|  |
| --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **BÀI TẬP LỚN TRÍ TUỆ NHÂN TẠO & ỨNG DỤNG**  ***Đề tài: Predict Students' Dropout or Academic Success***  **Nguyễn Thành Trung**  **20203915**  Hà Nội, 2- 2023 |

**LỜI NÓI ĐẦU**

Trí tuệ nhân tạo, Internet kết nối vạn vật hay Dữ liệu lớn là những yếu tố cốt lõi của Cách mạng công nghiệp 4.0. Không thể phủ nhận lợi ích đem lại của Trí tuệ nhân tạo khi nó có thể xữ lí dữ liệu nhanh hơn, khoa học hơn, hệ thống hơn với quy mô rộng hơn so với con người. Chính vì lý do này, Trí tuệ nhân tạo đóng một vai trò vô cùng quan trọng và thiết thực trong cuộc sống của chúng ta: Sử dụng trong hệ thống ngân hàng tài chính, sử dụng trong y học, với các ngành công nghiệp nặng, vận tải hàng hóa và ngay cả với những trò chơi siêu đẳng.

Là một sinh viên đang ngồi trên ghế nhà trường, em đã được trau dồi một số kiến thức chuyên môn của ngành học nói chung và môn Trí tuệ nhân tạo & Ứng dụng nói riêng. Việc học lý thuyết và bài tập trên lớp đã giúp chúng em mở rộng vốn hiểu biết của mình, tuy nhiên đó chỉ là một phần kiến thức nhỏ bé so với khối kiến thức thực tế mà chúng em sẽ phải đối mặt khi tốt nghiệp. Vì vậy, chúng em mong muốn có thể vận dụng các kiến thức đã học vào các sản phẩm thực tế như trong bài tập lớn môn học này với đề tài: Predict Students’ Dropout or Academic Success.

Quy trình thực hiện vấn đề bao gồm:

* Phân tích dữ liệu.
* Thử nghiệm các mô hình.
* Lựa chọn mô hình.
* Tinh chỉnh dữ liệu và mô hình.
* Đánh giá kết quả.

**MỤC LỤC**

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT i](#_Toc110197544)

[DANH MỤC HÌNH VẼ ii](#_Toc110197545)

[DANH MỤC BẢNG BIỂU iii](#_Toc110197546)

[TÓM TẮT BÁO CÁO iv](#_Toc110197547)

[PHÂN CÔNG NHIỆM VỤ v](#_Toc110197548)

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 6](#_Toc110197549)

[1.1 Trí tuệ nhân tạo – Học máy – Học sâu 6](#_Toc110197550)

[1.1.1 Trí tuệ nhân tạo 6](#_Toc110197551)

[1.1.2 Học máy 7](#_Toc110197552)

[1.1.3 Học sâu 8](#_Toc110197553)

[1.2 Thuật toán sử dụng 8](#_Toc110197554)

[1.2.1 Logistic Regression 8](#_Toc110197555)

[1.2.2 K-nearest Neighbor 10](#_Toc110197556)

[1.2.3 Decision Tree 10](#_Toc110197557)

[1.2.4 Support Vector Machine 13](#_Toc110197558)

[1.2.5 Random Forest 14](#_Toc110197559)

[1.2.6 Grid Search CV 15](#_Toc110197560)

[1.2.7 Naïve Bayes 16](#_Toc110197561)

[1.3 Kiến thức khác 18](#_Toc110197562)

[1.3.1 Precision – Recall – F1 Score 18](#_Toc110197563)

[1.3.2 One – hot encoder 19](#_Toc110197564)

[1.3.3 Chuẩn hóa dữ liệu 19](#_Toc110197565)

[1.3.4 Kỹ thuật lấy quá mức cho dữ liệu nhãn thiểu số 20](#_Toc110197566)

[1.3.5 Phương pháp SMOTE Edited Nearest Neighbor 20](#_Toc110197567)

[CHƯƠNG 2. PREDICT BOSTON HOUSING PRICES 22](#_Toc110197568)

[2.1 Nhập thư viện và load dữ liệu 22](#_Toc110197569)

[2.1.1 Nhập thư viện 22](#_Toc110197570)

[2.1.2 Load dữ liệu 22](#_Toc110197571)

[2.1.3 Lấy dữ liệu theo yêu cầu 22](#_Toc110197572)

[2.2 Minh họa dữ liệu 22](#_Toc110197573)

[2.2.1 Code 23](#_Toc110197574)

[2.2.2 Kết quả 23](#_Toc110197575)

[2.3 Viết hàm Cost function 23](#_Toc110197576)

[2.3.1 Cost function thông thường 23](#_Toc110197577)

[2.3.2 Cost function được vector hóa 24](#_Toc110197578)

[2.4 Viết hàm Gradient Descent 25](#_Toc110197579)

[2.4.1 Gradient Descent thông thường 25](#_Toc110197580)

[2.4.2 Gradient Descent vector hóa 25](#_Toc110197581)

[2.5 Tối ưu tham số theta 26](#_Toc110197582)

[2.5.1 Viết hàm tối ưu 26](#_Toc110197583)

[2.5.2 Thử nghiệm và chạy kết quả 26](#_Toc110197584)

[2.6 Tối ưu hàm cost bài toán đặt ra 27](#_Toc110197585)

[2.6.1 Lựa chọn tính năng 27](#_Toc110197586)

[2.6.2 Chuẩn hóa dữ liệu 27](#_Toc110197587)

[2.6.3 Xây dựng mô hình 27](#_Toc110197588)

[2.6.4 Kiểm tra quá trình training 27](#_Toc110197589)

[CHƯƠNG 3. PREDICT DROPOUT OR ACADEMIC SUCCESS 29](#_Toc110197590)

[3.1 Thử nghiệm với các mô hình truyền thống 29](#_Toc110197591)

[3.2 Build mô hình mạng Neural 30](#_Toc110197592)

[3.2.1 One – Hot Encoder 30](#_Toc110197593)

[3.2.2 Xây dựng mạng neural network 30](#_Toc110197594)

[3.2.3 Training model 31](#_Toc110197595)

[3.2.4 Đánh giá model 32](#_Toc110197596)

[3.3 Tiền xử lý tập data & Cải thiện mô hình Neural Network 32](#_Toc110197597)

[3.3.1 Kỹ thuật lấy mẫu quá mức cho dữ liệu nhãn thiểu số 33](#_Toc110197598)

[3.3.2 Kỹ thuật SMOTE-ENN cho dữ liệu nhãn thiểu số 34](#_Toc110197599)

[KẾT LUẬN 36](#_Toc110197600)

[TÀI LIỆU THAM KHẢO 37](#_Toc110197601)

# DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| Từ viết tắt | Tiếng Anh | Tiếng Việt |
| AI | Aritificial Intelligence | Trí tuệ nhân tạo |
| ML | Machine Learning | Học máy |
| ENN | Edited Nearest Neighbor | Thuật toán KNN chỉnh sửa |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# DANH MỤC HÌNH VẼ

[Hình 1.1 Đồ thị biểu diễn hàm S0(t) 9](#_Toc110197602)

[Hình 1.2 Cấu trúc chung của cây quyết định 11](#_Toc110197603)

[Hình 1.3 Minh họa thuật toán Support Vector Machine 13](#_Toc110197604)

[Hình 1.4 Minh họa kỹ thuật mã hóa One – Hot 19](#_Toc110197605)

[Hình 2.1 Minh họa cho 2 đặc tính INDUS và RM 23](#_Toc110197606)

[Hình 2.2 Hàm cost qua 10000 lần gradient descent 28](#_Toc110197607)

[Hình 3.1 Quá trình training model 31](#_Toc110197608)

[Hình 3.2 Đánh giá model mạng neural network 32](#_Toc110197609)

[Hình 3.3 Theo dõi hàm loss và tính chính xác qua 400 epochs của mô hình 34](#_Toc110197610)

[Hình 3.4 Theo dõi hàm loss và độ chính xác mô hình qua 300 epochs 35](#_Toc110197611)

# DANH MỤC BẢNG BIỂU

[Bảng 3.1 Bảng thống kê độ chính xác training của 7 mô hình truyền thống 28](#_Toc109051587)

# TÓM TẮT BÁO CÁO

Báo cáo là quá trình ôn tập lại kiến thức trên lớp cũng như tìm hiểu thêm và áp dụng vào bài toán thực tế của nhóm. Bao gồm:

* Chương 1: Cơ sở lý thuyết
* Chương 2: Phần 1 – Linear Regression
* Chương 3: Phần 2 – Predict Dropout or Academic Success
* Chương 4: Tổng kết

# CƠ SỞ LÝ THUYẾT

## Trí tuệ nhân tạo – Học máy – Học sâu

### Trí tuệ nhân tạo

Trí tuệ nhân tạo (AI) là một nhánh rộng lớn của khoa học máy tính liên quan đến việc xây dựng các máy móc thông minh có khả năng thực hiện các tác vụ thường đòi hỏi trí thông minh của con người.

