho các phân vùng key bé lại, việc phân vùng key bé lại mang ý nghĩa rất lớn trong việc phân bổ dữ liệu của cụm Cassandra, việc phân vùng nhỏ lại và các nút sát nhau hơn đưa hệ thống càng gần đến với việc tất cả dữ liệu sẽ được phân bổ đều khắm các nút, xác suất dữ liệu được đưa vào các nút là cân bằng nhau khi mà trên một khoảng key nhỏ ta có đầy đủ các nút ảo hoặc nút vật lý. Trường hợp hoàn hảo nhất là các nút vật lý đều có thành phần hiện diện của mình đêu khắp trên vòng. Phân vùng key quản lý khi có và không có nút ảo.

* + 1. Nhân bản dữ liệu trong Cassandra

Để thỏa mãn tính sẵn sàng và liên tục trong Cassandra, mỗi đối tượng dữ liệu có thể được nhân bản và lưu giữ trên nhiều máy chủ. Nếu một trong các máy chủ lưu một phiên bản dữ liệu bị lỗi hoặc là phiên bản cũ, không phải là phiên bản được cập nhật dữ liệu mới nhất, Cassandra có cơ chế đồng bộ để luôn đảm báo các thao tác đọc sẽ luôn trả về dữ liệu mới nhất. Đồng thời với việc này Cassandra tiến hành thao tác sửa lỗi đọc (read repair) là tiến trình ngầm để cập nhật trạng thái mới nhất cho tất cả các máy chủ lưu trữ nhân bản của dữ liệu. Cassandra tổ chức các nút máy chủ thành cụm theo định dạng vòng tròn và dữ liệu được phân tán theo vòng tròn này theo bảng hàm băm nhất quán (Distributed consistent hashing). Nếu mỗi dữ liệu của Cassandra được sao lưu trên N nút, khi một khóa k được quyết định sẽ lưu vào một nút nào đó, nút đó sẽ được coi là nút điều phối. Nút điều phối có nhiệm vụ phân phối bản ghi đấy cho N-1 nút còn lại theo nguyên tắc: từ nút điều phối, đi theo chiều kim đồng hồ, dữ liệu sẽ được ghi lên 2 nút tiếp theo được gặp. Hình trên mô tả khi khóa k được xác định là sẽ ghi vào nút B, nút B sẽ đóng vai trò điều phối, luân chuyển khóa đấy cho 2 nút tiếp theo là nút C và nút D. như vậy, nút D sẽ lưu trữ các khóa nằm trong vùng [A; D]. Danh sách các khóa trong vùng này được gọi là danh sách liên kết của nút D. Việc đưa các giá trị của khóa k sang các nút khác áp dụng cho tất cả các tác vụ ghi, cập nhật hay xóa. Vì việc việc quyết định số lượng nút được luân chuyển ngay lập tức mỗi khi có tác vụ ghi diễn ra ảnh hưởng trực tiếp đến mức độ nhất quán của hệ thống.

