**BỘ QUỐC PHÒNG**

**TRƯỜNG ĐẠI HỌC THÔNG TIN LIÊN LẠC**

**KHÓA LUẬN TỐT NGHIỆP**

***Đề tài :***

**VẬN DỤNG MACHINE LEARNING VÀO KHAI PHÁ VÀ DỰ ĐOÁN DỮ LIỆU**

**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 :***

**VẬN DỤNG MACHINE LEARNING VÀO KHAI PHÁ VÀ DỰ ĐOÁN DỮ LIỆU**

**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.

Nha Trang, tháng 06 năm 2018

Sinh Viên Thực Hiện

Trịnh Đình Phúc

DANH MỤC BẢNG BIỂU VÀ HÌNH VẼ

Bảng

Hình ảnh

MỤC LỤC

MỞ ĐẦU

Chương 1:  
TỔNG QUAN ĐỀ TÀI VẬN DỤNG MACHINE LEARNING VÀO KHAI PHÁ VÀ DỰ ĐOÁN DỮ LIỆU

* 1. Định nghĩa tên đề tài

**Vận Dụng**: Sự vận dụng (tiếng anh: Manipulation) là việc đem tri thức, lí luận áp dụng vào thực tiễn, vận dụng những kiến thức đã học vào việc ứng dụng, tạo ra sản phẩm.

**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.

**Khai Phá – Dự đoán**:

* 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 (Data Mining Big Data)” 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. Các yêu cầu tính năng cơ bản đối với sản phẩm

Sản phẩm từ nghiên cứu đề tài “Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn” cần có các tính năng cơ bản như sau:

* Data Cleaning.
* ETL (Extract – Transform - Load).
* Feature Selection.
* Feature Engineering .
* Modeling.
* Evaluation Model.
* Tunning Model.

Từ đó có thể chia sản phẩm thành 4 phần chính đảm nhiệm các nhiệm vụ khác nhau:

* Data Understanding (hiểu dữ liệu).
  + Xem xét các yêu cầu về dữ liệu.
  + Thu thập dữ liệu.
  + Data Exploitation.
  + Thống kê dữ liệu.
* Data Preparation (chuẩn bị dữ liệu).
  + Feature Selection.
  + ETL (Extract – Transform – Load).
  + Data Cleaning.
  + Data Transformation.
  + Feature Scaling.
* Modeling (mô hình hóa).
  + Lựa chọn kỹ thuật mô hình thích hợp.
  + Train-Test-Split.
  + Tunning parameter.
  + Fitting model.
* Model Evaluation.
  + Hypothesis Testing.
  + Hold-out.
  + Confusion matrix.
  + Cross-Validation.
  + Root Mean Square Error (RMSE).
  + Receiver Operating Characteristic (ROC).
  1. Trọng tâm đề tài

Đúng như tên đề tài đã nêu “Tìm hiểu và vận dụng phương pháp xử lý dữ liệu lớn” thì trọng tâm sẽ là triển khai quy trình khai phá dữ liệu.

Như vậy tài nguyên này và cơ chế để tự sinh website sẽ là đối tượng khai thác chính trong đề tài “ứng dụng tự dộng sinh website”. Ngoài ra còn có các chức năng quản lý kèm theo không kém phần quan trọng để có thể tạo ra một ứng dụng đạt được nhu cầu thực tiền với yêu cầu đề tài.

* 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. Năm 2008, Facebook chuyển nó cho cộng đồng mã nguồn mở và được Apache tiếp tục phát triển đến ngày hôm nay. Cassandra được coi là sự kết hợp của Amazon’s Dynamo và Google’s BigTable. 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 1: 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: 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 3: 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 4: 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

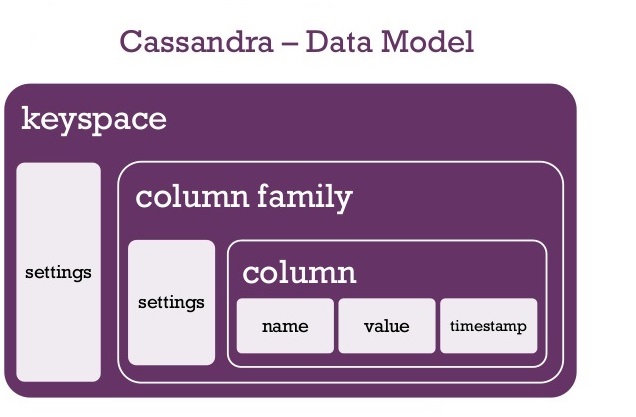
****

Hình 5: 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 6: 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 7: 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 8: 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 9: 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**.

Ở Python, thư viện sklearn cho phép thực hiện scale khoảng **[0, 1]** đơn giản với hàm: min\_max\_scaler()

*#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:  
TỔNG QUAN VỀ CÁC 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 10: Hiển thị dữ liệu về chiều cao & cân nặng trên đồ thị

**Nghiệm theo công thức:**

Công thức: **Can\_nang =** **w\_1\*(chieu\_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 11: 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 12: Mô hình Linear Regression sau khi học xong

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
  2. K-Means
  3. Logistic Regression
  4. Neural Network
  5. Neural Network

Chương 4:  
CÀI ĐẶT THỬ NGHIỆM

1. 1. Website quản lý tổng:
      1. Đăng nhập



Hình 12: Giao diện đăng nhập website quản lý tổng

* + 1. Quản lý người dùng



Hình 13: Giao diện quản lý người dùng website quản lý tổng

* + 1. Quản lý người dùng ứng dụng



Hình 14: Giao diện quản lý người dùng ứng dụng

* + 1. Quản lý mã nguồn website dùng chung
       1. Thêm mã nguồn mới



Hình 15: Giao diện quản lý mã nguồn website dùng chung

* + - 1. Thêm các bản nâng cấp mã nguồn đã có



Hình 16: giao diện cập nhật phiên bản mã nguồn dùng chung

* 1. Hệ thống ứng dụng tự sinh website
     1. Đăng nhập



Hình 17: Giao diện đăng nhập ứng dụng tự động sinh website

* + 1. Quản lý người dùng



Hình 18: Giao diện quản lý người dùng

* + 1. Phân quyền



Hình 19: Giao diện phân quyền người dùng

* + 1. Quản lý mã nguồn website
       1. Thêm mã nguồn mới



Hình 20: Giao diện thêm mã nguồn mới

* + - 1. Thêm các bản nâng cấp



Hình 21: Giao diện cập nhật phiên bản cho mã nguồn

* + - 1. Thêm các tệp cơ sở dữ liệu tùy chỉnh



Hình 22: Giao diện thêm các tệp cơ sở dữ liệu tùy chỉnh

* + 1. Quản lý triển khai website



Hình 23: Giao diện quản lý triển khai website

* + 1. Sinh website
       1. Sinh tệp



Hình 24: Tính năng sinh tệp

* + - 1. Sinh cơ sở dữ liệu



Hình 25: Tính năng sinh cơ sở dữ liệu

* + - 1. Cấu hình kết nối server



Hình 26: Tính năng cấu hình kết nối server

* + - 1. Tự động triển khai



Hình 27: Tính năng tự động triển khai mã nguồn lên server

KẾT LUẬN

PHỤ LỤC

TÀI LIỆU THAM KHẢO