4 loại Trí tuệ nhân tạo bao gồm:

* Reactive Machine: tuân theo những nguyên tắc cơ bản nhất của AI. Như tên gọi của nó, chỉ có khả năng sử dụng trí thông minh của mình để nhận thức và phản ứng với thế giới trước mặt nó. Reactive Machine không thể lưu trữ bộ nhớ và kết quả là không thể dựa vào kinh nghiệm trong quá khứ để thông báo cho việc ra quyết định trong thời gian thực.
* Limited Memory: có khả năng lưu trữ dữ liệu và dự đoán trước đó khi thu thập thông tin và cân nhắc các quyết định tiềm năng - về cơ bản là nhìn vào quá khứ để tìm manh mối về những gì có thể xảy ra tiếp theo. Limited Memory AI phức tạp hơn và thể hiện nhiều khả năng hơn so với Reactive Machine.
* Theory of Mine: Khái niệm này dựa trên tiền đề tâm lý của việc hiểu rằng các sinh vật sống khác có những suy nghĩ và cảm xúc ảnh hưởng đến hành vi của bản thân. Về AI machine, điều này có nghĩa là AI có thể hiểu cách con người, động vật và các máy móc khác cảm thấy và đưa ra quyết định thông qua sự tự phản ánh và quyết tâm, sau đó sẽ sử dụng thông tin đó để đưa ra quyết định của riêng chúng. Về cơ bản, máy móc sẽ phải có khả năng nắm bắt và xử lý khái niệm “tâm trí”, những dao động của cảm xúc trong quá trình ra quyết định và một loạt các khái niệm tâm lý khác trong thời gian thực, tạo ra mối quan hệ hai chiều giữa con người và trí tuệ nhân tạo.
* Self-awareness: Loại trí tuệ nhân tạo này sở hữu ý thức cấp độ con người và hiểu được sự tồn tại của chính nó trên thế giới, cũng như sự hiện diện và trạng thái cảm xúc của người khác. Nó có thể hiểu những gì người khác có thể cần dựa trên không chỉ những gì họ truyền đạt cho họ mà còn bằng cách họ truyền đạt nó.

### Học máy

Máy học là một nhánh của trí tuệ nhân tạo (AI) và khoa học máy tính, tập trung vào việc sử dụng dữ liệu và thuật toán để bắt chước cách con người học, dần dần cải thiện độ chính xác của nó.

Hệ thống học tập của một thuật toán học máy gồm 3 thành phần chính:

* Quy trình quyết định: Nói chung, các thuật toán học máy được sử dụng để đưa ra dự đoán hoặc phân loại. Dựa trên một số dữ liệu đầu vào, có thể được gắn nhãn hoặc không được gắn nhãn, thuật toán sẽ đưa ra ước tính về một mẫu trong dữ liệu.
* Hàm lỗi: đánh giá dự đoán của mô hình: Nếu có các ví dụ đã biết, một hàm lỗi có thể thực hiện so sánh để đánh giá độ chính xác của mô hình.
* Quy trình tối ưu hóa mô hình: Nếu mô hình có thể phù hợp hơn với các điểm dữ liệu trong tập huấn luyện, thì trọng số được điều chỉnh để giảm sự khác biệt giữa ví dụ đã biết và ước tính mô hình. Thuật toán sẽ lặp lại quá trình “đánh giá và tối ưu hóa” này, cập nhật các trọng số một cách tự động cho đến khi đạt đến ngưỡng độ chính xác

Các mô hình học máy được chia thành 3 loại chính:

* Học có giám sát, còn được gọi là học máy có giám sát, được định nghĩa bằng cách sử dụng các tập dữ liệu được gắn nhãn để huấn luyện các thuật toán phân loại dữ liệu hoặc dự đoán kết quả một cách chính xác. Khi dữ liệu đầu vào được đưa vào mô hình, mô hình sẽ điều chỉnh trọng lượng của nó cho đến khi nó được lắp một cách thích hợp.
* Học không giám sát, còn được gọi là học máy không giám sát, sử dụng các thuật toán học máy để phân tích và phân cụm các tập dữ liệu không được gắn nhãn. Các thuật toán này phát hiện ra các mẫu hoặc nhóm dữ liệu ẩn mà không cần sự can thiệp của con người.
* Học tập bán giám sát cung cấp một phương tiện giữa học tập có giám sát và không giám sát. Trong quá trình đào tạo, nó sử dụng một tập dữ liệu có nhãn nhỏ hơn để hướng dẫn phân loại và trích xuất tính năng từ một tập dữ liệu lớn hơn, không được gắn nhãn.

### Học sâu

Học sâu là một lĩnh vực con của học máy liên quan đến các thuật toán lấy cảm hứng từ cấu trúc và chức năng của não được gọi là mạng thần kinh nhân tạo.

Học sâu có thể nhập dữ liệu phi cấu trúc ở dạng thô của nó (Ví dụ: văn bản hoặc hình ảnh) và nó có thể tự động xác định tập hợp các tính năng giúp phân biệt các danh mục dữ liệu khác nhau với nhau. Điều này giúp loại bỏ một số sự can thiệp của con người cần thiết và cho phép sử dụng các tập dữ liệu lớn hơn.

Học sâu và mạng thần kinh được ghi nhận là giúp tăng tốc tiến bộ trong các lĩnh vực như thị giác máy tính, xử lý ngôn ngữ tự nhiên và nhận dạng giọng nói.

## Thuật toán sử dụng

### Logistic Regression

#### Định nghĩa

Logistic Regresion là một thuật toán dựa vào thống kê đánh giá các input đầu vào (1 vài feature X) và trả về kết quả (y) là một giá trị đại diện cho một sự kiện có xảy ra hay không. Với y = 1 nói lên là sự kiện đó sẽ xảy ra còn với y = 0 thì sự kiện đó không xảy ra.

Với các bài toán Regression thông thường kết quả sẽ trả về sẽ là một đại lượng liên tục còn với Logistic Regression thì kết quả trả về sẽ là một đại lượng rời rạc.

#### Lý thuyết toán học

Logistic Regression là một loại thuật toán supervised learning tính toán mối quan hệ giữa các feature trong input và output dựa trên hàm logistic/sigmoid. Mặc dù gọi là Logistic Regression nhưng thuật toán này không dự đoán ra giá trị thực như các thuật toán Regression khác, Logistic Regression được dùng để dự đoán ra một kết quả nhị phân (với giá trị 0/1 hay -1/1 hay True/False) dựa vào input của nó. Nhưng Logistic Regression cũng có một chút giống với Linear Regression trong quá trình xây dựng model.

Logistic Regression thực hiện chạy kết quả thông qua một hàm non-linear (phi tuyến tính) đặc biệt được gọi là hàm logistic hoặc hàm sigmoid để tạo ra đầu ra là một xác suất p.

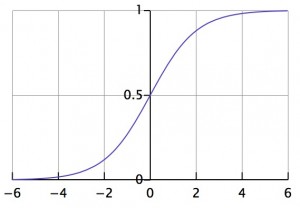
#### Công thức hồi quy của model Logistic Regression:

Với log() được gọi là logit(p) hay còn gọi là *log-odds* ta sẽ tính được xác suất p như sau

p =

Logistic được ký hiệu là S0 là hàm sigmoid với đầu ra là một số có giá trị từ 0 đến 1 được định nghĩa với công thức bên dưới.

S0(t) = với đồ thị được biểu thị bên dưới.