* + 1. Giao tiếp giữa các nút trong Cassandra

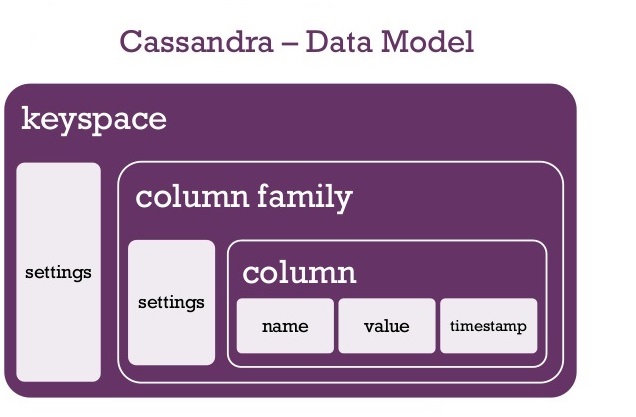
****

Hình 2.6: Việc giao tiếp giữa các nút trong cassandra

Mỗi khi cụm Cassandra bổ sung hoặc loại bỏ một nút ra khỏi cụm, dữ liệu trong cụm sẽ phải được phân bố lại. Khi bổ sung một nút, nút đó sẽ lấy đi 1 phần dữ liệu của các nút khác khi nó được cấp cho 256 nút ảo. Khi một nút bị loải khỏi cụm, dữ liệu của nút đó sẽ phải được rải đều cho các nút khác. Trong Cassandra, các nút giao tiếp với nhau thông qua giao thức ***Gossip***. Gossip là một giao thức dùng để cập nhật thông tin về trạng thái của các node khác đang tham gia vào cluster. Đây là một giao thức liên lạc dạng peer-to-peer trong đó mỗi node trao đổi định kỳ thông tin trạng thái của chúng với các node khác mà chúng có liên kết. Tiến trình gossip chạy mỗi giây và trao đổi thông tin với nhiều nhất là ba node khác trong cluster. Các node trao đổi thông tin về chính chúng và cả thông tin với các node mà chúng đã trao đổi, bằng cách này toàn bộ những node có thể nhanh chóng hiểu được trạng thái của tất cả các node còn lại trong cluster. Một gói tin gossip bao gồm cả version đi kèm với nó, như thế trong mỗi lần trao đổi gossip, các thông tin cũ sẽ bị ghi đè bởi thông tin mới nhất ở một số node. Khi một node được khởi động, nó sẽ xem file cấu hình *cassandra.yaml* để xác định tên cluster chứa nó và các nút khác trong cluster được cấu hình trong file, được biết với tên là seed node. Để ngăn chặn sự đứt đoạn trong truyền thông gossip, tất cả các nút trong cluster phải có cùng 1 danh sách các seed node được liệt kê trong file cấu hình. Bởi vì, phần lớn các xung đột được sinh ra khi 1 node được khởi động. Mặc định, 1 node sẽ phải nhớ những node mà nó đã từng gossip kể cả khi khởi động lại và seed node sẽ không có mục đích nào khác ngoài việc cập nhật 1 node mới khi nó tham gia vào cluster. Tức là, khi một node tham gia vào cluster, nó sẽ liên lạc với các seed node để cập nhật trạng thái của tất cả các node khác trong cluster. Trong những cluster có nhiều data center, danh sách seed node nên chứa ít nhất một seed node trên mỗi data center, nếu không thì khi có 1 nút mới tham gia vào cluster, thì nó sẽ liên lạc với một seed node nằm trên data center khác. Cũng không nên để mọi node đều là seed node vì nó sẽ làm giảm hiệu năng của gossip và gây khó duy trì. Việc tối ưu gossip là không quan trọng như khuyến khích nên sử dụng một danh sách nhỏ các seed node, thông thường 3 seed node trên một data center.

* + 1. Mô hình dữ liệu của Cassandras

Mô hình dữ liệu của Cassandra tuân theo cách tiếp cận hệ thống cột, mà có thể dễ dàng được hiểu tương tự như một cấu trúc bảng quan hệ nhưng theo cách NoSQL. Mô tả dưới đây sẽ làm rõ ràng hơn:



Hình 2.7: Mô hình dữ liệu của cassandra

Keyspace: Một khóa có thể được xem như là kho chứa ngoài cùng nhất cho dữ liệu trong Cassandra. Tất cả dữ liệu trong Cassandra sẽ đặt trong một keyspace. Nó có thể được xem như là một cơ sở dữ liệu trong RDBMS, nó là một tập hợp các bảng. Trong trường hợp của Cassandra, một keyspace là một tập hợp hệ thống cột.

Column family (Hệ thống Cột): Một hệ thống cột có thể được xem như là một bộ sưu tập các hàng, và mỗi hàng có một tập hợp các cột. Nó tương tự như một bảng trong RDBMS nhưng có một số khác biệt. Hệ thống cột được xác định, nhưng không cần thiết mỗi hàng phải có tất cả các cột và các cột có thể được thêm vào hoặc xoá bỏ khỏi một hàng khi cần thiết.