Hình 1.1 Đồ thị biểu diễn hàm S0(t)

Chú ý rằng So(t) < 0.5 khi t < 0 và So(t) ≥ 0.5 khi t  ≥  0 mỗi khi Logistic Regression tính xác suất P model sẽ đưa ra dự đoán với công thức sau:

0 Nếu p < 0.5

1 Nếu p ≥0.5

Với y^ là kết quả dự đoán

### K-nearest Neighbor

#### Định nghĩa

Thuật toán K-nearest Neighbor là một kĩ thuật học có giám sát (supervised learning) dùng để phân loại quan sát mới bằng cách tìm điểm tương đồng giữa quan sát mới này với dữ liệu sẵn có.

#### Ứng dụng

KNN là một mô hình đơn giản và trực quan nhưng vẫn có hiệu quả cao vì nó không tham số; mô hình không đưa ra giả định nào về việc phân phối dữ liệu. Hơn nữa, nó có thể được sử dụng trực tiếp để phân loại đa lớp.

Thuật toán KNN có nhiều ứng dụng trong ngành đầu tư, bao gồm dự đoán phá sản, dự đoán giá cổ phiếu, phân bổ xếp hạng tín dụng trái phiếu doanh nghiệp, tạo ra chỉ số vốn và trái phiếu tùy chỉnh.

#### Ưu điểm

* Thuật toán đơn giản, dễ dàng triển khai
* Độ phức tạp tính toán nhỏ
* Xử lý tốt với tập dữ liệu nhiễu

#### Nhược điểm

* Với K nhỏ dễ gặp nhiễu dẫn tới kết quả đưa ra không chính xác.
* Cần nhiều thời gian để thực hiện do phải tính toán khoảng cách với tất cả các đối tượng trong tập dữ liệu.
* Cần chuyển đổi kiểu dữ liệu thành các yếu tố định tinh.

### Decision Tree

#### Định nghĩa

Cây quyết định là một kỹ thuật học tập có giám sát có thể được sử dụng cho cả vấn đề phân loại và hồi quy, nhưng chủ yếu nó được ưu tiên để giải quyết các vấn đề Phân loại. Nó là một trình phân loại có cấu trúc cây, trong đó các nút nội bộ đại diện cho các tính năng của tập dữ liệu, các nhánh đại diện cho các quy tắc quyết định và mỗi nút lá đại diện cho kết quả. Trong cây Quyết định, có hai nút, đó là Nút quyết định và Nút lá. Các nút quyết định được sử dụng để đưa ra bất kỳ quyết định nào và có nhiều nhánh, trong khi các nút Lá là đầu ra của các quyết định đó và không chứa bất kỳ nhánh nào nữa.

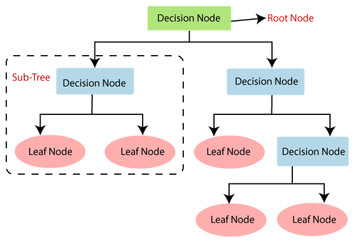
Các quyết định hoặc thử nghiệm được thực hiện trên cơ sở các tính năng của tập dữ liệu nhất định.

Nó là một biểu diễn đồ họa để nhận được tất cả các giải pháp có thể cho một vấn đề / quyết định dựa trên các điều kiện nhất định.

Để xây dựng một cây, ta sử dụng thuật toán CART, viết tắt của thuật toán Classification và Regression Tree.

Cây quyết định chỉ đơn giản là đặt một câu hỏi và dựa trên câu trả lời (Có /Không), nó tiếp tục chia cây thành các cây con.

Sơ đồ dưới đây giải thích cấu trúc chung của cây quyết định:



Hình 1.2 Cấu trúc chung của cây quyết định

#### Thuật toán

Trong một cây quyết định, để dự đoán lớp của tập dữ liệu đã cho, thuật toán bắt đầu từ nút gốc của cây. Thuật toán này so sánh các giá trị của thuộc tính root với thuộc tính bản ghi (tập dữ liệu thực) và dựa trên so sánh, theo dõi nhánh và chuyển đến nút tiếp theo.

Đối với nút tiếp theo, thuật toán lại so sánh giá trị thuộc tính với các nút con khác và di chuyển xa hơn. Nó tiếp tục quá trình cho đến khi nó đạt đến nút lá của cây. Quá trình hoàn chỉnh có thể được hiểu rõ hơn bằng cách sử dụng thuật toán dưới đây:

* **Bước 1:** Bắt đầu cây bằng nút gốc, S, chứa tập dữ liệu hoàn chỉnh cho biết.
* **Bước 2:** Tìm thuộc tính tốt nhất trong tập dữ liệu bằng cách sử dụng **Thước đo lựa chọn thuộc tính (ASM).**
* **Bước 3:** Chia S thành các tập con chứa các giá trị có thể có cho các thuộc tính tốt nhất.
* **Bước 4:** Tạo nút cây quyết định, chứa thuộc tính tốt nhất.
* **Bước 5:** Đệ quy tạo các cây quyết định mới bằng cách sử dụng các tập hợp con của tập dữ liệu được tạo ở bước -3. Tiếp tục quá trình này cho đến khi đạt đến một giai đoạn mà bạn không thể phân loại thêm các nút và được gọi là nút cuối cùng là nút lá.

#### Ưu điểm

* Thật đơn giản để hiểu vì nó tuân theo cùng một quy trình mà con người tuân theo trong khi đưa ra bất kỳ quyết định nào trong cuộc sống thực.
* Nó có thể rất hữu ích để giải quyết các vấn đề liên quan đến quyết định.
* Nó giúp suy nghĩ về tất cả các kết quả có thể cho một vấn đề.
* Có ít yêu cầu làm sạch dữ liệu hơn so với các thuật toán khác.

#### Nhược điểm

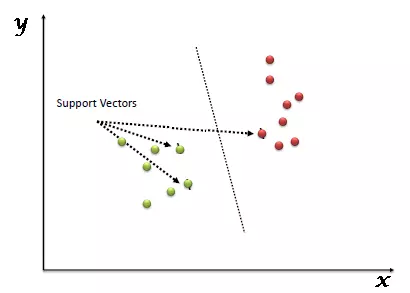
* Cây quyết định chứa rất nhiều lớp, làm cho nó phức tạp.
* Nó có thể có một vấn đề overfitting, có thể được giải quyết bằng cách sử dụng thuật toán Random Forest.
* Đối với nhiều nhãn lớp hơn, độ phức tạp tính toán của cây quyết định có thể tăng lên.

### Support Vector Machine

#### Định nghĩa

Support Vector Machine là một thuật toán giám sát, nó có thể sử dụng cho cả việc phân loại hoặc đệ quy. Tuy nhiên nó được sử dụng chủ yếu cho việc phân loại. Trong thuật toán này, chúng ta vẽ đồi thị dữ liệu là các điểm trong n chiều ( ở đây n là số lượng các tính năng bạn có) với giá trị của mỗi tính năng sẽ là một phần liên kết. Sau đó chúng ta thực hiện tìm "đường bay" (hyper-plane) phân chia các lớp. Hyper-plane nó chỉ hiểu đơn giản là 1 đường thẳng có thể phân chia các lớp ra thành hai phần riêng biệt.

Support Vectors hiểu một cách đơn giản là các đối tượng trên đồ thị tọa độ quan sát, Support Vector Machine là một biên giới để chia hai lớp tốt nhất.



Hình 1.3 Minh họa thuật toán Support Vector Machine

#### Ưu điểm

* Xử lý trên không gian số chiều cao: SVM là một công cụ tính toán hiệu quả trong không gian chiều cao, trong đó đặc biệt áp dụng cho các bài toán phân loại văn bản và phân tích quan điểm nơi chiều có thể cực kỳ lớn.
* Tiết kiệm bộ nhớ: Do chỉ có một tập hợp con của các điểm được sử dụng trong quá trình huấn luyện và ra quyết định thực tế cho các điểm dữ liệu mới nên chỉ có những điểm cần thiết mới được lưu trữ trong bộ nhớ khi ra quyết định.
* Tính linh hoạt - phân lớp thường là phi tuyến tính. Khả năng áp dụng Kernel mới cho phép linh động giữa các phương pháp tuyến tính và phi tuyến tính từ đó khiến cho hiệu suất phân loại lớn hơn.

#### Nhược điểm

* Bài toán số chiều cao: Trong trường hợp số lượng thuộc tính (p) của tập dữ liệu lớn hơn rất nhiều so với số lượng dữ liệu (n) thì SVM cho kết quả khá tồi.
* Chưa thể hiện rõ tính xác suất: Việc phân lớp của SVM chỉ là việc cố gắng tách các đối tượng vào hai lớp được phân tách bởi siêu phẳng SVM. Điều này chưa giải thích được xác suất xuất hiện của một thành viên trong một nhóm là như thế nào. Tuy nhiên hiệu quả của việc phân lớp có thể được xác định dựa vào khái niệm margin từ điểm dữ liệu mới đến siêu phẳng phân lớp mà chúng ta đã bàn luận ở trên.

### Random Forest

#### Định nghĩa

Random là ngẫu nhiên, Forest là rừng, nên ở thuật toán Random Forest mình sẽ xây dựng nhiều cây quyết định bằng thuật toán Decision Tree, tuy nhiên mỗi cây quyết định sẽ khác nhau (có yếu tố random). Sau đó kết quả dự đoán được tổng hợp từ các cây quyết định. Ở bước huấn luyện thì mình sẽ xây dựng nhiều cây quyết định, các cây quyết định có thể khác nhau (phần sau mình sẽ nói mỗi cây được xây dựng như thế nào).

Sau đó ở bước dự đoán, với một dữ liệu mới, thì ở mỗi cây quyết định mình sẽ đi từ trên xuống theo các node điều kiện để được các dự đoán, sau đó kết quả cuối cùng được tổng hợp từ kết quả của các cây quyết định.

#### Cách xây dựng thuật toán

Giả sử bộ dữ liệu của mình có n dữ liệu (sample) và mỗi dữ liệu có d thuộc tính (feature). Để xây dựng mỗi cây quyết định mình sẽ làm như sau:

* Lấy ngẫu nhiên n dữ liệu từ bộ dữ liệu với kĩ thuật Bootstrapping, hay còn gọi là random sampling with replacement. Tức khi mình sample được 1 dữ liệu thì mình không bỏ dữ liệu đấy ra mà vẫn giữ lại trong tập dữ liệu ban đầu, rồi tiếp tục sample cho tới khi sample đủ n dữ liệu. Khi dùng kĩ thuật này thì tập n dữ liệu mới của mình có thể có những dữ liệu bị trùng nhau.
* Sau khi sample được n dữ liệu từ bước 1 thì mình chọn ngẫu nhiên ở k thuộc tính (k < n). Giờ mình được bộ dữ liệu mới gồm n dữ liệu và mỗi dữ liệu có k thuộc tính.
* Dùng thuật toán Decision Tree để xây dựng cây quyết định với bộ dữ liệu ở bước 2.

Do quá trính xây dựng mỗi cây quyết định đều có yếu tố ngẫu nhiên (random) nên kết quả là các cây quyết định trong thuật toán Random Forest có thể khác nhau. Thuật toán Random Forest sẽ bao gồm nhiều cây quyết định, mỗi cây được xây dựng dùng thuật toán Decision Tree trên tập dữ liệu khác nhau và dùng tập thuộc tính khác nhau. Sau đó kết quả dự đoán của thuật toán Random Forest sẽ được tổng hợp từ các cây quyết định. Khi dùng thuật toán Random Forest, mình hay để ý các thuộc tính như: số lượng cây quyết định sẽ xây dựng, số lượng thuộc tính dùng để xây dựng cây. Ngoài ra, vẫn có các thuộc tính của thuật toán Decision Tree để xây dựng cây như độ sâu tối đa, số phần tử tối thiểu trong 1 node để có thể tách.

### Grid Search CV

#### Định nghĩa

GridSearchCV là một kỹ thuật để tìm kiếm thông qua các giá trị tham số tốt nhất từ tập hợp lưới tham số nhất định. Về cơ bản nó là một phương pháp xác nhận chéo. mô hình và các tham số bắt buộc phải được cung cấp. Các giá trị tham số tốt nhất được trích xuất và sau đó đưa ra các dự đoán.

#### Hoạt động của GridSearchCV

GridSearchCV là một thư viện học máy cho python. Chúng tôi có một tìm kiếm đầy đủ về các giá trị tham số được chỉ định cho một công cụ ước tính. Đối tượng ước tính về cơ bản cần cung cấp một hàm điểm hoặc bất kỳ loại điểm nào phải được thông qua. Có 2 phương pháp chính có thể được triển khai trên GridSearchcv, chúng phù hợp và dự đoán. Ngoài ra còn có các dự đoán\_công việc khác, chức năng quyết định, v.v ... Nhưng hai cách được đề cập thường xuyên được sử dụng. Theo loại thuật toán được sử dụng cho tập dữ liệu để phân tích, nó có các tham số khác nhau của riêng nó. Người dùng cần cung cấp một bộ giá trị khác cho các tham số quan trọng. Gridsearchcv bằng cách xác nhận chéo sẽ tìm ra giá trị tốt nhất cho các tham số được đề cập. Có các giá trị mặc định được đặt cho các tham số cũng có thể được xem xét.

### Naïve Bayes

#### Định nghĩa

Naïve Bayes là một thuật toán học máy xác suất dựa trên Định lý Bayes, được sử dụng trong nhiều nhiệm vụ phân loại.

#### Naive Bayes Classifier

Xét bài toán classification với C classes 1,2,…,C Giả sử có một điểm dữ liệu x∈Rd. Hãy tính xác suất để điểm dữ liệu này rơi vào class c. Nói cách khác, hãy tính:

*p(y=c|x) (1)*

hoặc viết gọn thành p(c|x).

Tức tính xác suất để đầu ra là class *c* biết rằng đầu vào là vector x.

Biểu thức này, nếu tính được, sẽ giúp chúng ta xác định được xác suất để điểm dữ liệu rơi vào mỗi class. Từ đó có thể giúp xác định class của điểm dữ liệu đó bằng cách chọn ra class có xác suất cao nhất:

Biểu thức (2) thường khó được tính trực tiếp. Thay vào đó, quy tắc Bayes thường được sử dụng:

*c=arg⁡max p(c|x)(3) =arg⁡max p(x|c)p(c)p(x) (4) =arg⁡max p(x|c)p(c)(5)*

Từ (3) sang (4) là vì quy tắc Bayes. Từ (4) sang (5) là vì mẫu số *p(x)* không phụ thuộc vào c.

Tiếp tục xét biểu thức (5), p(c) có thể được hiểu là xác suất để một điểm rơi vào class c. Giá trị này có thể được tính bằng MLE, tức tỉ lệ số điểm dữ liệu trong tập training rơi vào class này chia cho tổng số lượng dữ liệu trong tập training; hoặc cũng có thể được đánh giá bằng MAP estimation. Trường hợp thứ nhất thường được sử dụng nhiều hơn.

Thành phần còn lại p(x|c) tức phân phối của các điểm dữ liệu trong class c*c*, thường rất khó tính toán vì x*x* là một biến ngẫu nhiên nhiều chiều, cần rất rất nhiều dữ liệu training để có thể xây dựng được phân phối đó. Để giúp cho việc tính toán được đơn giản, người ta thường giả sử một cách đơn giản nhất rằng các thành phần của biến ngẫu nhiên x*x* là độc lập với nhau, nếu biết c . Tức là:

*p(x|c)=p(x1,x2,…,xd|c)=p(xi|c) (6)*

Giả thiết các chiều của dữ liệu độc lập với nhau, nếu biết c*c*, là quá chặt và ít khi tìm được dữ liệu mà các thành phần hoàn toàn độc lập với nhau. Tuy nhiên, giả thiết *ngây ngô* này lại mang lại những kết quả tốt bất ngờ. Giả thiết về sự độc lập của các chiều dữ liệu này được gọi là *Naive Bayes* . Cách xác định class của dữ liệu dựa trên giả thiết này có tên là *Naive Bayes Classifier (NBC)*.

NBC, nhờ vào tính đơn giản một cách *ngây thơ*, có tốc độ training và test rất nhanh. Việc này giúp nó mang lại hiệu quả cao trong các bài toán large-scale.

Ở bước **training**, các phân phối p(c) và p(xi|c),i=1,…,d sẽ được xác định dựa vào training data. Việc xác định các giá trị này có thể dựa vào Maximum Likelihood Estimation hoặc Maximum A Posteriori.