Column (Cột): Cột là đơn vị dữ liệu cơ bản trong Cassandra. Nó có 3 giá trị: khoá hoặc tên cột, giá trị cột, và một mốc thời gian.

* 1. 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 2.8: Quy trình CRISP-DM

* + 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
  + 1. 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
  + 1. 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
  + 1. 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
  + 1. Đánh giá mô hình (Evaluation Model)
* Đánh giá mô hình trong bối cảnh ứng với các tiêu chí nghiệp vụ
* Phê duyệt mẫu
  + 1. 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
* Lập kế hoạch bảo trì / cập nhật
* Đánh giá dự án

1. 3. Các phương pháp khai thác dữ liệu
      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.
  + 1. 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 để dự đoán về population.

* Suy luận thường là mục tiêu của các mô hình thống kê (Statistical models).
* Suy luận phụ thuộc rất nhiều vào population và sample.
  + 1. Phân tích dữ liệu thăm dò (Exploratory Data Analysis - EDA)

Là một cách tiếp cận để 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 visual. EDA khác với phân tích dữ liệu ban đầu (Initial Data Analysis - IDA) tập trung hẹp hơn trong việc kiểm tra các giả định cần thiết cho việc kiểm tra mô hình và giả thuyết, và xử lý các giá trị thiếu và thực hiện các biến đổi khi cần. EDA bao gồm IDA.

* 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.
  + 1. 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 báo 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. Nó sử dụng một số khai thác dữ liệu, mô hình tiên đoán và các kỹ thuật phân tích để tập hợp quản lý, công nghệ thông tin và quy trình nghiệp vụ mô hình hóa 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.

* + 1. Phân tích nguyên nhân (Causal analysis)
    2. Phân tích cơ chế (Mechanistic analysis)

Hiểu được chính xác những thay đổi ở các biến dẫn tới những thay đổi các biến khác.

* Khó để suy luận, trừ những tình huống đơn giản.
* Thường mô hình hóa bởi một tập các phương trình xác định.
* Nếu các phương trình được biết nhưng các thông số thì không, chúng có thể được suy ra cùng với phân tích dữ liệu.
  1. Mô hình chung cho các bài toán Machine Learning

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

Hình 2.9: Mô hình chung cho các bài toán Machine Learning

* + 1. Training phase
       1. Đầu vào

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.

* + - 1. Đầu ra
* Raw training input:

Raw input là tất cả các thông tin ta biết về dữ liệu. Ví dụ: với ảnh thì là giá trị của từng pixel; với văn bản thì là từng từ, từng câu; với file âm thanh thì nó là một đoạn tín hiệu; với cơ sở dữ liệu Iris thì nó là độ dài các cánh hoa và đài hoa,…Dữ liệu thô này thường không ở dạng vector, không có số chiều như nhau. Thậm chí có thể có số chiều như nhau nhưng số chiều quá lớn, như một bức ảnh màu 1000 pixel x 1000 pixel thì số elements đã là 3x〖10〗^6 (3 vì ảnh màu thường có 3 channels: Red, Green, Blue). Đây là một con số quá lớn, không lợi cho lưu trữ và tính toán.

* Output của training set

Trong các bài toán Unsupervised learning, ta không biết output nên hiển nhiên sẽ không có đầu vào này. Trong các bài toán Supervised learning, có khi dữ liệu này cũng không được sử dụng. Ví dụ: nếu raw input 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, 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 raw input, 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, chúng 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, chúng 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.
  + 1. Testing phase

Bước này đơn giản hơn nhiều. Với raw input 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.

* 1. Feature Engineering

Khi làm việc với các bài toán Machine Learning thực tế, nhìn chung chúng ta chỉ có được dữ liệu thô (raw) chưa qua chỉnh sửa, chọn lọc. Chúng 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.

* + 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. Chúng ta cần chọn ra một số lượng nhỏ hơn các feature phù hợp với bài toán.

*Một số cách trong Feature Selection như:*

* + - 1. Cách 1: Loại bỏ các features có giá trị phương sai thấp:

Ngưỡng phương sai (*VarianceThreshold*) 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.

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

* + - 1. Dimentionality Reduction

Khi 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 2.10: Sử dụng phương pháp PCA để giảm chiều dữ liệu 3D xuống 2D

Chú ý: Feature learning không nhất thiết phải làm giảm số chiều dữ liệu, đôi khi feature vector còn có số chiều lớn hơn raw data. Random projection cũng có thể làm được việc này nếu ma trận chiếu là một ma trận cao (số cột ít hơn số hàng).

* + 1. Bag-of-Word

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 "Bag of Words". Với test-data mới, tiến hành tìm ra số lần từng từ của test data xuất hiện trong "bag". 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(Term Frequency) là tần số xuất hiện của 1 từ trong 1 văn bản có cách tính như sau:

*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, là IDF (Inverse Document Frequency): 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.

*Trong đó:*

* ***N:*** *là tổng các văn bản trong một tập văn bản* ***N =* |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:

**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:*

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.

* + 1. 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, chúng 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:

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**.

*#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 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:

*Trong đó:* ***,***  *lần lượt là kỳ vọng và phương sai (standard deviation) của thành phần đó trên toàn bộ training data.*

*#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:

Chương 3:  
MỘT SỐ MÔ HÌNH MACHINE LEARNING

1. 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.

* + 1. Giới thiệu

Một căn nhà rộng  , có  phòng ngủ và cách trung tâm thành phố  **km** có giá là bao nhiêu. Giả sử chúng ta đã có số liệu thống kê từ 1000 căn nhà trong thành phố đó, liệu rằng khi có một căn nhà mới với các thông số về diện tích, số phòng ngủ và khoảng cách tới trung tâm, chúng ta có thể dự đoán được giá của căn nhà đó không? Nếu có thì hàm dự đoán  sẽ có dạng như thế nào. Ở đây  là một vector hàng chứa thông tin input,  là một số vô hướng (scalar) biểu diễn output (tức giá của căn nhà trong ví dụ này).

Một cách đơn giản nhất, chúng ta có thể thấy rằng:

* Dện 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:

Trong đó,  là các hằng số, còn được gọi là **bias tức hệ số tự do**. Mối quan hệ  bên trên là một mối quan hệ tuyến tính (*linear*). Bài toán chúng ta đang làm là một bài toán thuộc loại regression. Bài toán đi tìm các hệ số tối ưu chính vì vậy được gọi là bài toán Linear Regression.

* + 1. Dạng của Linear Regression

*Trong đó:*

* **w = {}** là vector (cột) cần phải tối ưu.
* là vector (hàng) dữ liệu đầu vào, số **1** được đặt ở đầu để phép tính đơn giản và thuận tiện hơn cho việc tính toán.
  + 1. Sai số dự đoán

Ta mong muốn rằng sự sai khác  giữa giá trị thực  và giá trị dự đoán là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

**=**

Trong đó hệ số để thuận tiện cho việc tính toán (khi tính đạo hàm thì số sẽ bị triệt tiêu). Còn vì , việc ta lấy bình phương cũng là để thuận tiện trong việc đạo hàm.

* + 1. Hàm mất mát (Cost Function /Lost Function)

Điều tương tự xảy ra với tất cả các cặp (input, outcome) với **N** là số dữ liệu quan sát được. Điều chúng ta muốn, tổng sai số là nhỏ nhất, tương đương với việc tìm **w** để hàm số sau đạt giá trị nhỏ nhất:

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

Trước khi đi tìm lời giải, chúng ta đơn giản hóa phép toán trong phương trình hàm mất mát . Đặt là một vector cột chứa tất cả các output của training data. 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  được viết dưới dạng ma trận đơn giản hơn:

Với là Euclidean norm (chuẩn Euclid, hay khoảng cách Euclid), nói cách khác 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 .