Ở bước **test**, với một điểm dữ liệu mới x, class của nó sẽ được xác đinh bởi:

c=arg max p(c)p(xi|c) (7)

Khi d*d* lớn và các xác suất nhỏ, biểu thức ở vế phải của (7) sẽ là một số rất nhỏ, khi tính toán có thể gặp sai số. Để giải quyết việc này, (7) thường được viết lại dưới dạng tương phải:

*c= arg⁡max =log⁡(p(c))+log⁡(p(xi|c)) (7.1)*

Việc này không ảnh hưởng tới kết quả vì log là một hàm đồng biến trên tập các số dương.

Mặc dù giả thiết mà Naive Bayes Classifiers sử dụng là quá phi thực tế, chúng vẫn hoạt động khá hiệu quả trong nhiều bài toán thực tế, đặc biệt là trong các bài toán phân loại văn bản, ví dụ như lọc tin nhắn rác hay lọc email spam. Trong phần sau của bài viết, chúng ta cùng xây dựng một bộ lọc email spam tiếng Anh đơn giản.

Cả việc training và test của NBC là cực kỳ nhanh khi so với các phương pháp classification phức tạp khác. Việc giả sử các thành phần trong dữ liệu là độc lập với nhau, nếu biết class, khiến cho việc tính toán mỗi phân phối p(xi|c)*p(xi|c)* trở nên cực kỳ nhanh.

Mỗi giá trị p(c),c=1,2,…,C có thể được xác định như là tần suất xuất hiện của class c*c* trong training data.

Việc tính toán *p(xi|c)* phụ thuộc vào loại dữ liệu. Có ba loại được sử dụng phổ biến là: Gaussian Naive Bayes, Multinomial Naive Bayes, và Bernoulli Naive .

## Kiến thức khác

### Precision – Recall – F1 Score

Precision lý tưởng nên là 1 (cao) đối với một bộ phân loại tốt. Precision chỉ trở thành 1 khi tử số và mẫu số bằng nhau, tức là TP = TP + FP , điều này cũng có nghĩa là FP bằng không. Khi FP tăng giá trị của mẫu số trở nên lớn hơn tử số và giá trị độ chính xác giảm (điều mà chúng ta không muốn).

Recall còn được gọi là độ nhạy hoặc tỷ lệ dương tính thực sự và được định nghĩa như sau:

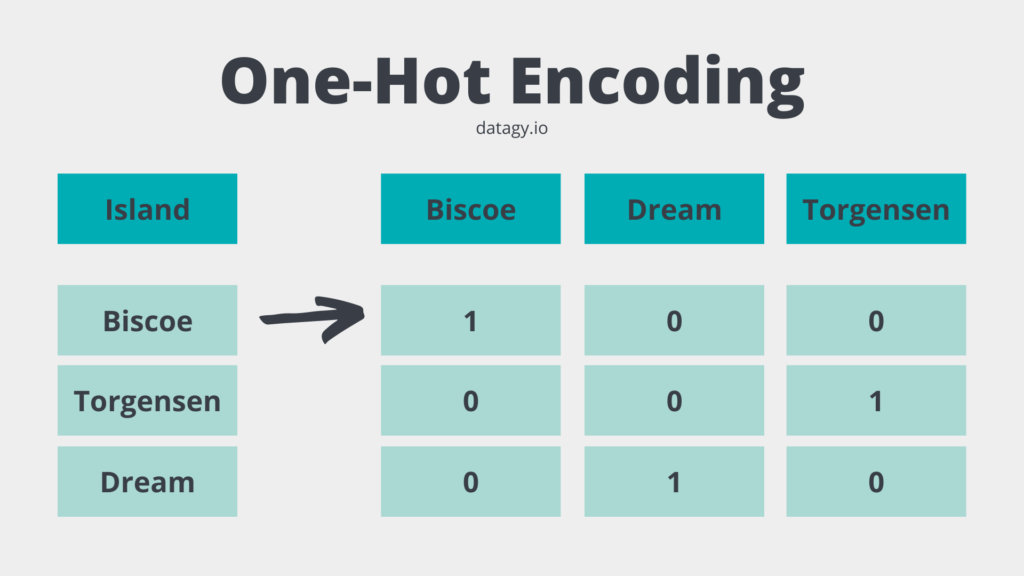
Recall lý tưởng nên là 1 (cao) cho một bộ phân loại tốt. Gọi lại chỉ trở thành 1 khi tử số và mẫu số bằng nhau tức là TP = TP + FN , điều này cũng có nghĩa là FN bằng không. Khi FN tăng, giá trị của mẫu số trở nên lớn hơn tử số và giá trị thu hồi giảm (điều mà chúng ta không muốn).

Vì vậy, lý tưởng nhất là trong một bộ phân loại tốt, chúng ta muốn cả Precision và Recall là 1, điều này cũng có nghĩa là FP và FN bằng 0. Do đó, chúng tôi cần một số liệu có tính đến cả Precision và Recall. Điểm F1 là một số liệu có tính đến cả Precision và Recall và được định nghĩa như sau:

Điểm F1 chỉ trở thành 1 khi Precision và Recall đều là 1. Điểm F1 chỉ trở nên cao khi cả Precision và Recall  đều cao. Điểm F1 là trung bình hài hòa của Precision và Recall và là một thước đo tốt hơn Accuracy.

### One – hot encoder

Mã hóa One - hot là quá trình mà dữ liệu phân loại được chuyển đổi thành dữ liệu số để sử dụng trong học máy. Các đối tượng phân loại được chuyển thành các đối tượng địa lý nhị phân được mã hóa "một lần", có nghĩa là nếu đối tượng địa lý được đại diện bởi cột đó, nó sẽ nhận được a 1. Nếu không, nó nhận được a 0.



Hình 1.4 Minh họa kỹ thuật mã hóa One – Hot

Mã hóa One – hot quan trọng với học máy vì các thuật toán học máy giả định (và yêu cầu) dữ liệu là số, dữ liệu phân loại phải được xử lý trước để được chấp nhận.

### Chuẩn hóa dữ liệu

Scaling là biến đổi khoảng giá trị của dữ liệu về một khoảng đặc biệt như 0-100 hay 0-1, thường là 0-1. Trong một số thuật toán Machine Learning mà khoảng cách giữa các điểm dữ liệu là quan trong, như SVM hay KNN, thì việc scale dữ liệu là vô cùng quan trọng, vì mỗi thay đổi nhỏ của dữ liệu cũng mang đến kết quả khó đoán trước.

Giả sử trong dữ liệu về cân nặng 1 người, có 2 loại dữ liệu là kg và pound. Khi chúng ta sử dụng trực tiếp những giá trị này với SVM hoặc KNN, thuật toán sẽ hiểu sự 1kg = 1pound, nhưng điều đó không đúng trong thực tế (1pound = 0.45359237kg). Qua ví dụ trên, chắc mọi người cũng đã hiểu được tầm quan trọng của việc scale dữ liệu.

Scale dữ liệu cần thực hiện trên cùng mục đích sử dụng của thước đo (tiền - tiền, chiều cao - chiều cao,...), chứ hiển nhiên ta ko thể scale cân nặng với chiều cao, vì giữa chúng cơ bản không có sự liên quan mật thiết.

### Kỹ thuật lấy quá mức cho dữ liệu nhãn thiểu số

SMOTE là một thuật toán khác để lấy mẫu các lớp nhỏ hơn. Ý tưởng chính đằng sau SMOTE là các cá thể đã tạo nên được xây dựng từ các quan sát có sẵn, nhưng không được giống hệt nhau. SMOTE tạo các phiên bản mới của một lớp thiểu số theo thuật toán sau:

* Các khu vực đường viền được ước lượng gần đúng bằng SVM sau khi đào tạo bộ phân loại SVM trên tập dữ liệu đào tạo ban đầu. Sau khi được tính toán, các mẫu được tổng hợp bên cạnh ranh giới gần đúng.
* Chọn một cặp điểm ngẫu nhiên từ cùng một lớp thiểu số, x\_i và x\_j, trong đó điểm sau được lấy mẫu từ k- láng giềng của x\_i.
* Lấy mẫu đồng nhất một giá trị t từ phân phối đều (U [0, 1]).
* Tạo một phiên bản mới x = t \* x\_i + (1-t) \* x\_j.
* Lặp lại cho đến khi đủ số lượng mẫu theo ngưỡng tỷ lệ lấy mẫu quá mức (siêu tham số).