* + 1. Nghiệm trên bài toán Linear Regression

Để tìm nghiệm cho bài toán tối ưu là giải phương trình đạo hàm (gradient) bằng **0**.

Đạo hàm theo của hàm mất mát là:

Phương trình đạo hàm bằng tương đương với:

*Ký hiệu tức là đặt bằng*

Nếu ma trận vuông khả nghịch (non-singular hay invertible) thì phương trình có nghiệm duy nhất: .

Nếu ma trận vuông không khả nghịch (có định thức bằng 0), thì phương trình vô nghiệm hoặc vô số nghiệm. Khi đó ta sẽ xử dụng khái niệm **pseudo inverse** (giả nghịch đảo)  **(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:

* + 1. 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 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 | **?** |

Để kiểm tra độ chính xác của model tìm được, chúng 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

*# 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.

* + 1. 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 biễu diễn được các mô hình phức tạp. Mặc dù trong phần trên, chúng ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa outcome và input không nhất thiết phải là tuyến tính, nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế, những mô hình có các hàm như **.**

* 1. Gradient Decent

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 chúng 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.

* + 1. Gradient Decent cho hàm một biến

****

Hình 3.5: Parapol của hàm f(x)

Giả sử là điểm ta tìm được sau vòng lặp **t**. Ta cần tìm một thuật toán để đưa về càng gần càng tốt.

Từ hình trên ta thấy: Nếu đạo hàm của hàm số tại thì nằm về bên phải so với (và ngược lại). Để điểm tiếp theo gần với hơn, chúng ta cần di chuyển về phía bên trái, tức về phía âm. Nói cách khác, ta cần di chuyển ngược dấu với đạo hàm

*Trong đó:*

* *là một đại lượng ngược dấu với đạo hàm*
* *càng xa về phía bên phải thì* *càng lớn hơn 0 (và ngược lại). Vậy, lượng di chuyển , một cách trực quan nhất, là tỉ lệ thuận với*

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

*Trong đó: (đọc là eta) là một số dương được gọi là learning rate (tốc độ học). Dấu trừ thể hiện việc chúng ta phải đi ngược với đạo hàm (Đây cũng chính là lý do phương pháp này được gọi là Gradient Descent - descent nghĩa là đi ngược).*

**Ví dụ đơn giản với Python:**

Xét hàm số với đạo hàm  **2x + 5cos(x).** Giả sử bắt đầu từ một điểm nào đó, tại vòng lặp thứ **,** chúng ta sẽ cập nhật như sau:

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 () và điểm bắt đầu. Thuật toán dừng lại khi đạo hàm có độ lớn đủ nhỏ

Các điểm khởi tạo khác nhau là và

**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 chúng 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 , nghiệm hội tụ nhanh hơn, vì điểm ban đầu gần với nghiệm hơn. Hơn nữa, với 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

* + 1. Gradient Decent cho hàm nhiều biến

Giả sử ta cần tìm global minimum cho hàm trong đó **θ** (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). Đạo hàm của hàm số đó tại một điểm **θ** bất kỳ được ký hiệu là (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 **,** sau đó, ở vòng lặp thứ , quy tắc cập nhật là:

* + 1. Stopping Criteria (điều kiện dừng)

Khi nào thì chúng 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 bẳng 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.
  + 1. Biến thể của Gradient Descent
       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 , chúng ta sử dụng tất cả các điểm dữ liệu .

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.

* + - 1. 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 rồi cập nhật 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 . Với SGD thì mỗi epoch ứng với lần cập nhật với lần cập nhật với 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.

* + 1. 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:

*Trong đó: là hàm mất mát với chỉ 1 cặp điểm dữ liệu (input, label) là .*

* 1. K-Means
     1. Giới thiệu

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

* + 1. 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). Hình bên dưới là một ví dụ về 3 cụm dữ liệu.



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.

* + 1. Phân tích mặt toán học

Giả sử có điểm dữ liệu là và là số cluster chúng ta muốn phân chia. Chúng ta cần tìm các center và label của mỗi điểm dữ liệu.