SMOTE đã cho thấy sự sử dụng rộng rãi và thành công lớn trong các ứng dụng và tác vụ khác nhau. Nó vẫn là một trong những cơ chế lấy mẫu quá mức phổ biến nhất và đã dẫn đến một nhóm lớn các biến thể, mỗi biến thể đều có điểm mạnh và điểm hạn chế riêng. Tuy nhiên, cuối cùng thì không có biến thể nào của SMOTE luôn được chứng minh là có cải thiện về sức mạnh và hiệu suất của nó.

### Phương pháp SMOTE Edited Nearest Neighbor

Được phát triển bởi Batista và cộng sự (2004), phương pháp này kết hợp khả năng SMOTE để tạo ra các ví dụ tổng hợp cho lớp thiểu số và khả năng ENN để xóa một số quan sát từ cả hai lớp được xác định là có lớp khác nhau giữa lớp của quan sát và phần lớn hàng xóm gần nhất K của nó lớp. Quá trình SMOTE-ENN có thể được giải thích như sau.

* (Bắt đầu SMOTE) Chọn dữ liệu ngẫu nhiên từ lớp thiểu số.
* Tính khoảng cách giữa dữ liệu ngẫu nhiên và k lân cận gần nhất của nó.
* Nhân sự khác biệt với một số ngẫu nhiên từ 0 đến 1, sau đó thêm kết quả vào lớp thiểu số dưới dạng mẫu tổng hợp.
* Lặp lại bước số 2–3 cho đến khi đáp ứng được tỷ lệ nhóm thiểu số mong muốn. ( Hết SMOTE )
* (Bắt đầu của ENN) Xác định K, là số láng giềng gần nhất. Nếu không xác định được thì K = 3.
* Tìm K-láng giềng gần nhất của quan sát trong số các quan sát khác trong tập dữ liệu, sau đó trả về lớp đa số từ K-láng giềng gần nhất.
* Nếu lớp của quan sát và lớp đa số từ láng giềng gần nhất K của quan sát là khác nhau, thì quan sát và láng giềng gần nhất K của nó sẽ bị xóa khỏi tập dữ liệu.
* Lặp lại bước 2 và 3 cho đến khi tỷ lệ mong muốn của mỗi lớp được đáp ứng. (Hết ENN)

# PREDICT BOSTON HOUSING PRICES

## Nhập thư viện và load dữ liệu

### Nhập thư viện

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  %matplotlib inline |

### Load dữ liệu

|  |
| --- |
| from sklearn.datasets import load\_boston  x = load\_boston().data  y = load\_boston().target  # Split it into train and test subsets.  from sklearn.model\_selection import train\_test\_split  train\_X, test\_X, train\_y, test\_y = train\_test\_split(x, y, test\_size = 0.2, random\_state = 23) |

Tải dữ liệu load\_boston bằng thư viện sklearn.datasets, gán vào x và y. Tiếp tục chia thành phần dữ liệu để huấn luyện và dữ liệu để test thông qua lệnh train\_test\_split trong thư viện sklearn.model\_selection.

### Lấy dữ liệu theo yêu cầu

|  |
| --- |
| x = load\_boston().data[:,[2,5]]  y = load\_boston().target  # Split it into train and test subsets.  from sklearn.model\_selection import train\_test\_split  train\_X, test\_X, train\_y, test\_y = train\_test\_split(x, y, test\_size = 0.2, random\_state = 23) |

Để đơn giản, dữ liệu lấy vào gồm 2 đặc tính: INDUS và RM, hay đặc tính thứ hai và thứ năm trong tập data load\_boston.

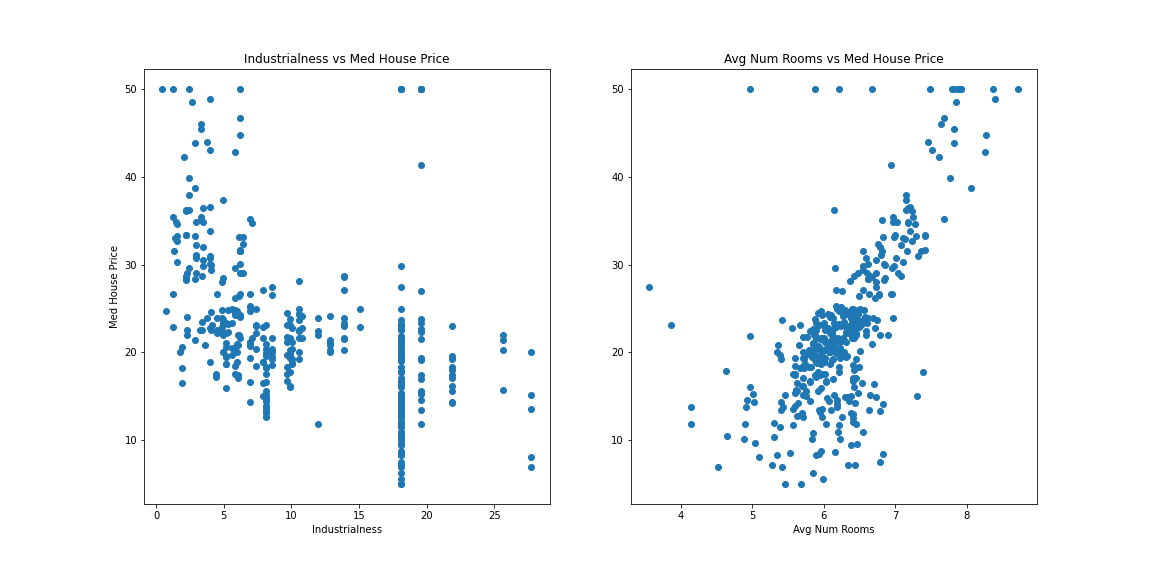
## Minh họa dữ liệu

Để hiểu thêm về tập dữ liệu ta có thể làm thêm một số minh họa như sau:

### Code

|  |
| --- |
| fig = plt.figure()  fig1 = fig.add\_subplot(121)  fig2 = fig.add\_subplot(122)  fig1.scatter(train\_X[:, 0], train\_y)  fig1.set\_title("Industrialness vs Med House Price")  fig1.set\_xlabel("Industrialness")  fig1.set\_ylabel("Med House Price")  fig2.scatter(train\_X[:,1], train\_y)  fig2.set\_title("Avg Num Rooms vs Med House Price")  fig2.set\_xlabel("Avg Num Rooms")  #fig2.set\_ylabel("Med House Price") |

### Kết quả



Hình 2.1 Minh họa cho 2 đặc tính INDUS và RM

## Viết hàm Cost function

### Cost function thông thường

Khai triển ta được:

|  |
| --- |
| def np\_cost(w, b, X, y):      '''      Evaluate the cost function in a non-vectorized manner for      inputs `X` and targets `y`, at weights `w` and `b`.        X: 2D array dataset that has (m, n) dimension.      y: list of targets that has (n, ) dimension.      w: list of weights that has (n, ) dimension      b: a scalar bias.        Return a scalar cost value of `w`, `b`.      '''        cost = 0      m = X.shape[0]      for i in range(m):          y\_hat = w[0]\*X[i, 0] + w[1]\*X[i, 1] + b # hypothesis          cost += (y\_hat - y[i])\*\*2 # sum cost of each data point.        return cost/(2\*m) |
| np\_cost([3, 5], 20, train\_X, train\_y) # 2198.1440481435648  np\_cost([3, 5], 0, train\_X, train\_y) # 1163.1688006188117 |

### Cost function được vector hóa

|  |
| --- |
| def np\_cost\_vectorized(w, b, X, y):      '''      Evaluate the cost function in a vectorized manner for      inputs `X` and targets `t`, at weights `w` and `b`.        X: dataset matrix has (m, n) dimension.      y: targets vector has (n, ) dimension.      w: weights vector has (n, ) dimension      b: a scalar bias.        Return a scalar cost value of `w`, `b`.      '''        m = X.shape[0] # number of samples in dataset      w = np.array(w) # convert to numpy array      y\_hat = np.dot(X, w) + b # hypothesis        return np.sum((y\_hat - y)\*\*2)/(2\*m) |
| np\_cost\_vectorized([3, 5], 20, train\_X, train\_y) # 2198.1440481435643  np\_cost\_vectorized([3, 5], 0, train\_X, train\_y) # 1163.168800618812 |