Với mỗi điểm dữ liệu đặt là label vector của nó, trong đó nếu được phân vào cluster thì và . Điều này có nghĩa là có đúng một phần tử của vector là bằng 1 (tương ứng với cluster của) các phần tử còn lại bằng 0. Ví dụ: nếu một điểm dữ liệu có label vector là thì nó thuộc vào cluster 1, là 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 có thể viết dưới dạng toán học như sau:

* + 1. Thuật toán

**Đầu vào:** Dữ liệu  và số lượng cluster cần tìm .

**Đầu ra:** Các center và label vector cho từng điểm dữ liệu .

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

Nếu ta coi center là center (hoặc representative) của mỗi cluster và ước lượng tất cả các điểm được phân vào cluster này bởi thì một điểm dữ liệu được phân vào cluster sẽ bị sai số là **( )**. Chúng 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:

Hơn nữa, vì được phân vào cluster nên . Khi đó, biểu thức bên trên sẽ được viết lại là:

Sai số cho toàn dữ liệu sẽ là:

*Trong đó,* *lần lượt là các ma trận được tạo bởi label vector của mỗi điểm dữ liệu và center của mỗi cluster. là hàm số mất mát với ràng buộc như trong phương trình .*

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

* + 1. Thuật toán tối ưu hàm mất mát

Một cách đơn giản để giải bài toán là xen kẽ giải và 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:

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

**=**

Chỉ có một phần tử của vector label bằng 1 nên bài toán **(3)** có thể tiếp tục viết dưới dạng đơn giản hơn:

Vì chính là bình phương khoảng cách tính từ điểm tới center , ta có thể kết luận rằng mỗi điểm thuộc vào cluster có center gần nó nhất. Từ đó ta có thể dễ dàng suy ra label vector của từng điểm dữ liệu.

* + - 1. 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:

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 nên chúng ta sẽ tìm được giá trị nhỏ nhất và điểm tối ưu tương ứng.

Đặt là hàm bên trong dấu , ta có đạo hàm:

Giải phương trình đạo hàm bằng 0 ta có:

Ta thấy là trung bình cộng của các điểm trong cluster

* 1. Logistic Regression
     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).

* + 1. 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 |

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

* + 1. Mô hình Logistic Regression

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

*Trong đó: 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:

* + Là hàm số liên tục nhận giá trị thực, bị chặn trong khoảng .
  + 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.
  + Mượt (smooth) nên có đạo hàm mọi nơi, có thể được lợi trong việc tối ưu.
    1. Sigmoid Function

Đây là hàm được sử dụng nhiều nhất, vì nó bị chặn trong khoảng . Thêm nữa:

**;**

* + 1. Tanh Function (Hyperpolic Tangent)

Hàm số này nhận giá trị trong khoảng **,** với khoảng hàm có dạng:

* 1. Artificial Neural Network (ANN)
     1. Kiến trúc tổng quát của một ANN

Mạng Neuron nhân tạo (Artificial Neural Network- ***ANN***) 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 3.15: Kiến trúc tổng quát của một mạng Neuron nhân tạo

Processing Elements (PE): Các PE của ANN gọi là Neuron, mỗi Neuron nhận các dữ liệu vào (Inputs) xử lý chúng và cho ra một kết quả (output) duy nhất. Kết quả xử lý của một Neuron có thể làm Input cho các Neuron khác

Kiến trúc chung của một ANN gồm 3 thành phần đó là 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 theo. Trong một ANN có thể có nhiều Hidden Layer.

* + 1. Quá trình xử lý thông tin của một ANN

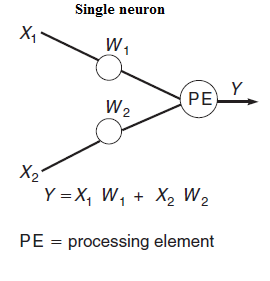


Hình 3.16: Quá trình xử lý thông tin của một ANN

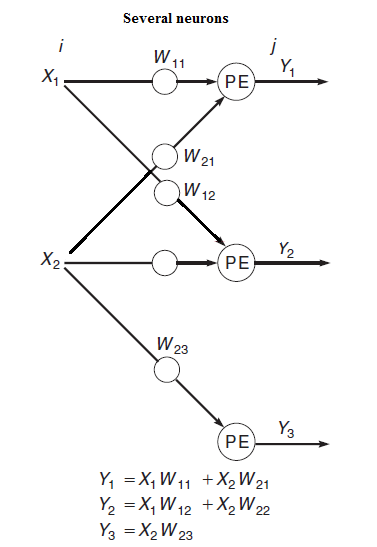
**Inputs**: Mỗi Input tương ứng với 1 thuộc tính (attribute) của dữ liệu (patterns).

**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 (phần tử xử lý PE). Hàm tổng của một Neuron đối với **n** input được tính theo công thức sau:

Hình 3.17: Hàm tổng của một neuron



Hình 3.18: Hàm tổng của nhiều neuron

* + 1. Transfer function

Hàm tổng (Summation 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 3.19: 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.

*Trong đó:*

* *: hàm chuyển đổi*
* *: 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. Nếu output của một neuron nào đó nhỏ hơn Threshold thì nó sẻ không được chuyển đến Layer tiếp theo.

* + 1. Quá trình học (Learning Processing) của ANN

Supervised learning: 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 3.20: Quá trình học của ANN

1. Tính giá trị output .
2. So sánh output với giá trị mong muốn (desired value).
3. Nếu chưa đạt được giá trị mong muốn thì hiệu chỉnh trọng số (weights) và tính lại output
   * 1. 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).

* 1. Support Vector Machines

Chương 4:  
ĐÁNH GIÁ MÔ HÌNH (MODEL EVALUATION)

* 1. Giới Thiệu

Khi xây dựng một mô hình Machine Learning, chúng 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 chúng 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, Top R error.

* 1. 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.

* 1. Confusion Matrix (CM)

Cách tính sử dụng Accuracy như ở trên chỉ cho chúng 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, chúng 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:



Có tổng cộng 165 điểm dữ liệu. Chúng 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.

Chúng 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:

**import** numpy **as** np  
**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 ma trận y\_true* 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 :'**)  
print(cnf\_matrix)  
*#Độ chính xác Accuracy được tính bằng tổng các thành phần trên đường chéo chính chia cho tổng các thành phần trên ma trận*print(**'\nĐộ chính xác Accuracy: {}%'**.format(int(np.diagonal(cnf\_matrix).sum()/cnf\_matrix.sum()\*100)))

*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 ]

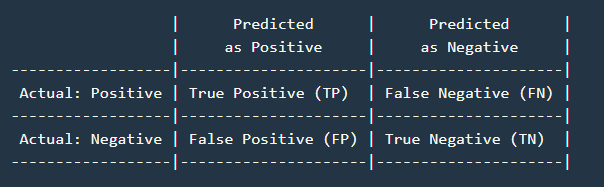
[ 0.33333333 0.66666667 0. ]

[ 0. 0.33333333 0.66666667]]

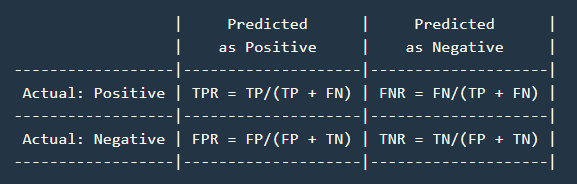
* 1. True/False Positive/Negative

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:



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



**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** cho mỗi lớp nếu coi lớp đó là lớp Positive, các lớp còn lại gộp chung thành lớp Negative.
  1. Receiver operating characteristic curve

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).



Hình 4.1: Ví dụ về đường ROC

* 1. 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.

* 1. 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:

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.

* 1. 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.



KẾT LUẬN

PHỤ LỤC

TÀI LIỆU THAM KHẢO