## Viết hàm Gradient Descent

### Gradient Descent thông thường

|  |
| --- |
| def np\_grad\_fn(w, X, y):      '''      Given `w` - a current "Guess" of what our weights should be            `X` - matrix of shape (m, n + 1) of input features            `y` - target y values      Return gradient of each weight evaluated at the current value      '''        #TODO: Complete the below followed the above expressions      grad\_w = np.zeros(np.array(w).shape)      m = X.shape[0]      y\_hat = np.dot(X, w)      for j in range(grad\_w.shape[0]):          for i in range(m):              grad\_w[j] += (y\_hat[i] - y[i])\*X[i, j]/m      return grad\_w |
| w = [0.1, 0.2, 0.3]  one\_column = np.ones((train\_X.shape[0], 1))  one\_column.shape  train\_X\_new = np.append(one\_column, train\_X, axis = 1) # Add bias  np\_grad\_fn(w, train\_X\_new, train\_y)  # array([ -17.94781361, -161.77719027, -116.62229795]) |

### Gradient Descent vector hóa

|  |
| --- |
| np\_grad\_fn\_vectorized(w, X, y):      '''      Given `w` - a current "Guess" of what our weights should be            `X` - matrix of shape (m, n + 1) of input features            `y` - target y values      Return gradient of each weight evaluated at the current value      '''        #TODO: Complete the below followed the above expressions      m, n = X.shape      y\_hat = np.dot(X, w)      grad\_w = np.dot(X.T, y\_hat - y)/m        return grad\_w |
| w = [0.1, 0.2, 0.3]  np\_grad\_fn\_vectorized(w, train\_X\_new, train\_y)  # array([ -17.94781361, -161.77719027, -116.62229795]) |

## Tối ưu tham số theta

### Viết hàm tối ưu

|  |
| --- |
| def np\_solve\_via\_gradient\_descent(X, y, print\_every = 5000,                                    niter = 100000, alpha = 0.005):      '''      Given `X` - matrix of shape (m, n+1) of input features            `y` - target y values      Solves for linear regression weights.      Return weights after `niter` iterations.      '''      m, n = X.shape      # initialize all the weights to zeros      w = np.zeros((n,))      for k in range(niter):            dw = np\_grad\_fn\_vectorized(w, X, y)          w = w - alpha\*dw            if k % print\_every == 0:              print('Weight after %d iteration: %s' % (k, str(w)))      return w |

### Thử nghiệm và chạy kết quả

|  |
| --- |
| opt\_w = np\_solve\_via\_gradient\_descent(train\_X\_new, train\_y, niter = 500000)  opt\_w # array([-22.41842742,  -0.3374834 ,   7.71503255])  print("Training cost:", np\_cost\_vectorized(opt\_w, train\_X\_new, train\_y)) # Training cost: 18.90049122519828  print("Testing cost:", np\_cost\_vectorized(opt\_w, test\_X\_new, test\_y)) # Testing cost: 22.371460612916348 |

**Nhận xét:** Hàm cost training cũng như hàm testing vẫn còn lớn. Cần tăng số tính năng hay độ phức tạp của mô hình hồi quy tuyến tính.

## Tối ưu hàm cost bài toán đặt ra

### Lựa chọn tính năng

Số lượng tính năng tối đa có thể có là 10. Sử dụng thư viện sklearn.feature\_selection với class SelectKBest.

|  |
| --- |
| # pearson's correlation feature selection for numeric input and numeric output  from sklearn.datasets import make\_regression  from sklearn.feature\_selection import SelectKBest  from sklearn.feature\_selection import f\_regression  # define feature selection  fs = SelectKBest(score\_func = f\_regression, k = 10)  # apply feature selection  x\_selected = fs.fit\_transform(x, y)  print(x\_selected.shape)  # (506, 10) |

### Chuẩn hóa dữ liệu

Chuẩn hóa dữ liệu nhằm tăng độ chính xác cho mô hình, tránh rơi vào cục bộ địa phương và tăng tốc độ training mô hình. Sử dụng thư viện sklearn.preprocessing với Class MinMaxScaler.

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler  scaler = MinMaxScaler()  x = scaler.fit\_transform(x\_selected) |

### Xây dựng mô hình

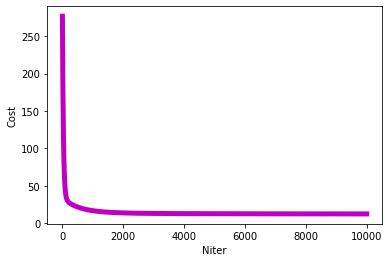
Tương tự mục 2.3, 2.4 và 2.5.

### Kiểm tra quá trình training

#### Code

|  |
| --- |
| n\_niter = []  jplot = []  count = 0  for i in range(len(J\_all)):      jplot.append(J\_all[i])      n\_niter.append(count)      count += 1  jplot = np.array(jplot)  n\_niter = np.array(n\_niter)  plot\_cost(jplot, n\_niter) |

#### Kết quả



Hình 2.2 Hàm cost qua 10000 lần gradient descent

Hình 2.2 thể hiện giá trị hàm cost qua 10000 lần thực hiện gradient descent. Giá trị hàm cost vẫn chưa thể tối ưu về 0.

#### Thực nghiệm

|  |
| --- |
| print("Training cost:", np\_cost\_vectorized(opt\_w, train\_X\_new, train\_y)) # Training cost: 12.449672904641005  print("Testing cost:", np\_cost\_vectorized(opt\_w, test\_X\_new, test\_y)) # Testing cost: 14.910843981138196 |

Hàm cost đã được tối ưu hóa hơn so với phần chỉ training với 2 feature.

# PREDICT DROPOUT OR ACADEMIC SUCCESS

## Thử nghiệm với các mô hình truyền thống

Áp dụng với 7 mô hình truyền thống, ta được bảng sau:

Bảng 3.1 Bảng thống kê độ chính xác training của 7 mô hình truyền thống

|  |  |  |  |
| --- | --- | --- | --- |
| Mô hình | Code | Độ chính xác | F1 Score |
| LogiscticRegression | from sklearn.linear\_model import LogisticRegression  logistic\_model = LogisticRegression(max\_iter = 10000) | 0.7796 | 0.81  0.4  0.86 |
| K-Nearest Neighbor | from sklearn.neighbors import KNeighborsClassifier  knn1\_model = KNeighborsClassifier(n\_neighbors = 17) | 0.6203 | 0.58  0.19  0.73 |
| Decision Tree | from sklearn.tree import DecisionTreeClassifier  dtree\_model = DecisionTreeClassifier() | 0.6779 | 0.68  0.35  0.79 |
| Random Forest | from sklearn.ensemble import RandomForestClassifier  rforest\_model = RandomForestClassifier(n\_estimators = 300) | 0.7876 | 0.81  0.44  0.86 |
| Suport Vector Machine | from sklearn.svm import SVC  svm\_model = SVC() | 0.5017 | 0.00  0.00  0.67 |
| Grid Search CV | from sklearn.model\_selection import GridSearchCV  param\_grid = {'C':[0.1, 1, 10, 100, 1000], 'gamma':[1, 0.1, 0.01, 0.001, 0.0001]}  grid\_model = GridSearchCV(SVC(), param\_grid, verbose = 3) | 0.7163 | 0.71  0.39  0.82 |
| Naïve Bayes | from sklearn.naive\_bayes import GaussianNB  gnb\_model = GaussianNB() | 0.6802 | 0.73  0.21  0.76 |

Từ Bảng 3.1, nhận xét rằng với tập dữ liệu đã cho, mô hình Random Forest cho kết quả tốt nhất, theo sau đó là mô hình Logistic Regression và Grid Search CV. Mô hình KNN, Decision Tree và Naïve Bayes không phù hợp với bài toán. Mô hình SVM không thể áp dụng lên bài toán phân loại này.

Mặc dù cho kết quả tốt nhất (0.78), mô hình Random Forest vẫn có độ chính xác khá thấp và cần cải thiện bằng mạng Neural Network.

## Build mô hình mạng Neural

### One – Hot Encoder

Sử dụng kĩ thuật One – Hot encoder để biến đổi các nhãn là số nguyên sang vector.

|  |
| --- |
| # encode class values as integers  encoder = LabelEncoder()  encoder.fit(y)  encoded\_Y = encoder.transform(y)  # convert integers to dummy variables (i.e. one hot encoded)  dummy\_y = np\_utils.to\_categorical(encoded\_Y) |

Sau khi mã hóa, được kết quả:

* Graduate: 001
* Dropout: 100
* Enrolled: 010

### Xây dựng mạng neural network

Xây dựng mạng Neural Network với 3 lớp, lớp đầu vào với 36 nút tương ứng với 36 tính năng, lớp ẩn với 12 nút và lớp đầu ra với 3 nút tương ứng với 3 nhãn đã được mã hóa One – hot như trên.

|  |
| --- |
| #Dependencies  import keras  from keras.models import Sequential  from keras.layers import Dense, Dropout  # Neural network  model = Sequential()  model.add(Dense(units = 36, activation='relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 12, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 3, activation = 'softmax'))  model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy']) |

Tham số loss của hàm compile là categorical\_crossentropy phù hợp với bài toán phân loại đa lớp. Tham số metrics là accuracy nhằm xem xét chất lượng training model dựa trên thông số này.

### Training model

Training model qua 600 epochs với tập đối chiếu là (X\_test, y\_test).

|  |
| --- |
| history = model.fit(X\_train, y\_train, epochs = 600, validation\_data = (X\_test, y\_test)) |

Quá trình training model khoảng 3 phút, như sau:

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình 3.1 Quá trình training model

### Đánh giá model

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 3.2 Đánh giá model mạng neural network

Hình 3.2 thể hiện hàm loss cũng như độ chính xác trong quá trình training model qua 600 epochs. Hàm loss training có xu hướng về 0 tuy nhiên hàm los của tệp validation có xu hướng tăng lên 0.6, không đáng kể. Giá trị của accuracy giữa tệp data training và tệp validation tương đương nhau, xấp xỉ 0.81.

**Kết luận:** Mô hình đã tốt hơn trước với độ chính xác dự đoán cao hơn. Tuy nhiên, giá trị này vẫn tương đối thấp, có thể cải thiện thêm.

## Tiền xử lý tập data & Cải thiện mô hình Neural Network

Xem xét tập dữ liệu:

* Graduate: 2209 mẫu
* Dropout: 1421 mẫu
* Enrolled: 794 mẫu

Tập dữ liệu tương đối mất cân bằng. Cần có một số phương pháp tiền xử lý dữ liệu để cân bằng tập.

### Kỹ thuật lấy mẫu quá mức cho dữ liệu nhãn thiểu số

#### Triển khai

|  |
| --- |
| from imblearn.over\_sampling import SMOTE  smote = SMOTE(k\_neighbors = 15)  X, y = smote.fit\_resample(X, y) |

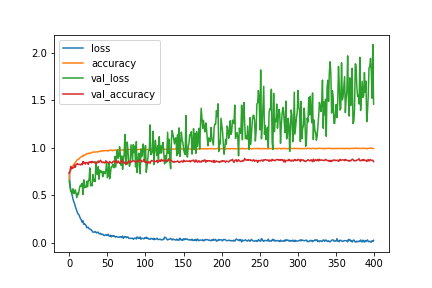
Lúc này, tập data tăng từ 4424 mẫu lên 6627 mẫu với số mẫu mỗi nhãn là:

* Graduate: 2209 mẫu
* Dropout: 2209 mẫu
* Enrolled: 2209 mẫu

Đồng thời xây dựng lại mô hình Neural Network với số lớp và số nút mỗi lớp lớn hơn:

|  |
| --- |
| # Neural network  model = Sequential()  model.add(Dense(units = 512, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 512, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 512, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 512, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 512, activation = 'relu'))  model.add(Dropout(0.2))  model.add(Dense(units = 3, activation = 'softmax'))  model.compile(loss = 'categorical\_crossentropy', optimizer = 'adam', metrics = ['accuracy']) |

#### Kết quả



Hình 3.3 Theo dõi hàm loss và tính chính xác qua 400 epochs của mô hình

Hình 3.3 đánh giá hàm loss và accuracy qua 400 lần fit dữ liệu của mô hình. Nhận thấy hàm loss training có xu hướng về 0 tuy nhiên hàm loss qua tệp đối chiếu có xu hướng tăng. Độ chính xác training đã tiến sát tới 1, độ chính xác của tệp validation ở khoảng 0.86.

**Kết luận:** Mô hình đã tốt hơn trước với độ chính xác dự đoán cao hơn. Tuy nhiên, giá trị này vẫn có thể cải thiện thêm.

### Kỹ thuật SMOTE-ENN cho dữ liệu nhãn thiểu số

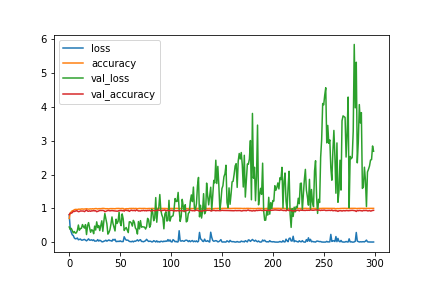
#### Triển khai

|  |
| --- |
| from imblearn.combine import SMOTEENN  import math  bordersmote = SMOTEENN()  X, y = SMOTEENN().fit\_resample(X, y) |

Lúc này, tập data giảm từ 4424 mẫu xuống 3086 mẫu với số mẫu mỗi nhãn là:

* Graduate: 739 mẫu
* Dropout: 1020 mẫu
* Enrolled: 1416 mẫu

#### Kết quả



Hình 3.4 Theo dõi hàm loss và độ chính xác mô hình qua 300 epochs

Hình 3.4 đánh giá hàm loss và accuracy qua 300 lần fit dữ liệu của mô hình. Hàm loss training tiến gần tới 0, loss của tệp validation biến động quanh 1 – 2. Độ chính xác của tệp test tiến tới 0.94.

**Kết luận:** Mô hình đạt đến sự chính xác cao, xấp xỉ 94%.

KẾT LUẬN

Bài tập lớn của nhóm đã hoàn thành được các mục tiêu đề ra: Hiểu thêm về cách hoạt động của Numpy và Tensorflow; Viết và debug thuật toán với Tensorflow và Numpy; Hiểu thêm về hồi quy tuyến tính; Tối ưu các tham số của mô hình qua bài toán Dự đoán giá nhà tại Boston. Phân tích dữ liệu, xây dựng mô hình và đánh giá mô hình với bài toán Dự đoán sinh viên bỏ học hay hoàn thành khóa học.

Báo cáo đầy đủ các công việc đã làm, chứng minh được công việc đã làm thỏa mãn các yêu cầu của đề bài; Các bước tinh chỉnh mô hình đã được làm chi tiết. Kết quả về độ chính xác của mô hình cũng được tối ưu lên 94%.

Bài toán có thể cải thiện với độ chính xác cao hơn nếu sử dụng các kĩ thuật xử lý về tập dữ liệu như Augmentation (Tăng cường dữ liệu) hay Feature Engineering.

Chúng em xin chân thành cảm ơn thầy Nguyễn Việt Tùng đã tận tình giảng dạy và củng cố các kiến thức của môn học để chúng em có thể hoàn thành bài tập lớn này. Trong quá trình làm Bài tập lớn chắc chắn chúng em sẽ gặp phải các sai sót, kính mong thầy có thể góp ý, chỉnh sửa để nhóm hoàn thiện bài tập và có thêm kinh nghiệm trong các sản phẩm về sau.

TÀI LIỆU THAM KHẢO

<https://builtin.com/artificial-intelligence>

<https://www.ibm.com/cloud/learn/machine-learning>

<https://towardsdatascience.com/imbalanced-classification-in-python-smote-enn-method-db5db06b8d50>

<https://turintech.ai/insights/what-is-imbalanced-data-and-how-to-handle-it/>

<https://viblo.asia/p/scaling-vs-normalization-oOVlYJJz58W>

[https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd#](https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd)