DỰ ĐOÁN SUY TIM DỰA VÀO CÁC PHƯƠNG PHÁP ÁP DỤNG MÁY HỌC

Nguyen Hoang Tuan
20520344
KHNT2020

Nguyen Minh Ly
20521592
KHNT2020

Mai Duy Ngoc
20520654
KHNT2020

Dao Danh Dang Phung
20520699
KHNT2020

Tran Dang Khoa
20520589
KHNT2020

Le Viet Lam Quang
20520290
KHNT2020

Phan Tan Thuong
20522001
KHNT2020

Nguyen Dang Quang Tuan
20522116
KHNT2020

***Abstract*—Máy học ngày nay có rất nhiều ứng dụng trong thực tiễn lấy ví dụ đơn giản như là việc dự đoán xem một người có bị suy tim dựa trên các đặc điểm được cho trước như nhóm lấy làm ví dụ. Việc dự đoán này là kết quả của bài toán Classification, nhưng cho đến nay thì rất nhiều mô hình classification đã được nghiên cứu và phát triển. Chính vì thế bài viết này được tạo ra để có thể giới thiệu một số mô hình classification phổ biến ngày nay và có thể biết rõ được thời điểm thích hợp để sử dụng mô hình cũng như các ưu nhược điểm hiện có của các mô hình này. Trong bài viết này, nhóm sử dụng bộ dữ liệu Heart Failure để có thể chấm điểm mô hình dựa trên các thang điểm: F1, Accuracy, Precision và Recall từ đó có thể đưa ra các đánh giá về các mô hình trên bộ dữ liệu này. Khi đã thực nghiệm trên bộ dữ liệu thì nhóm đã tìm ra được mô hình dự đoán tốt nhất trên bộ dữ liệu được sử dụng là AdaBoost.**

Keywords—Suy tim, máy học, logistic regression, Decision Tree, Random Forest, Naïve Bayes, KNN, SVM, Neural Network, AdaBoosting .

I. GIỚI THIỆU

Cùng với sự phát triển của thế giới, con người ngày càng đòi hỏi nhiều hơn về tính hiệu quả lẫn tài nguyên dành ra cho lĩnh vực công nghệ thông tin. Học máy cũng không ngoại lệ, tuy chỉ mới được phát triển trong một thời gian gần, nhưng lĩnh vực này đã mau chóng trở thành xu hướng được "săn đuổi" nhất trong ngành công nghệ cao hiện nay - một công cụ mạnh mẽ để đưa ra những dự đoán phục vụ con người. Cốt lõi tạo nên giá trị và cách thức vận hành của Học máy đó chính là những mô hình được xây dựng dựa trên các cơ sở toán học mà con người đã nghiên cứu và áp dụng.

Trong phạm vi đề tài này, nhóm sẽ tìm hiểu cách hoạt động, phân tích và đánh giá một số mô hình học máy. Đồng thời, sử dụng bộ dữ liệu Heart Failure mà nhóm đã sưu tầm để cài đặt và rút trích những ưu yếu điểm của các mô hình.

Vậy làm thế nào để chúng ta biết thuật toán nào mạnh thuật toán nào yếu và mạnh yếu trong trường hợp nào. Bạn muốn chọn mua một chiếc balo để nhưng phân vân không biết chọn chiếc nào, bạn sẽ đưa ra các đặc điểm ví dụ như màu sắc, kích thước , kiểu dáng,... và chấm điểm những chiếc balo qua các đặc điểm này. Nếu chiếc balo bạn chọn có nhiều màu sắc đa dạng, kiểu dáng của siêu nhân hoạt hình nó sẽ có điểm rất cao trong mắt các em học sinh tiểu học, nhưng lại rất trẻ trâu trong mắt các bạn sinh viên, nếu chiếc balo bạn chọn là sẫm màu ,dáng gọn gàng thì ngược lại. Đánh giá các thuật toán cũng vậy, trong bài toán này chúng ta sẽ tính các điểm số F1, Accuracy, Precision và Recall trên bộ dữ liệu Heart Failure của từng thuật toán, ở mỗi thuật toán chúng ta sẽ điều chỉnh các tham số sao cho thuật toán có kết quả tốt nhất, chúng ta sẽ so sánh các chỉ số này và ngoài ra so sánh các thuật toán trong một số trường hợp cụ thể để biết được cái nào tốt, xấu trong trường hợp nào.

Input: các chỉ số chuẩn đoán y khoa của một bệnh nhân gồm :tuổi, giới tính, lượng mỡ máu , chế độ tập luyện,…

Output: dự đoán bệnh nhân đó có bị suy tim hay không.

Áp dụng vào thực tiễn: công cụ có áp dụng phương pháp các máy học dùng để hỗ trợ chuẩn đoán sàng lọc ở các bệnh viện.

II. DATASET

Dữ liệu là yếu tố quyết định và bắt buộc trong mọi giả pháp Máy Học. Dataset của chúng tôi được thu thập từ website Kaggle[1]. Dataset được xây dựng bằng cách kết hợp nhiều các nguồn dữ liệu khác sẵn có, với độ uy tín cao, được sử dụng trên thế giới, trong có bao gồm nguồn bộ dữ liệu Heart Disease của UCI.

A. Giới thiệu dataset:

Trước tiên nói về tập dữ liệu Heart Disease của UCI, đây là một bộ dữ liệu được tin dùng trong việc nghiên cứu các vấn đề y học dựa vào các đặc trưng thống kê được thu thập số liệu trực tiếp từ bệnh nhân từ những viện, trung tâm liên quan đến lĩnh vực y học lớn trên thế giới. Tập dữ liệu Heart Disease của UIC đã được xây dựng từ năm 1988, được tổng hợp bởi 4 nguồn cơ sở dữ liệu khác nhau:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.

Ban đầu, bộ dữ liệu của UCI chứa 76 thuộc tính về các chuẩn đoán về y khoa từ các bệnh nhân, tuy nhiên, các nghiên cứu đã công bố chỉ có 14 thuộc tính có liên quan đến việc dự đoán bệnh tim. Do đó, bộ dữ liệu đã được tinh giảm gồm 920 bệnh nhân với 14 thuộc tính chính.

Về bộ dữ liệu nhóm thu thập được trên Kaggle, bộ dữ liệu này được tạo bằng cách kết hợp các bộ dữ liệu khác nhau đã có sẵn một cách độc lập. Bộ dữ liệu là sự tổng hợp của 5 nguồn dữ liệu (4 trong đó là từ UCI dataset) với hơn 11 đặc trưng phổ biến khiến cho nó trở thành bộ dữ liệu bệnh tim lớn nhất hiện có phục vụ cho mục đích nghiên cứu, cũng như là cho các phương pháp dự đoán bệnh tim kết hợp

với mô hình học máy. Năm nguồn dữ liệu được thu thập gồm:

- Cleveland: 303 quan sát
- Hungarian: 294 quan sát
- Switzerland: 123 quan sát
- Long Beach VA: 200 quan sát
- Stalog (Heart) Data Set: 270 quan sát

Trong đó:

Tổng số: 1190 quan sát
Bị trùng lập: 272 quan sát
Còn lại: 918 quan sát

B. Mô tả dataset:

Bộ dữ liệu sẽ được trình bày dưới dạng bảng thống kê (file csv), với các thông tin mô tả, số liệu về các đặc trưng liên quan đến các chuẩn đoán y khoa của 918 bệnh nhân, gồm có 12 đặc trưng:

- Age: Tuổi của bệnh nhân [Năm]
- Sex: giới tính của bệnh nhân [M: Nam, F: Nữ]
- ChestPainType: loại đau ngực [TA: Đau thắt ngực điển hình, ATA: Đau thắt ngực không điển hình, NAP: Không đau, ASY: Không có triệu chứng]
- RestingBP: huyết áp khi nghỉ ngơi [mm Hg]
- Cholesterol: Cholesterol trong huyết thanh [mm/dl]
- FastingBS: lượng đường trong máu [1: Nếu FastingBS > 120 mg/dl, 0: Còn lại]
- RestingECG: kết quả điện tâm đồ khi nghỉ ngơi [Bình thường: Bình thường, ST: Có sóng ST-T bất thường (sóng T đảo ngược và/hoặc ST chênh lên hoặc xuống > 0,05 mV), LVH: Cho thấy phì đại thất trái có thể xảy ra hoặc chắc chắn theo tiêu chí của Estes]
- MaxHR: nhịp tim tối đa đạt được [Giá trị số từ 60 đến 202]
- ExerciseAngina: đau thắt ngực do gắng sức [Y: Có, N: Không]
- Oldpeak: ST chênh xuống [Giá trị số được do gắng sức so với nghỉ ngơi]
- ST_Slope: độ dốc của đoạn ST trong kết quả điện tâm đồ cao nhất [Up: Dốc lên, Flat: Bằng phẳng, Down: Dốc xuống]
- HeartDisease: Chuẩn đoán suy tim [1: Suy tim, 0: Bình thường]

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.00	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.00	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.00	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.50	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.00	Up	0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.20	Flat	1
914	68	M	ASY	144	193	1	Normal	141	N	3.40	Flat	1
915	57	M	ASY	130	131	0	Normal	115	Y	1.20	Flat	1
916	57	F	ATA	130	236	0	LVH	174	N	0.00	Flat	1
917	38	M	NAP	138	175	0	Normal	173	N	0.00	Up	0

918 rows × 12 columns

Hình 1. Full dataset

Mục tiêu đã nêu của vấn đề của chúng tôi là dự đoán trước được nguy cơ của bệnh nhân có bị suy tim hay không, do đó, tập dữ liệu của chúng tôi phải được dán nhãn trước.

Do đây cũng là một bài toán về chuẩn đoán vấn đề liên quan đến y khoa, vì vậy nguồn dữ liệu phải yêu cầu cần có độ uy tín và chuẩn xác cao. Những đặc trưng có trong bộ dữ liệu chính là những đặc trưng sẽ góp phần liên quan việc dự đoán suy tim đã qua những nghiên cứu y học chứng minh.

C. Khai phá dataset

	count	mean	std	min	25%	50%	75%	max
Age	918.00	53.51	9.43	28.00	47.00	54.00	60.00	77.00
RestingBP	918.00	132.40	18.51	0.00	120.00	130.00	140.00	200.00
Cholesterol	918.00	198.80	109.38	0.00	173.25	223.00	267.00	603.00
FastingBS	918.00	0.23	0.42	0.00	0.00	0.00	0.00	1.00
MaxHR	918.00	136.81	25.46	60.00	120.00	138.00	156.00	202.00
Oldpeak	918.00	0.89	1.07	-2.60	0.00	0.60	1.50	6.20
HeartDisease	918.00	0.55	0.50	0.00	0.00	1.00	1.00	1.00

Hình 2. Description of the dataset

Comment 1: Bộ dữ liệu có 918 quan sát, trong đó gồm 11 thuộc tính với 5 thuộc tính là dạng Numerical, 7 thuộc tính là Categorical vì vậy cần phải thực hiện encode để đưa các thuộc tính Categorical về dạng Numerical, các thuộc tính đó là: Sex, ChestPainType, FastingBS, RestingECG, ExerciseAngina, ST_Slope.

```
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

Hình 3. Thông tin của tập dữ liệu

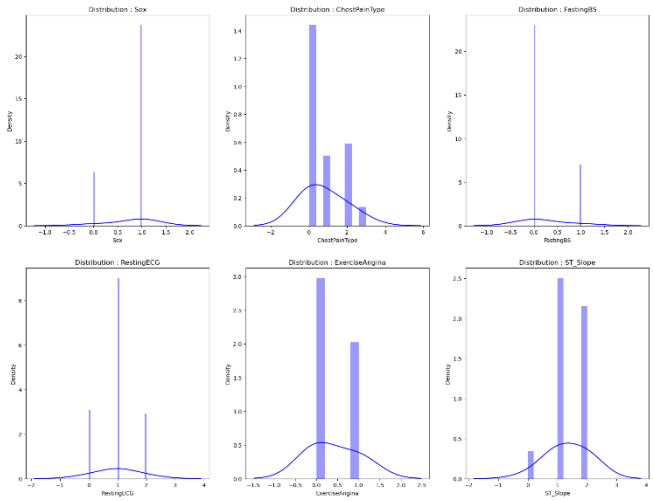
Comment 2: Bộ dữ liệu không có thuộc tính nào có dữ liệu bị khuyết.

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

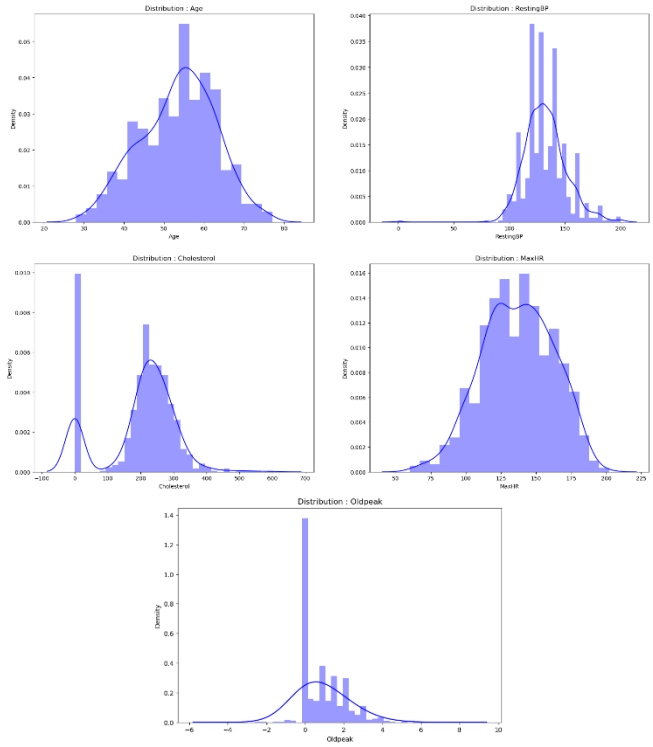
Hình 4. Số lượng dữ liệu khuyết

Comment 3: Phân bố của từng các đặc trưng dữ liệu:

- Các features numerical: Age, RestingPB, MaxHR có phân phối gần giông với phân phối chuẩn.
- Cholesterol bị nhiễu ở cholesterol = 0 nên không tuân theo phân phối chuẩn. Có thể ta sẽ remove phần nhiễu đó để cholesterol tuân theo phân phối chuẩn.

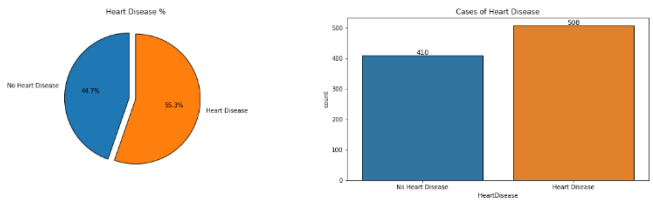


Hình 5. Phân bố của các đặc trưng Catelogical



Hình 6. Phân bố của các đặc trưng Numerical

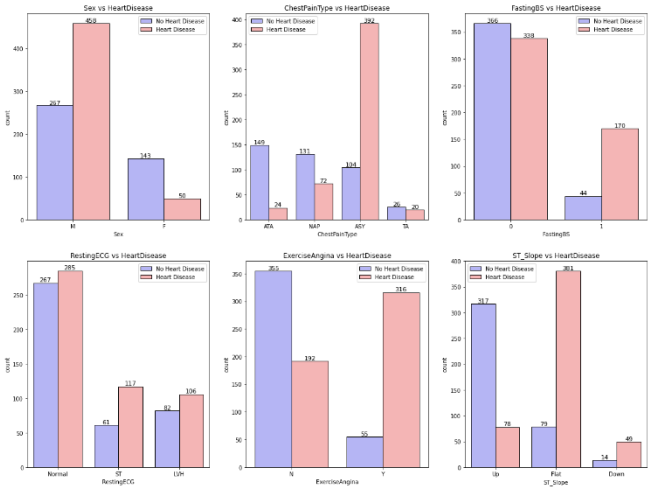
Comment 4: Có thể nói đây là một bộ dữ liệu không bị mất cân bằng, tỉ lệ của hai class target gần như là cân bằng, phân bố bị lệch cho nhãn 1: Suy tim nhưng với tỉ lệ nhỏ (tỉ lệ 0.553:0.447)



Hình 7. Phân bố của target trong tập dữ liệu

Comment 5:

- Người nam dễ bị bệnh tim mạch hơn là người nữ. Tỉ lệ bệnh ở nam là 458/(458 + 267) = 63%, trong khi đó ở nữ là 50/(50 + 143) = 26%.
- Người thuộc loại ASY trong features chestpaintype dễ bị bệnh hơn so với phần còn lại với tỉ lệ 392/(392 + 104) = 79%.
- Nếu một người được chuẩn đoán mắc bệnh Fasting Blood Sugar thì có tới 170/(170 + 44) = 79% khả năng mắc thêm bệnh tim mạch. Nhưng nếu không mắc bệnh Fasting Blood Sugar thì xác suất bệnh tim mạch vẫn là khoảng 50%. Do đó, Fasting Blood Sugar vẫn có thể dùng để phân loại tim mạch, nhưng nó không phải là đặc trưng quá tốt.
- RestingECG không giúp phân tách các trường hợp bệnh tim mạch tốt. Do với mỗi trường hợp trong RestingECG, tỉ lệ mắc bệnh không có sự khác biệt nhiều.
- Bệnh nhân có Exercise Induced Engina mang giá trị Y thường dễ bệnh hơn giá trị N.
- Với ST_Slope, flat slope cho thấy khả năng cao mắc bệnh tim. Down cũng tương tự như flat, nhưng có khả ít điểm dữ liệu mang giá trị down

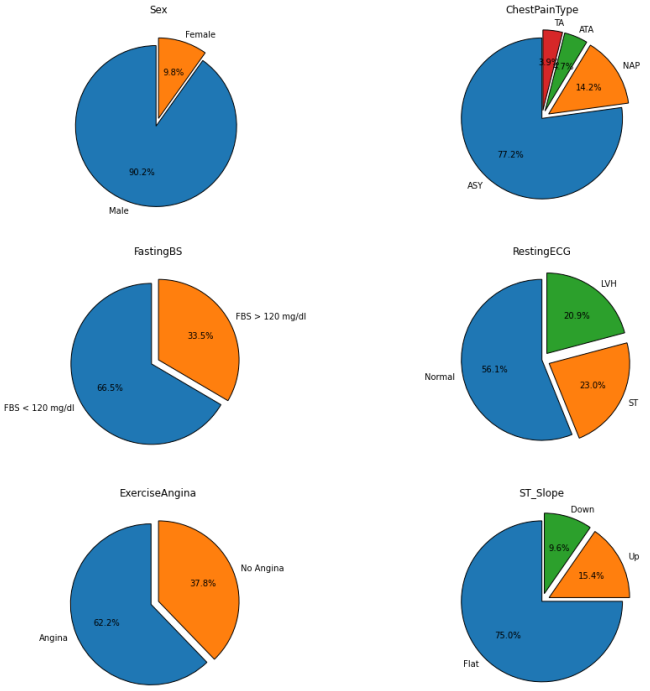


Hình 8. Mối liên hệ giữa các đặc trưng categorical và target (heart disease)

Comment 6:

Trong số những người bị bệnh tim có:

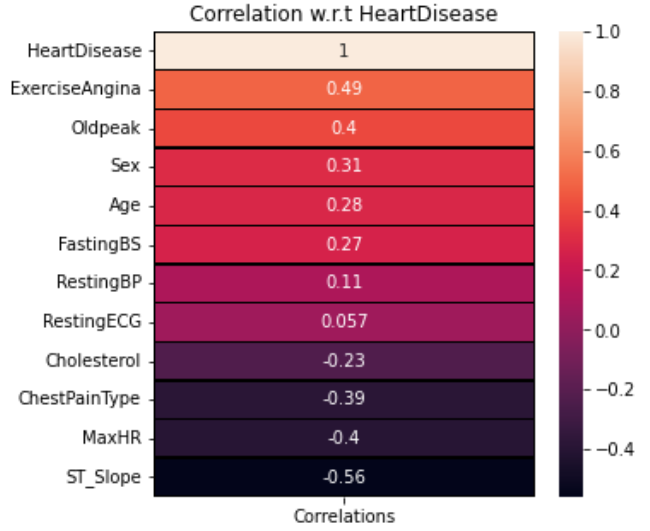
- Hơn 90% là nam giới.
- Hơn 77% quan sát thuộc loại ASY (Không có triệu chứng) (chest pain)
- 2/3 số quan sát có Fasting Blood Sugar level < 120 mg/dl.
- Hơn 1/2 quan sát thuộc Normal level (RestingECG)
- Hơn 62% quan sát có đau thắt ngực do gắng sức (Exercise Angina: Angina)
- 3/4 số quan sát có độ dốc của đoạn ST trong kết quả điện tâm đồ cao nhất là bằng phẳng (ST_Slope: Flat)



Hình 8. Mối liên hệ giữa các đặc trưng Numerical và target (heart disease)

Comment 6:

Nhìn vào độ tương quan của các features với heartdisease, ta thấy ResingBP và RestingECG có tương quan yếu với HeartDisease so với các features khác (lần lượt là 0.11, 0.27). Nên chúng tôi sẽ xem xét loại bỏ 2 đặc trưng này khỏi bộ dữ liệu.



Hình 9. Độ tương quan giữa các đặc trưng với target (heart disease)

D. Tiền xử lý dataset

Qua bước phân tích bộ dữ liệu để đưa ra các hướng giải quyết trên, chúng tôi sẽ thực hiện loại bỏ một số đặc trưng RestingBP, RestingECG, chỉ giữ lại các đặc trưng Age, Sex, ChestPainType, Cholesterol, FastingBS, MaxHR, ExerciseAngina, Oldpeak, ST_Slope.

Ngoài ra chúng tôi còn cần đưa các thuộc tính Categorical là Sex, ChestPainType, FastingBS, RestingECG, ExerciseAngina, ST_Slope về Numerical. Công cụ hỗ trợ mà chúng tôi sẽ sử dụng cho bước này sẽ là LabelEncoder của thư viện Sklearn.

Đây là bộ dữ liệu sau khi thực hiện tiền xử lý:

	Age	Sex	ChestPainType	Cholesterol	FastingBS	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	289	0	172	0	0.00	2	0
1	49	0	2	180	0	156	0	1.00	1	1
2	37	1	1	283	0	98	0	0.00	2	0
3	48	0	0	214	0	108	1	1.50	1	1
4	54	1	2	195	0	122	0	0.00	2	0
...	...	...	...	...	...	...	...	...	...	...
913	45	1	3	264	0	132	0	1.20	1	1
914	68	1	0	193	1	141	0	3.40	1	1
915	57	1	0	131	0	115	1	1.20	1	1
916	57	0	1	236	0	174	0	0.00	1	1
917	38	1	2	175	0	173	0	0.00	2	0

918 rows × 10 columns

Hình 10. Tập dữ liệu sau khi được tiền xử lý

III. MACHINE LEARNING METHODS

Bài toán phân loại (Classification) là một phương pháp trong nhóm Học có giám sát (Supervised learning) trong Machine Learning, trong đó phân loại chính là một quá trình phân lớp một đối tượng dữ liệu vào một hay nhiều lớp có sẵn nhờ vào mô hình phân lớp.

Mô hình phân lớp sẽ được xây dựng dựa trên một tập dữ liệu đầu vào mà trước đó đã được gắn nhãn trước. Chúng tôi gọi đây là tập huấn luyện. Quá trình phân lớp có thể hiểu là quá trình gắn nhãn cho đối tượng dữ liệu.

Có nhiều bài toán phân loại như phân lớp nhị phân (binary classification) hay phân lớp đa lớp (multiclass classification). Bài toán phân lớp nhị phân là bài toán gán nhãn dữ liệu cho đối tượng vào một trong hai lớp khác nhau dựa vào việc dữ liệu đó có hay không có các đặc trưng (feature) của bộ phân lớp. Bài toán phân lớp đa lớp là quá trình phân lớp dữ liệu với số lượng lớp lớn hơn hai. Như vậy với từng dữ liệu phải xem xét và phân lớp chúng vào những lớp khác nhau chứ không phải là 2 lớp như bài toán phân lớp nhị phân.

Với bài toán phân loại:

- Input: Giá trị của các biến độc lập X của quan sát
- Output: Nhãn của quan sát (biến phụ thuộc Y)

Để hiểu rõ hơn về bài toán phân loại, chúng tôi sẽ sử dụng một bộ dữ liệu kinh điển trong khi bắt đầu tìm hiểu về các thuật toán phân loại trong Machine Learning - bộ dữ liệu hoa diên vĩ (Iris dataset).

Chúng ta sẽ có một danh sách các đối tượng với mỗi đối tượng sẽ có các thuộc tính hay còn gọi là danh sách các quan sát (observation). Trong bộ dữ liệu này, với mỗi quan sát ta sẽ có các thuộc tính như chiều dài, chiều ngang của đài hoa (sepal) và cánh hoa (petal). Ngoài ra, với mỗi quan sát, chúng ta sẽ có một lớp (class) và trong trường hợp này lớp của một quan sát chính là loài của nó. Ở đây chúng ta sẽ có ba loài là Setosa, Versicolour và Virginica.

Ví dụ chúng ta sẽ có 210 quan sát nhưng 10 quan sát cuối sẽ không biết được lớp của nó là gì (không biết hoa thuộc loài nào). Vậy chúng ta sẽ dựa trên 200 đối tượng mà đã biết trước lớp (loài hoa) để xây dựng mô hình để phân loại 10 đối tượng cuối. Bằng cách thực hiện các phép so sánh thuộc tính giữa các loài như loài Setosa thường có cánh hoa dài hơn loài Versicolour, loài Virginica có độ rộng đài hoa lớn hơn Setosa hay loài Versicolour thường có cánh hoa và đài hoa nhỏ hơn loài Virginica, … chúng ta có thể tổng hợp ra một chuỗi các điều kiện để phân loại các đối tượng, nói cách khác, chúng tôi

xây dựng được mô hình để phân loại được 10 đối tượng "hoa" cuối cùng dựa trên các thuộc tính.

A. Logistic Regression:

Bài toán phân loại là bài toán mà bạn cố gắng dự đoán các kết quả rời rạc, chẳng hạn như liệu ai đó có mắc bệnh hay không. Ngược lại, bài toán hồi quy là bài toán mà bạn cố gắng dự đoán giá trị của một biến liên tục, chẳng hạn như giá bán một căn nhà. Mặc dù hồi quy logistic có hồi quy trong tên của nó, nhưng đó là một thuật toán cho các vấn đề phân loại. Logistic Regression có thể được xem như là phương pháp phân loại học tập có giám sát quan trọng nhất. Đó là một phần mở rộng linh hoạt, nhanh chóng của một mô hình tuyến tính tổng quát.

1. Giới thiệu

Hồi quy logistic là một phương pháp phân tích thống kê được sử dụng để dự đoán giá trị dữ liệu dựa trên các quan sát trước đó của tập dữ liệu. Hồi quy logistic là một thuật toán cơ bản nhưng tuyệt vời, nó hoạt động tốt khi mối quan hệ giữa các đặc trưng dữ liệu và mục tiêu không quá phức tạp.

Mục đích của hồi quy logistic là ước tính xác suất của các sự kiện, bao gồm xác định mối quan hệ giữa các thuộc tính từ đó đự đoán xác suất của các kết quả, nên đối với hồi quy logistic ta sẽ có:

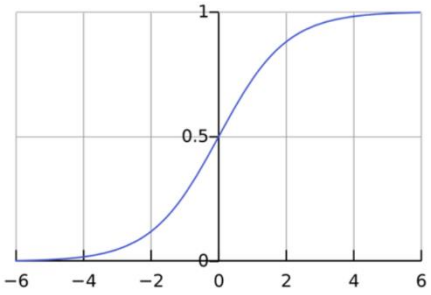
- Input: điểm dữ liệu (gồm các đặc trưng của điểm dữ liệu)
- output: Xác suất điểm dữ liệu rơi vào nhãn 0 hoặc nhãn 1

Hồi quy logistic tạo ra các trọng số cho các đặc trưng dữ liệu thường có thể diễn giải được, điều này là hữu ích nếu như ta cần lí giải cho những dự đoán của các kết quả.
Có thể nói giải quyết bài toán hồi quy logistic cũng chính là bài toàn tối ưu hóa.

2. Hàm hồi quy Logistic

Logistic Regression là một mô hình thống kê sử dụng hàm logistic, hay hàm logit trong toán học làm phương trình giữa x và y. Hàm logit ánh xạ y dưới dạng hàm sigmoid của x:

$$f(x) = \frac{1}{1 + e^{-x}}$$



Hình 11. Đồ thị hàm Sigmoid

Hàm logic chỉ trả về giá trị nằm trong khoảng từ 0 đến 1 cho biến phụ thuộc, bất kể các giá trị của biến độc lập. Đây là cách hồi quy logistic ước tính giá trị của biến phụ thuộc. Các phương pháp hồi quy logistic cũng mô hình hóa các phương trình giữa nhiều biến độc lập và một biến phụ thuộc.

Các công thức hồi quy logistic giả định mối quan hệ tuyến tính giữa các biến độc lập để từ đó có thể suy ra được sự ảnh hưởng đến giá trị của biến phụ thuộc.

$$y = f(w_0 + w_1x_1 + \cdots + w_nx_n)$$

với: w đại diện cho hệ số hồi quy,
x đại diện cho các biến độc lập,
y đại diện cho biến phụ thuộc,
f đại diện cho hàm logit

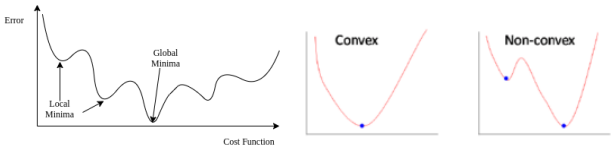
3. Hàm mất mát (Loss function)

Với đầu ra của mô hình chính là xác suất của điểm dữ liệu rơi vào nhãn nào, giả sử rằng xác suất để điểm dữ liệu x rơi vào class 1 là $f(w^Tx)$ và rơi vào class 0 là $1 - f(w^Tx)$. Với mô hình được giả sử như vậy, với các điểm dữ liệu dữ liệu huấn luyện (đã biết đầu ra y), ta có thể viết như sau:

$$P(y_i = 1|x_i; w) = f(w^Tx_i)\ (1)$$
$$P(y_i = 0|x_i; w) = 1 - f(w^Tx_i)\ (1)$$

trong đó (1) được hiểu là xác suất xảy ra sự kiện đầu ra $y_i = 1$ khi biết tham số mô hình là w và dữ liệu đầu vào là x_i. Việc giải bài toán hồi quy logistic cũng chính là tìm hệ số w sao cho $f(w^Tx_i)$ càng gần với 1 càng tốt với các điểm dữ liệu x thuộc class 1 và càng gần 0 với những điểm thuộc class 0. Nói cách khác, mục tiêu của việc huấn luyện mô hình chính là ta đi tìm bộ trọng số w sao cho giá trị hàm mất mát đạt giá trị nhỏ nhất.

Hàm mất mát chính là hàm số để đo sự sai khác giữa giá trị thực tế (ground true) và giá trị mô hình dự đoán. Trong hồi quy tuyến tính, ta thường sử dụng hàm lỗi bình phương trung bình (MSE) làm hàm mất mát. Nhưng trong hồi quy logistic, sử dụng giá trị trung bình của sự khác biệt bình phương giữa kết quả thực tế và kết quả dự đoán làm hàm độ lỗi để tìm nghiệm tối ưu bài toán có thể đưa ra một hàm không lồi, lượn sóng; chứa nhiều cực tiểu cục bộ, gây khó khăn trong việc tìm nghiệm tối ưu để tối thiểu được hàm mất mát.



Hình 12. Ví dụ về các trường hợp của hàm độ lỗi

Đối với những bài toán hồi quy logistic 2 nhãn thường gặp, cũng như là bài toán dự đoán suy tim ở đây (bệnh nhân có bị suy tim hoặc không) thì một trong những hàm mất mát phổ biến thường được sử dụng nhất là Binary Cross Entropy.

a) Binary Cross Entropy

Cross Entropy được định nghĩa là 1 thước đo sự khác nhau giữa 2 phân phối xác suất của 1 biến ngẫu nhiên. Nó được sử dụng rộng rãi cho bài toán phân loại đối tượng, đặt biệt là các bài toán phân lớp nhị phân.

$$L_{Binary-CE(y,\hat{y})} = -(ylog(\hat{y}) + (1-y)\log(1-\hat{y}))$$

hoặc

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\big(p(y_i)\big) + (1-y_i)\log\big(1-p(y_i)\big)$$

với y chính là nhãn thực và p(y) chính là dữ đoán xác xuất cho diểm dữ liệu và tính trung bình sai số trên toàn bộ N điểm dữ liệu.

Ví dụ: mục tiêu của ta là dự đoán một điểm dữ liệu có nhãn là 1, nếu mô hình đưa ra dự đoán là 1 thì hàm mất mát có giá trị là 0 vậy thì vấn đề ta đề ra ban đầu là cực tiểu hàm mất mát đã được giải quyết. Nhưng nếu kết quả dự đoán của mô hình đưa ra khác 1 và càng gần 0 thì khi ấy giá trị của hàm mất mát sẽ càng lớn, khi ấy ta cần quay lại bài toán được đề ra ban đầu chính là tìm bộ trọng số w sao cho cực tiểu được giá trị hàm mất mát.

+Ưu điểm: Binary Cross-Entropy làm việc tốt nhất trong trường hợp dữ liệu phân phối đều, dựa theo phân phối Bernoulli.

+Nhược điểm: Không thực sự hoạt động tốt khi dữ liệu bị mất cân bằng.

b) alpha-beta Binary Cross Entropy

Đây được xem như là một biến thể của Binary Cross-Entropy (BCE) và Weighted Binary Cross Entropy (W_BCE). Hàm mất mát này được sử dụng trong trường hợp các bộ dữ liệu bị mất cân bằng giữa các nhãn. Hàm loss này quan tâm đến cả 2 mẫu negatives và positives, nên cả 2 đều được đánh trọng số.

$$L_{alpha_{beta}-BCE(y,\hat{y})} = -\alpha*(ylog(\hat{y})+\beta*(1-y)\log(1-\hat{y})$$

với 2 trọng số α và β, ta có thể điều chỉnh mức độ ảnh hưởng của các nhãn (ví dụ: giảm ảnh hưởng từ các mẫu dễ học, tăng ảnh hưởng của các mẫu khó học).

4. Tối ưu hàm mất mát

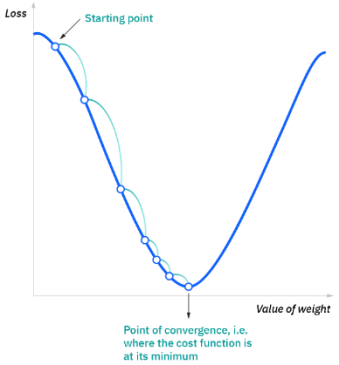
Tóm lại, mục tiêu đề ra chính của bài toán hồi quy Logistic chính là tìm bộ trọng số w để giải được bài toán tối ưu cực tiểu được hàm mất mát.

Gradient descent là một thuật toán tối ưu hóa lặp đi lặp lại, thuật toán này giúp ta tìm được cực tiêu của một hàm khả vi bằng cách đi ngược hướng của đạo hàm. Trong quá trình này, thuật toán sẽ tiến hành lần lượt thử các giá trị trọng số khác nhau và cập nhật lại chúng để đạt được giá trị tối ưu, giảm thiểu được đầu ra. Nó đúng như yêu cầu của mục tiêu đề ra chính của bài toán hồi quy nói trên. Bằng cách này, chúng ta có thể tìm ra một bộ trọng số w tối ưu để giảm thiểu được chi phí của mô hình: $\min_{w} J(w)$

Ta sẽ cập nhập bộ trọng số như sau:

$$Repeat\ \{w_i = w_i - \mu \frac{\partial}{\partial w_i} J(w)\}$$

với $J(w)$ chính là hàm mất mát được sử dụng, $\frac{\partial}{\partial w_i} J(w)$ là đạo hàm theo từng biến của w.



Hình 13. Ví dụ Gradient Descent cập nhật trọng số

5. Regularization

Regularization là một kỹ thuật để giải quyết vấn đề overfitting trong thuật toán học máy bằng cách thêm giá trị phạt (penalty) vào hàm chi phí.

a) Ridge (L2 Regularization)

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} Cost(h(x^{(i)}), y^{(i)}) + \frac{l}{2m}\sum_{j=1}^{m} w_j^2$$

$\sum_{j=1}^{m} w_j^2$ được xem là thành phần phạt (penalty term) của hàm mất mát, và l chính là tham số Regularization, nó có nhiệm vụ điều chỉnh mức độ "phạt" của thành phần phạt lên hàm mất mát.

+Trường hợp l nhỏ thì vai trò của thành phần phạt trở nên ít quan trọng. Mức độ kiểm soát quá khớp của mô hình sẽ trở nên kém hơn.

+Trường hợp l lớn chúng ta muốn gia tăng mức độ kiểm soát lên độ lớn của các hệ số ước lượng và qua đó giảm bớt hiện tượng qúa khớp.

Giá trị l càng lớn đồng nghĩa với việc khiến cho w_i co dần về 0, điều này có thể dẫn đến mô hình sẽ bị underfitting.

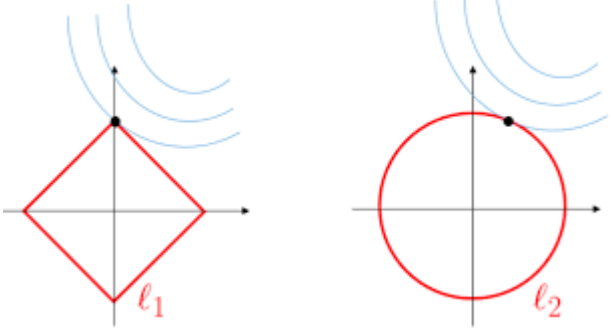
$l = 0$ tương đương với việc loại bỏ đi thành phần phạt, hàm mất mát sẽ quay trở về ban đầu.

b) Lasso (L1 Regularization)

Trong Lasso (L1 Regularization) thay vì ta sử dụng thành phần "phạt" là norm chuẩn bậc hai thì chúng ta sử dụng norm chuẩn bậc 1. Hồi qui Lasso sẽ có xu hướng lựa chọn ra một biến trong nhóm các biến đa cộng tuyến và bỏ qua những biến còn lại. Nói một cách dễ hiểu, hồi quy Lasso có thể được gọi là hồi quy 'chọn biến'.

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} Cost(h(x^{(i)}), y^{(i)}) + \frac{l}{2m}\sum_{j=1}^{m} w_j$$

$\sum_{j=1}^{m} w_j$ là thành phần "phạt", và l chính là tham số điều chuẩn. Ta sẽ tăng giảm hệ số l để mô hình bớt phứt tạp hơn, và tác động của l cũng sẽ tương tự như hồi quy Ridge.



Hình 14. Hàm mục tiêu và miền xác định của Regularization

c) Elastic Net

Elastic Net cho phép chúng ta kết hợp đồng thời cả 2 thành phần điều chuẩn nói trên theo một kết hợp tuyến tính lồi.

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} Cost(h(x^{(i)}), y^{(i)}) + \frac{l}{2m}(\alpha\sum_{j=1}^{m} w_j + \frac{(1-\alpha)}{2}\sum_{j=1}^{m} w_j^2)$$

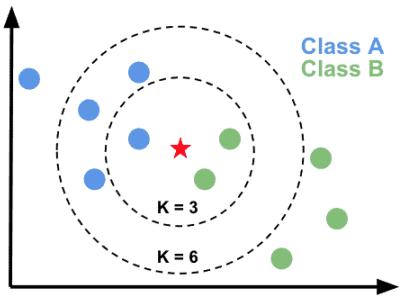
l vẫn đóng vai trò là hệ số nhân cho thành phần điều chuẩn.

α chính là hệ số nhân của norm chuẩn bậc 1 trong thành phần điều chuẩn. Giá trị của α sẽ thuộc [0 ; 1], nếu như $\alpha = 0$ thì thành phần điều chuẩn hoàn toàn trở thành norm chuẩn bậc 2, và với $\alpha = 1$ thì bài toán trở thành norm chuẩn bậc 1.

B. K-nearest neigbors:

1. *Giới thiệu*

KNN (K-Nearest Neighbours) là một thuật toán Supervised Learning khá đơn giản, có thể áp dụng được cho cả bài toán Classification và Regression, nhưng thường là Classification. Đối với bài toán Classification, thuật toán này dự đoán nhãn của một điểm dữ liệu mới bằng cách xem xét K điểm dữ liệu gần với điểm dữ liệu này nhất, từ đó quyết định được nhãn của điểm đó.



Hình 15. Ví dụ về các trường hợp K khác nhau ở KNN

Ở ví dụ này, với K=3, điểm sao sẽ được gán nhãn thuộc lớp B, do trong 3 điểm gần nó nhất nó 2 điểm B và 1 A, tương tự vậy với K=6, nó sẽ được gán là A (theo cách hiểu đơn giản nhất, major voting)

2. *Chi tiết thuật toán*

Cụ thể hơn về cách hoạt động của KNN, khi muốn dự đoán một điểm dữ liệu mới, thuật toán sẽ tính "khoảng cách" từ điểm này đến TẤT CẢ các điểm dữ liệu có trong tập training. "Khoảng cách" này có thể được tính bằng nhiều hàm khác nhau tùy vào số chiều không gian của dữ liệu. Sau đó các khoảng cách được sắp xếp lại để từ đó chọn ra K điểm có khoảng cách gần nhất. Việc quyết định nhãn của điểm dữ liệu mới có thể được thực hiện bằng cách biểu quyết theo số đông (major voting) hoặc đánh trọng số cho các điểm gần nhất.

Cần lưu ý rằng với KNN, mô hình thực chất không học gì cả mà chỉ nhớ các điểm dữ liệu training, đến khi ta cần dự đoán nhãn một điểm mới thì mô hình mới bắt đầu thực hiện các bước tính toán để cho ra kết quả

3. *Các tham số trong mô hình KNN*

Ta sử dụng hàm có sẵn trong thư viện **Sklearn** để gọi mô hình KNN:

sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)

Sau đây ta sẽ tìm hiểu một số tham số quan trọng, việc thay đổi giá trị các tham số này sẽ ảnh hưởng trực tiếp đến kết quả của mô hình:

a) n_neighbors (k)

Đây chính là tham số k, hay chính là số nhóm trong thuật toán. Tham số này sẽ quyết định chọn bao nhiêu điểm dữ liệu gần với điểm cần dự đoán để dự đoán nhãn cho nó. Không có giá trị k khuyên dùng cho tất cả bài toán vì nó phụ thuộc rất nhiều vào dữ liệu, tuy nhiên trong các bài toán chỉ có 2 nhãn (binary), k sẽ thường hay có giá trị lẻ để tránh trường hợp khi k chẵn thì số điểm của cả 2 nhãn gần điểm quan tâm bằng nhau -> không thể quyết định nhãn của điểm này.

Tham số chỉ nhận giá trị nguyên dương. Mặc định là bằng 5. Giá trị k càng lớn thì những điểm nhiễu sẽ càng ít ảnh hưởng, nhưng ranh giới giữa 2 class sẽ ít rõ ràng hơn.

b) Weights

Tham số có thể nhận 1 trong 2 giá trị hàm cung cấp sẵn: "uniform" hoặc "distance", hoặc người dùng có thể tự cung cấp hàm để gán trọng số.

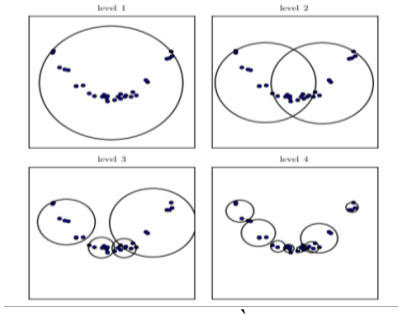
Tham số này quy định cách đánh trọng số cho k điểm gần nhất, để từ đó quyết định nhãn cho điểm đang xét. "uniform" nghĩa là mô hình sẽ xem k điểm gần điểm đang xét có độ quan trọng là như nhau, hay có thể hiểu là trọng số cho các điểm đó đều là 1. "distance" thì mô hình sẽ gán trọng số khác nhau tùy thuộc vào khoảng cách giữa điểm đang xét và k điểm kia. Điểm nào càng gần thì sẽ càng quan trọng hơn, hay có trọng số cao hơn và ngược lại. Mặc định là "uniform".

c) algorithm

Tham số này quy định cách mô hình giải bài toán tìm k điểm gần nhất với dữ liệu là tất cả các khoảng cách đã được tính trước. Nhận một trong 4 giá trị sau: {"brute", "ball_tree", "kd_tree", "auto"}. Mặc định là "auto".

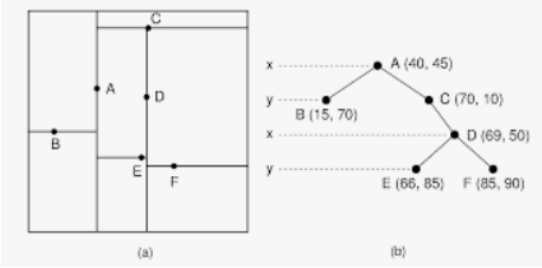
Khi tham số là "brute", nghĩa là mô hình dùng cách "trâu bò", thủ công hay brute-force. Mô hình sẽ lần lượt so sánh tất cả các giá trị khoảng cách với nhau để tìm ra các điểm gần nhất.

Với "ball_tree", mô hình sẽ lưu trữ các điểm có nhiều chiều vào cấu trúc ball_tree, một cấu trúc cây nhị phân nhưng nó bao gồm các "cluster"(cụ thể hơn là vòng tròn 2D hoặc hình cầu 3D) bao bọc lấy nhau. Với cấu trúc dữ liệu này, việc tìm ra k điểm gần nhất sẽ nhanh chóng hơn rất nhiều khi bộ dữ liệu lớn dần.



Hình 16. Ví dụ về Ball-tree

"kd_tree" cũng tương tự như ball tree, là một cấu trúc cây nhị phân dùng để lưu trữ các điểm dữ liệu nhiều chiều, nhưng sẽ chia đôi các điểm dữ liệu theo các giá trị trung bình do người dùng lựa chọn (mean, mode, median) theo từng trục lần lượt nhau.



Hình 17. Cấu trúc kd_tree

Cuối cùng, "auto" sẽ giúp mô hình tự lựa chọn 1 trong 3 thuật toán trên để sử dụng tùy thuộc vào dataset truyền vào hàm.

Tất cả thuật toán đều có ưu nhược điểm riêng: "brute" sẽ cho ra kết quả rất tốt đối với bộ dữ liệu nhỏ, nhưng khi kích thước bộ dữ liệu lớn dần thì 2 cấu trúc cây sẽ thể hiện tốt hơn. Nếu số chiều của bộ dữ liệu nhỏ thì "kd_tree" sẽ vượt trội hơn, ngược lại, nếu số chiều của bộ dữ liệu lớn thì "ball_tree" là tốt nhất. Vì vậy quan trọng là ta phải biết chọn algorithm phù hợp để đạt được kết quả tốt nhất trong thuật toán KNN.

d) leaf_size

Tham số này có ý nghĩa khi thuật toán ta chọn là 1 trong 2 cấu trúc cây, nó sẽ quy định kích thước của những node lá trong cấu trúc cây. Tham số này sẽ ảnh hưởng đến tốc độ hình thành nên cây cũng như lượng bộ nhớ cần thiết cho cấu trúc cây.

e) p

Tham số này mang giá trị là số mũ trong độ đo Minkowski. Vì đối với độ đo này, với các giá trị số mũ khác nhau thì công thức sẽ khác nhau. Nếu p = 1, công thức sẽ tương đương manhattan_distance(l1), bằng 2 thì là euclidean_distance(l2), với giá trị p khác thì sẽ áp vào công thức gốc của Minkowski.

f) metric

Là công thức dùng để tính "khoảng cách" giữa các điểm dữ liệu, mặc định và thường dùng nhất là "minkowski", tuy nhiên vẫn có thể sử dụng độ đo khác do người dùng truyền vào.

4. Ưu và nhược điểm của KNN

g) Ưu điểm:

+KNN là một thuật toán đơn giản, dễ hiểu dễ áp dụng.

+Như đã nói ở trên, mô hình không học gì cả nên bước training là không gần như không có.

+Do là thuật toán phụ thuộc vào bộ nhớ (memory-based approach) nên KNN có thể phản hồi nhanh chóng với các thay đổi trong dữ liệu training.

+KNN có thể dùng cho cả 2 bài toán Classification và Regression.

+Dễ dàng cài đặt khi số nhãn đầu ra lớn hơn 2.

h) Nhược điểm:

+Do phải nhớ tất cả điểm dữ liệu và cần tính toàn bộ khoảng cách nên rõ ràng KNN rất chậm, tốn nhiều bộ nhớ cũng như chi phí tính toán cao khi dữ liệu đầu vào lớn.

+Khi số chiều của dữ liệu đầu vào lớn, độ chính xác của mô hình sẽ giảm đáng kể.

+Do là thuật toán tính khoảng cách (distance-based) nên dữ liệu đầu vào cần phải được chuẩn hóa theo cùng một kiểu để cho ra kết quả chính xác.

+Thuật toán rất nhạy cảm với các điểm dữ liệu nhiễu vì nó đơn giản chọn các điểm gần nhất dựa theo khoảng cách chứ không theo nhãn.

+Thuật toán sẽ cho kết quả không tốt khi dữ liệu mất cân bằng (số nhãn của 1 lớp lớn hơn nhiều so với các lớp còn lại).

C. Decision Tree:

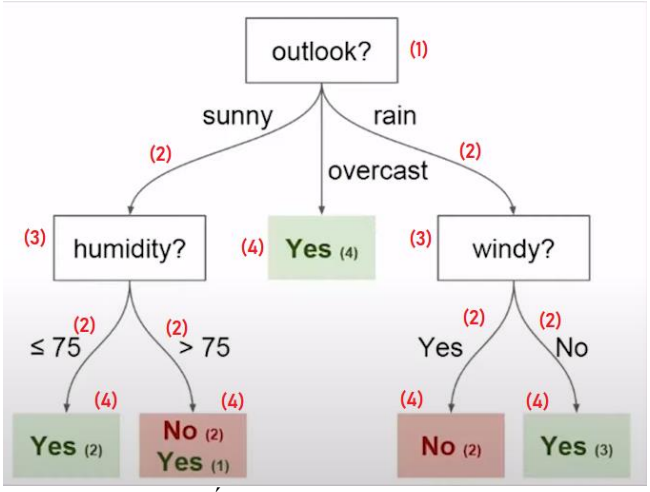
Decision Tree là một phương pháp học thuộc nhóm Học có giám sát (Supervised learning) được sử dụng trong các bài toán Phân loại (classification) và Hồi quy (Regression). Mục tiêu là tạo ra mô hình dự đoán giá trị của biến mục tiêu bằng các học các quy tắc quyết định đơn giản được suy ra từ các thuộc tính của dữ liệu đầu vào. Một cây có thể được coi là một hằng số xấp xỉ từng phần. Cây càng sâu, các quy tắc quyết định càng phức tạp và mô hình càng phù hợp.

Trong bài báo cáo này, tôi chỉ đề cập tới bài toán Phân loại, đặc biệt thuật toán CART (Classification and Regression Trees)

1. Cây quyết định cơ bản

Có thể nói, việc xây dựng Decision Tree giống với quá trình đưa ra quyết định của con người. Tôi đưa vào dữ liệu ban đầu các câu hỏi, điều kiện và trong quá trình đó, tập dữ liệu dữ liệu sẽ được chia nhỏ thành các tập dữ liệu con cho đến khi đưa ra quyết định.

Lấy ví dụ về Decision Tree cơ bản về việc quyết định đi chơi:



Hình 18. Cây quết định trong việc lựa chọn đi chơi hay không?

Decision Tree được biểu diễn dưới dạng các node liên kết với nhau. Trong đó chia ra làm hai loại là node lá (leaf node hoặc terminal node) và node thể hiện câu hỏi (non-leaf node). Chi tiết hơn:

(1): Node gốc (Root node) là node thể hiện câu hỏi đầu tiên.

(2): Cạnh (Edge) thể hiện hướng kết quả chia của node hiện tại tới node kế tiếp.

(3): Các node con (child node) thuộc các node thể hiện câu hỏi, có đầu vào và có đầu ra.

(4): Các node con (child node) thuộc các node lá, chỉ có đầu vào. Đây là node thể hiện dự đoán kết quả của Cây quyết định.

2. Thuật toán CART

Thuật toán CART là một trong những thuật toán được sử dụng để xây dựng Decision Tree. Thuật toán sẽ hoạt động thông qua một quá trình sau:

- Tìm điểm phân chia tốt nhất của mỗi đầu vào thu được.
- Dựa trên các điểm phân chia tốt nhất ở bước 1, điểm phân chia "tốt nhất" mới được xác định.
- Tách đầu vào đã chọn theo điểm phân chia "tốt nhất".
- Tiếp tục tách cho đến khi thỏa mãn quy tắc dừng hoặc không còn khả năng tách mong muốn nữa.

Các bước để tạo Decision Tree thông qua thuật toán:

Sử dụng thuật toán Tham lam: Trong phần này, dữ liệu đầu vào được chia bằng đệ quy nhị phân, các giá trị sẽ được căn chỉnh và các điểm phân chia sẽ được thử và đánh giá bằng cách sử dụng hàm chi phí.

Tiêu chí dừng: Vì sử dụng phương pháp chia đệ quy nhị phân nên thuật toán phải biết khi nào nên dừng. Thuật toán sẽ sử dụng lượng dữ liệu đào tạo tối thiểu được phân bổ cho mỗi node. Nếu số lượng nhỏ hơn ngưỡng xác định, việc chia tiếp sẽ bị từ chối và node đó cũng được coi là node cuối cùng.

Tỉa cây (Tree pruning): Độ phức tạp của cây được xác định là số phân chia trong cây nên chọn những cây có ít nhánh hơn vì chúng dễ nắm bắt và ít bị phân cụm dữ liệu hơn. Bằng cách thông qua từng node và đánh giá hiệu quả của việc xóa nó bằng cách sử dụng bộ kiểm tra giữ lại (hold-out test set) là phương pháp cắt tỉa nhanh nhất và đơn giản nhất.

Chuẩn bị dữ liệu (data preparation): Không cần chuẩn hóa dữ liệu cho thuật toán.

3. Chỉ số đánh giá

Tôi sẽ sử dụng gini index để đánh giá việc phân chia ở các node có điều kiện tốt hay không. Để tính được Gini index thì đầu tiên ta phải tính được chỉ số Gini ở từng node với công thức:

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Trong đó:

C : số lớp cần phân loại

$p_i = \frac{n_i}{N}$, với n_i là số lượng phần tử ở lớp thứ i, N là tổng số lượng phần tử ở node đó.

Sau khi có được chỉ số Gini ở node cha và node con, tôi sẽ tính được Gini index qua công thức:

$$Gini_{index} = gini(p) - \sum_{i=1}^{K} \frac{m_k}{M} gini(c_k)$$

Trong đó:

$gini(p)$ là chỉ số gini ở node cha

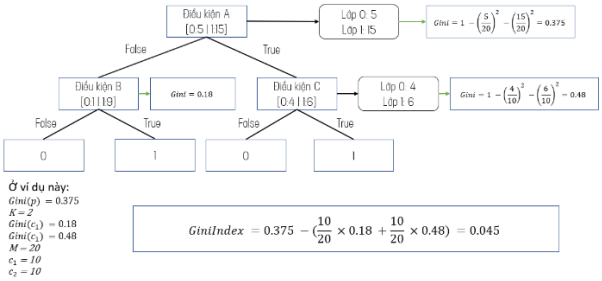
K là số node con được tách ra

$gini(c_k)$ là chỉ số gini ở node con thứ k

M là số phần tử ở node p ($\sum_{i=1}^{K} m_i = M$)

m_i là số phần tử ở node con thứ I

Ví dụ:



Hình 19. Ví dụ về cách tính điểm gini

Với trường hợp sử dụng điểm Gini để đánh giá mô hình cây, tôi sẽ muốn Gini index càng thấp càng tốt.

4. *Overfitting*

Các thuật toán Decision Tree nói chung nếu xây dựng một cây đủ sâu thì sẽ tách ra được các node chỉ chứa dữ liệu của một lớp nhất định nên mô hình sẽ dễ bị trường hợp overfitting.

Để giải quyết trường hợp mô hình Decision Tree bị overfitting với dữ liệu, tôi sẽ có hai cách giải quyết:

- Dừng việc thêm các node điều kiện vào cây bằng cách
  Giới hạn độ sâu của cây
  Chỉ định số phần tử tối thiểu trong node lá, nếu số phần tử ít hơn chỉ định thì sẽ không tách nữa.
- Purning (phương pháp tỉa cây)

5. *Ưu, nhược điểm*

a. *Ưu điểm:*

- Đơn giản, dễ hiểu. Cây có thể được hình dung qua sơ đồ.
- Yêu cầu chuẩn hóa dữ liệu không nhiều.
- Chi phí sử dụng cây là logarit theo số lượng điểm dữ liệu được sử dụng để huấn luyện cây.
- Có khả năng xử lý các vấn đề đa đầu ra.
- Sử dụng mô hình hộp trắng. Nếu một tình huống nhất định có thể quan sát được trong một mô hình, thì lời giải thích cho điều kiện đó dễ dàng được giải thích bằng logic boolean.
- Có thể xác nhận một mô hình bằng cách sử dụng các bài kiểm tra thống kê.

b. *Nhược điểm:*

- Mô hình có thể tạo ra cây quyết định quá phức tạp không tổng quát được dữ liệu (overfitting), có thể sử dụng cơ chế cắt tỉa hay đặt số mẫu tối thiểu ở một node hoặc đặt giới hạn độ sâu để giảm tránh vấn đề này.
- Cây quyết định có thể không ổn định vì các biến thể nhỏ (outliers) trong dữ liệu có thể dẫn đến việc tạo ra một cây hoàn toàn khác. Vấn đề này được giảm thiểu bằng cách sử dụng Random Forest.
- Cần cân bằng dữ liệu trước khi fit với model.

D. Random Forest:

1. *Giới thiệu*

Random forest là thuật toán thuộc nhóm supervised learning, có thể giải quyết cả bài toán regression và classification. Mô hình *random forest* được huấn luyện dựa trên sự phối hợp giữa luật *kết hợp* (*ensembling*) và quá trình *lấy mẫu tái lặp* (*boostrapping*).

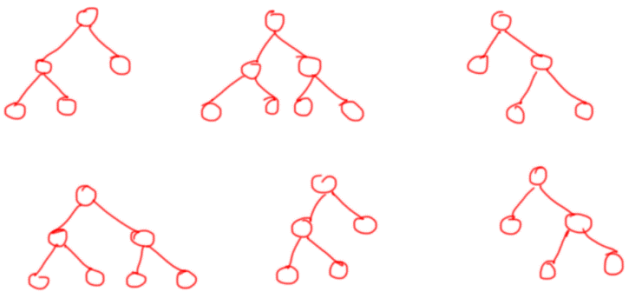
Random forest dựa vào nguyên lý khá đơn giản "The Wisdom of Crowds". Tức ý kiến đám đông sẽ đáng tin cậy hơn so với ý kiến cá nhân, kết quả dự đoán từ nhiều model sẽ đáng tin cậy hơn so với kết quả của 1 model. Random forest mang lại hiệu quả rất tốt trong thực nghiệm so với các mô hình machine learning truyền thống cùng tác vụ: Logistic regression, decision tree,...

Trong phần này, ta sẽ tìm hiểu random forest cho bài toán classification.

2. *Cách xây dựng Random Forest dựa vào luật kết hợp (ensembling)*

Ensembling là việc kết hợp nhiều mô hình (thuật toán) đơn giản lại với nhau, để tạo ra mô hình mới có độ chính xác cao và ổn định hơn so với từng mô hình riêng lẻ.

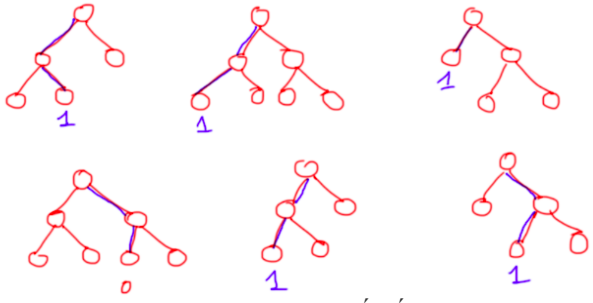
Thuật toán Random Forest sẽ bao gồm nhiều Decision tree, mỗi tree được xây dựng trên các tập dữ liệu và tập thuộc tính khác nhau chọn ngẫu nhiên từ dataset gốc.



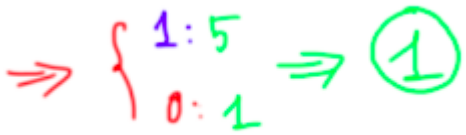
Hình 20. Xây dựng các decision tree trong mô hình Random forest.

Khi predict model, ta sẽ predict trên từng tree, sau đó kết hợp các kết quả lại bằng hình thức voting. Tức là model sẽ

"vote" cho class mà nó predict ra, class nào được vote nhiều nhất thì sẽ là kết quả dự đoán của mô hình random forest.



Hình 21. Predict sample trên tất tất cả các tree.



Hình 22. Tổng hợp kết quả từ các Decision tree.

Vấn đề tiếp theo là chọn dữ liệu train cho mỗi tree như thế nào?

3. *Chọn dữ liệu bằng kĩ thuật Boostrapping & train các Decision tree*

Bootstrap method là phương pháp lấy mẫu có hoàn lại (sampling with replacement). Phương pháp lấy mẫu có hoàn lại có nghĩa là một cá thể có thể xuất hiện nhiều lần trong một lần lấy mẫu. Giả sử ta có 5 quan sát (observation) được đánh nhãn A,B,C,D và E trên 5 quả bóng và bỏ tất cả chúng vào trong 1 cái giỏ.

Hình 23. Kĩ thuật boostrapping.

Từ 5 quan sát này, ta lấy ra 1 quả bóng từ giỏ một cách ngẫu nhiên và ghi lại nhãn của chúng, sau đó bỏ lại quả bóng vừa bốc được vào giỏ và tiếp tục lấy ra một quả bóng một cách ngẫu nhiên, ghi lại nhãn của bóng và bỏ lại quả bóng vào trong giỏ và tiếp tục thực hiện việc lấy mẫu như vậy cho đến khi kết thúc. Việc lấy mẫu này gọi là lấy mẫu có hoàn lại.

Kết quả của việc lấy mẫu như trên có thể như sau (giả sử kích thước mẫu là 10): C, D, E, E, A, B, C, B, A, E. Các quan sát có thể lặp lại trong mẫu và đó là đặc trưng của Bootstrap method.

Giả sử dataset của mình có n sample và mỗi sample có d feature. Để xây dựng một decision tree trong mô hình random forest, ta thực hiện 3 bước sau:

B1: Lấy ngẫu nhiên n sample dữ liệu với kĩ thuật boostrapping. Ta được dataset A.

B2: Chọn ngẫu nhiên k thuộc tính (k < d) của A làm dữ liệu train Decision tree.

B3: Train Decision tree

4. *Ưu & nhược điểm của Random forest*

a) *Ưu điểm*

- Nếu như mô hình Decision tree thường nhạy cảm với dữ liệu *outliers* thì mô hình *Random forest* được huấn luyện trên nhiều tập dữ liệu con khác nhau, trong đó có những tập được loại bỏ dữ liệu *outliers*, điều này giúp cho mô hình ít bị nhạy cảm với dữ liệu *outliers* hơn.
- Decision tree dễ bị overfit khi ta không giới hạn độ sâu cho cây, dẫn đến việc dự đoán kết quả mang tính chủ quan. Random forest giống như phiên bản upgrade của decision tree, việc sử dụng nhiều cây làm cho xác xuất mỗi cây bị overfit giảm, giúp model predict ổn định hơn, chính ít gặp overfit.
- Đối với dữ liệu nhiều chiều, Random forest càng thể hiện sức mạnh. Do mỗi tree được học trên các bộ dữ liệu có số features khác nhau, các tree sẽ có các "góc nhìn" khác nhau, tạo nên sự tổng quát (generate) cho model.

b) *Nhược điểm*

- Nó giống mô hình blackbox, ta gần như không can thiệp được cấu trúc model, data nào, features nào để train cho decision tree. Nó làm cho ta có cảm giác may rủi.
- Tốn nhiều tài nguyên tính toán. Đặt biệt nếu trong model có nhiều cây, thời gian predict chậm hơn rất nhiều so với các thuật toán như linear regression, SVM,...

E. *Naïve Bayes:*

1. *Naives Bayes Classifier:*

Xét bài toán phân loại với C classes (1,2,3,4,...,C). Giả sử điểm dữ liệu $\mathbf{x} \in \mathbf{Rd}$ .Hãy tính xác suất để điểm dữ liệu này rơi vào class c:

$$p(y=c|\mathbf{x}) \quad \textbf{(1)}$$

Viết gọn là p(c|**x**). Tức là tính xác suất điểm dữ liệu rơi vào lớp c với 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:

$$c = \arg\max_{c \in \{1,..,C\}} p(c|\mathbf{x}) \quad (2)$$

Ta sẽ sử dụng công thức bayes để tính nó :

$$c = \arg\max_c p(c|\mathbf{x}) \ (3) \ = \arg\max_c \frac{p(\mathbf{x}|c)p(c)}{p(\mathbf{x})} \ 4) \ = \arg\max_c p(\mathbf{x}|c)p(c) \ (5)$$

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 tỉ lệ số điểm dữ liệu trong tập training rơi vào class này chia cho tổng số lượng dữ liệu trong tập training.

Thành phần còn lại p(x|c), tức phân phối của các điểm dữ liệu trong class , thường rất khó tính toán vì 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 là độc lập nhau, nếu biết c(given c). Tức là:

$$p(\mathbf{x}|c) = p(x_1, x_2, \ldots, x_d|c) = \prod_{i=1}^{d} p(x_i|c) \quad (6)$$

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)*.

Ở bước **training**, các phân phối p(c) và p(xi|c) , i=1,2,...d,với mọi $c \in y$, $\mathbf{x} \in X$ sẽ được xác định dựa vào training data.(trong mỗi cột feature, nếu là data rời rạc ta phải có bảng

phân phối xác suất xi,c, nếu data liên tục thì phải xác định được hàm phân phối xác suất,trong mỗi điểm data,tại mỗi thuộc tính data sẽ xác định được p(xi|c)) .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 định bởi:

$$c = \arg\max_{c\in\{1,\ldots,C\}} p(c)\prod_{i=1}^{d} p(x_i|c) \quad (7)$$

Trong công thức (7) biểu thức trong argmax sẽ là một số rất nhỏ nếu d lớn và các xác suất có giá trị nhỏ gây sai số cho việc tính toán. Nên sẽ lấy log của biểu thức trong dấu argmax.

$$c=\text{argmax}\,(\log(p(c)) + \sum_{i=1}^{d}\log(p(xi|c))) \quad c\in\{1,2,..,C\}$$

Việc xác định p(xi|c) phụ thuộc vào loại dữ liệu. Có 3 loại được sử dụng phổ biến là: Gaussian Naive Bayes, Multinomial Naive Bayes, và Bernoulli Naive Bayes .

a) Gaussian Naive Bayes:

Mô hình này được sử dụng chủ yếu trong loại dữ liệu mà các thành phần là các biến liên tục.

Với mỗi chiều dữ liệu i và một class c, xi tuân theo một phân phối chuẩn có kỳ vọng μ_{ci} và phương sai σ_{ci}^2 :

$$p(x_i|c) = p(x_i|\mu_{ci},\sigma_{ci}^2) = \frac{1}{\sqrt{2\pi\sigma_{ci}^2}}\exp\left(-\frac{(x_i-\mu_{ci})^2}{2\sigma_{ci}^2}\right) \quad (8)$$

Trong đó, bộ tham số $\theta = \{\mu_{ci},\sigma_{ci}^2\}$được xác định bằng Maximum Likelihood:

$$(\mu_{ci},\sigma_{ci}^2) = \arg\max_{\mu_{ci},\sigma_{ci}^2}\prod_{n=1}^{N} p(x_i^{(n)}|\mu_{ci},\sigma_{ci}^2) \quad (9)$$

b) Multinomial Naive Bayes:

Mô hình này chủ yếu được sử dụng trong phân loại văn bản mà feature vectors được tính bằng Bags of Words. Lúc này, mỗi văn bản được biểu diễn bởi một vector có độ dài d chính là số từ trong từ điển. Giá trị của thành phần thứ i trong mỗi vector chính là số lần từ thứ i xuất hiện trong văn bản đó.

Khi đó, p(xi|c) tỉ lệ với tần suất từ thứ i (hay feature thứ i cho trường hợp tổng quát) xuất hiện trong các văn bản của class c. Giá trị này có thể được tính bằng cách:

$$\lambda_{ci} = p(x_i|c) = \frac{N_{ci}}{N_c} \quad (10)$$

+*N*ci là tổng số lần từ thứ i xuất hiện trong các văn bản của class c, nó được tính là tổng của tất cả các thành phần thứ i của các feature vectors ứng với class c.

+*N*c là tổng số từ (kể cả lặp) xuất hiện trong class c. Nói cách khác, nó bằng tổng độ dài của toàn bộ các văn bản thuộc vào class c. Có thể suy rằng:

$N_c = \sum_{i=1}^{d} N_{ci}$, từ đó $\sum_{i=1}^{d}\lambda_{ci} = 1$.

Cách tính này có một hạn chế là nếu có một từ mới chưa bao giờ xuất hiện trong class c thì biểu thức (10) sẽ bằng 0, điều này dẫn đến vế phải của (7) bằng 0 bất kể các giá trị còn lại có lớn thế nào. Việc này sẽ dẫn đến kết quả không chính xác.

Để giải quyết việc này, một kỹ thuật được gọi là *Laplace smoothing* được áp dụng:

$$\hat{\lambda}_{ci} = \frac{N_{ci}+\alpha}{N_c + d\alpha} \quad (11)$$

Với α là một số dương, thường bằng 1, để tránh trường hợp tử số bằng 0. Mẫu số được cộng với dα để đảm bảo tổng xác suất $\sum_{i=1}^{d}\hat{\lambda}_{ci} = 1$.

Như vậy, mỗi class c sẽ được mô tả bởi bộ các số dương có tổng bằng 1:

$$\hat{\lambda}_c = \{\hat{\lambda}_{c1},\ldots,\hat{\lambda}_{cd}\}.$$

c) Bernoulli Naive Bayes:

Mô hình này được áp dụng cho các loại dữ liệu mà mỗi thành phần là một giá trị binary - bằng 0 hoặc 1.Ví dụ: cũng với loại văn bản nhưng thay vì đếm tổng số lần xuất hiện của 1 từ trong văn bản, ta chỉ cần quan tâm từ đó có xuất hiện hay không.

Khi đó p(xi|c) được tính bằng:

$$p(x_i|c) = p(i|c)^{x_i}(1-p(i|c)^{1-x_i}$$

+với p(i|c) có thể được hiểu là xác suất từ thứ i xuất hiện trong các văn bản của class c.

Ví dụ nhỏ áp dụng Naive bayes:

Dựa vào các thuộc tính của thời tiết để đưa ra gợi ý đi chơi hay không.

	outlook	temperature	humidity	windy	Play gofl
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

Hình 24. Data huấn luyện ví dụ

Ta sẽ tính p(xi|c) cho mỗi cặp xi,c (xi là giá trị của 1 thuộc tính tại 1 điểm dữ liệu) như bảng dưới:

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Giả sử thời tiết hôm nay là: today=Sunny,Hot,Normal,False hãy đưa ra gợi ý nên đi chơi hay không.

$$P(yes|today) = \frac{P(sunnyoutlook|yes)P(hottemperature|yes)P(normalhumidity|yes)P(nowind|yes)P(yes)}{P(today)}$$

$$P(no|today) = \frac{P(sunnyoutlook|no)P(hottemperature|no)P(normalhumidity|no)P(nowind|no)P(no)}{P(today)}$$

P(today) là như nhau trong 2 công thức trên, ta coi nó bằng A

p(yes|today)=(2/9 *2/9 *6/9 *6/9 *9/14)/A=0.0141/A

p(no|today)=(3/5 * 2/5 * 1/5 *2/5 *2/14)/A=0.0068/A

Ta chuyển chúng thành xác suất:

p(yes|today)=p(yes|today)/(p(yes|today)+p(no|today))=0.67

p(yes|today)=p(no|today)/(p(yes|today)+p(no|today))=0.33

Vậy hôm nay nên đi chơi.

2. Ưu nhược điểm của Naive Bayes:

a) Ưu điểm.

- 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.
- 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) trở nên cực kỳ nhanh.Việc này giúp nó mang lại hiệu quả cao trong các bài toán large-scale.
- Cho phép kết hợp tri thức tiền nghiệm (prior knowledge) và dữ liệu quan sát được (obserwed data).Tốt khi có sự chệnh lệch số lượng giữa các lớp phân loại.

b) Nhược điểm.

- Độ chính xác của Naive Bayes trong các dữ liệu thực tế thường không cao bằng các thuật toán phân loại phức tạp khác, do giả thiết các feature dữ liệu độc lập nhau là thường không đúng với thực tế.Giả định độc lập (ưu điểm cũng chính là nhược điểm) hầu hết các trường hợp thực tế trong đó có các thuộc tính trong các đối tượng thường phụ thuộc lẫn nhau.
- Khi dữ liệu không nằm trong data train thì model không thể đưa ra kết quả dự đoán.
- Vấn đề zero (đã nêu cách giải quyết ở phía trên).
- Mô hình không được huấn luyện bằng phương pháp tối ưu mạnh và chặt chẽ. Tham số của mô hình là các ước lượng xác suất điều kiện đơn lẻ. Không tính đến sự tương tác giữa các ước lượng này.

F. AdaBoost:

Boosting là một trong những phương pháp học Ensemble cố gắng xây dựng một bộ phân loại mạnh từ tập hợp các bộ phân loại yếu. Ý tưởng chung của phương pháp Boosting là huấn luyện tuần tự các bộ dự đoán, mô hình sau sẽ cố gắng sửa lỗi cho mô hình trước. Quá trình này được thực hiện liên tiếp cho đến khi mô hình đạt được dự đoán hoàn hảo hoặc là đã đạt tới ngưỡng tối đa số lượng mô hình cho phép. Hiện nay, đã có rất nhiều phương pháp Boosting như: AdaBoost, Gradient Boosting,…

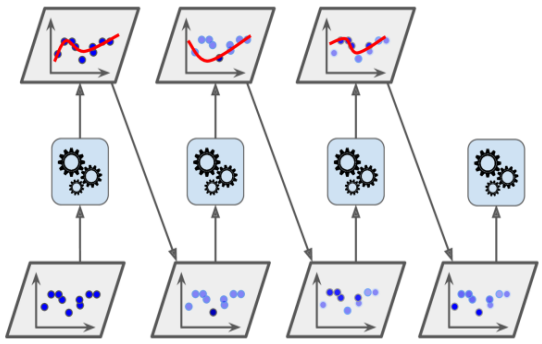
AdaBoost là thuật toán Boosting đầu tiên thành công được phát triển để giải quyết bài toán phân loại nhị phân. Nó được xây dựng bởi Yoav Freund và Robert Schapire và họ cũng đã giành được Giải thưởng Gödel năm 2003 nhờ thuật toán này.

1. Ý tưởng

Để bộ phân loại sau có thể sửa những sai lầm của bộ phân loại trước nó, Yoav Freund và Robert Schapire đã đề xuất một giải pháp đó chính là tập trung hơn vào những các mẫu huyến luyện mà bộ dự đoán trước bị dưới khớp. Điều này sẽ giúp cho mô hình chú ý vào các trường hợp khó.

2. Ví dụ

Khi huấn luyện một bộ phân loại AdaBoost với bộ dữ liệu Heart Failure, đầu tiên thuật toán sẽ chọn một mô hình cở sở làm gốc (Cây Quyết Định) và sử dụng mô hình này để thực hiện trên toàn tập bộ dữ liệu. Sau đó thuật toán sẽ tăng trọng số cho những mẫu dự đoán sai. Tiếp theo thực hiện huấn luyện trên bộ phân loại thứ 2 với trọng số đã cập nhật, thực hiện liên tục như thế cho đến khi đạt điều kiện dừng.



Hình 25. Minh họa thuật toán AdaBoost

3. Phân tích thuật toán

Ban đầu, thuật toán sẽ gán trọng số cho mỗi điểm là như nhau:

$$wi = \frac{1}{N}, \forall i = 1, N$$

Giả sử bộ dữ liệu Heart Failure có 7 điểm dữ liệu (N = 7):

Age	Sex	....	HeartDisease	Sample Weights
40	1		0	1/7
49	1		1	1/7
12	0		1	1/7
30	0		0	1/7
56	1		0	1/7
14	0		0	1/7
90	1		0	1/7

Hình 26. Khởi tạo trọng số ban đầu

Bộ dự đoán đầu tiên được huấn luyện và cho ra kết quả predict đầu tiên:

Age	Sex	....	HeartDisease	Sample Weights	Predict
40	1		0	1/7	0
49	1		1	1/7	0
12	0		1	1/7	1
30	0		0	1/7	1
56	1		0	1/7	0
14	0		0	1/7	0
90	1		0	1/7	0

Hình 27. Kết quả của bộ dự đoán đầu tiên

Ta thấy rằng ở dự đoán đầu tiên, có 2 mẫu dữ liệu được dự đoán sai, vì vậy tỉ lệ lỗi của bộ dự đoán này sẽ là: $r_1 = \frac{2}{7}$

Trọng số của bộ dự đoán sau đó sẽ được tính theo công thức:

$$\alpha = \beta \times \log\frac{1-r}{r}$$

Trong đó β chính là tốc độ học (trường hợp này sẽ sử dụng $\frac{1}{2}$, thuật toán AdaBoost gốc không sử dụng tốc độ học). Dễ nhận thấy, nếu bộ dự đoán cho kết quả chính xác càng nhiều điểm dữ liệu, thì trọng số sẽ càng cao. Ngược lại, nếu bộ dự đoán cho kết quả sai nhiều, trọng số sẽ âm.

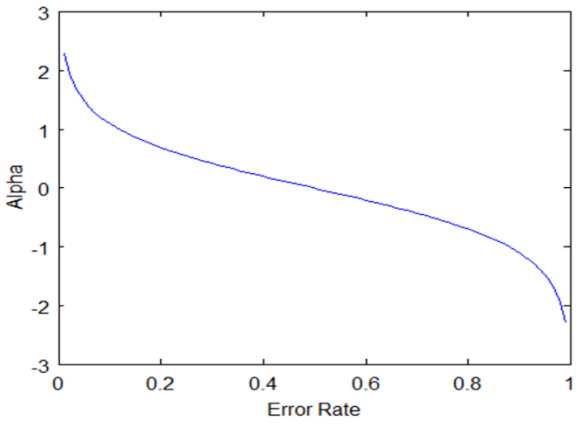
Vậy $\alpha_1 = 0.46$

Tiếp đó, thuật toán AdaBoost sẽ thực hiện cập nhật trọng số cho các điểm dữ liệu theo công thức:

$$w_i = \begin{cases} w_i \times e^{-\alpha}, \text{ khi điểm dữ liệu được dự đoán là đúng} \\ w_i \times e^{\alpha}, \text{ khi điểm dữ liệu được dự đoán là sai} \end{cases}$$

Làm rõ công thức cập nhật trọng số:

Dưới đây là đồ thị thể hiện mối quan hệ giữa α và r:



Hình 28.. Mối quan hệ giữa α và r

Qua đồ thị, thấy rõ 3 trường hợp:

Khi $r = 0.5$ và $\alpha = 0$: Bộ dự đoán này không đóng góp được gì vào quá trình huyến luyện.

Khi $r \to 1$ thì $\alpha \to -\infty$: Bộ phân loại cho kết quả dự đoán sai rất nhiều điểm dữ liệu. Khi đó e^{α} sẽ bé vô cùng, dẫn đến giá trị trọng số khi cập nhật cũng sẽ rất bé. Điều này đi ngược lại ý tưởng của thuật toán *'tăng trọng số cho những mẫu dự đoán sai'*. Giải thích cho vấn đề này, đặt trong trường hợp hầu hết các điểm dữ liệu đều dự đoán sai, vậy thì các điểm dữ liệu đó thực chất lại càng ít quan trọng, vì chúng ta chỉ cần thực hiện một bước đơn giản, *lấy ngược lại kết quả* mà bộ phân loại tồi tệ này đã dự đoán.

Khi $r \to 0$ thì $\alpha \to +\infty$: Bộ phân loại này cho kết quả dự đoán khá tốt. Khi đó e^{α} sẽ lớn vô cùng. Vì thế, đối với những điểm dự liệu được dự đoán đúng, ta cần phải lấy $e^{-\alpha}$ (giá trị rất nhỏ) vì ta không quan tâm đến những điểm dữ liệu này.

Tương tự với những điểm dữ liệu sai, mục đích của ta là cố gắng sửa lỗi những chỗ sai này nên ta cần $e^{+\alpha}$ (giá trị rất lớn) để chú ý đến điểm dữ liệu ấy.

Trọng số sau khi được cập nhật lần đầu tiên:

Age	Sex	....	HeartDisease	Sample Weights	Predict	New Weights
40	1		0	1/7	0	0.09
49	1		1	1/7	0	0.23
12	0		1	1/7	1	0.09
30	0		0	1/7	1	0.23
56	1		0	1/7	0	0.09
14	0		0	1/7	0	0.09
90	1		0	1/7	0	0.09

Hình 29. Trọng số sau khi cập nhật lần đầu

Đây là trọng số chưa được chuẩn hóa, thực hiện bước chuẩn hóa theo công thức để đưa tổng trọng số bằng 1:

$$w_i = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Trọng số cuối cùng sẽ là:

Age	Sex	....	HeartDisease	Sample Weights	Predict	New Weights	Normalize
40	1		0	1/7	0	0.09	0.098901099
49	1		1	1/7	0	0.23	0.252747253
12	0		1	1/7	1	0.09	0.098901099
30	0		0	1/7	1	0.23	0.252747253
56	1		0	1/7	0	0.09	0.098901099
14	0		0	1/7	0	0.09	0.098901099
90	1		0	1/7	0	0.09	0.098901099

Hình 30. Trọng số cuối cùng

Đây chính là đích đến mong muốn của ta ở bộ phân loại đầu tiên. Quá trình này sẽ tiếp tục được lặp đi lặp lại (tính trọng số của bộ dự đoán mới, cập nhật trọng số của mẫu, huấn luyện một bộ dự đoán khác và cứ như vậy). Thuật toán chỉ dừng lại khi đạt được đến số lượng bộ dự đoán mong muốn, hoặc khi tìm được một bộ dự đoán hoàn hảo.

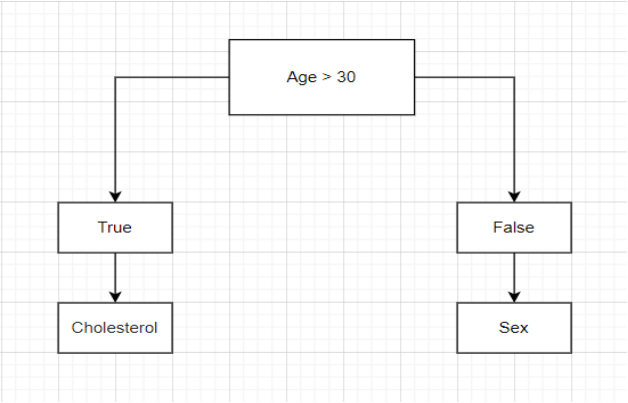
Khi dự đoán, AdaBoost chỉ đơn giản là tính đầu ra của từng bộ dự đoán, rồi gán thêm trọng số của bộ dự đoán đó. Nhãn lớp dự đoán là nhãn có tổng trọng số lớn nhất:

$$\mathrm{P} = argmax_k \sum_{\substack{j=1 \\ Predict(x)_j = k}}^{M} \alpha_j$$ trong đó M là số bộ dự đoán

Điểm khác biệt giữa AdaBoost sử dụng cây quyết định và Random Forest

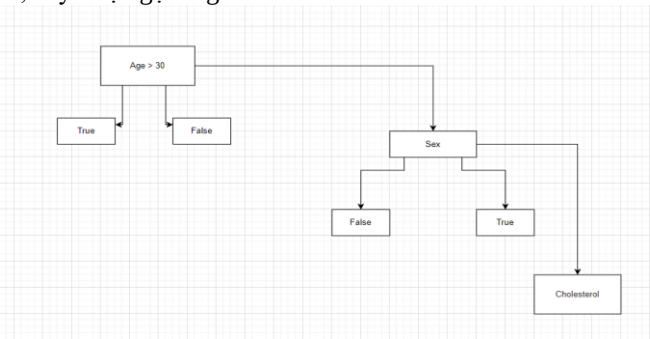
Khi mới tiếp cận, nhìn chung AdaBoost và Random Forest có thể khá giống nhau. Nhưng thật ra, cả 2 lại có rất nhiều điểm khác biệt. Dưới đây là một số điểm khác biệt chính của cả 2 thuật toán:

Kích thước cây: Đối với Random Forest, mỗi khi tạo một cây, ta sẽ tạo một cây hoàn chỉnh.



Hình 31. Cây trong Random Forest

Còn đối với AdaBoost, thường sẽ chỉ khởi tại một cành và 2 lá, đây được gọi là gốc.



Hình 32. Cây trong AdaBoost

Chọn biến: Random Forest với kích thước cây đầy đủ có thể sẽ tận dụng được tất cả các biến để đưa dự đoán. AdaBoost chỉ với một gốc chắc chắn sẽ không bì được với Random Forest khi mà Random Forest sử dụng tất cả các biến để đưa ra dự đoán, còn AdaBoost chỉ sử dụng một biến để đưa ra quyết định, đây được gọi là một người học yếu (Weak Classifier). Tuy nhiên, AdaBoost lại thích điều này, thuật toán sẽ tập hợp nhiều người học yếu lại kết hợp với nhau cho ra một kết quả chính xác hơn.

Tiếng nói của mỗi cây: Với Random Forest, mỗi cây trong rừng đều có tiếng nói như nhau. Với mỗi gốc cây trong AdaBoost thì lại khác, sẽ có gốc có tiếng nói mạnh hơn, cũng có gốc sẽ không có tiếng nói nào (các trường hợp mối quan hệ giữa α và r).

Cách thức thực hiện: Trong Random Forest, mỗi cây được thực hiện dự đoán song song nhau, tức là các cây sẽ độc lập và cây này sẽ không ảnh hưởng cây khác. Trong AdaBoost, do đây là thuật toán thực hiện tuần tự, nên việc chọn gốc là rất quan trọng, vì kết quả trọng số của gốc này sẽ ảnh hưởng rất nhiều đến trọng số của gốc khác.

4. Đánh giá thuật toán

a) Ưu điểm

Dễ sử dụng và ít điều chỉnh tham số.

Khi sử dụng AdaBoost với base model là DecisionTree (1-level) sẽ ít khả năng dẫn đến hiện tượng overfitting. Tuy nhiên, khi sử dụng các Weak Classifiers phức tạp sẽ khiến mô hình dễ rơi vào tình trạng này.

Đã được phát triển để phân loại hình ảnh và văn bản.

b) Nhược điểm

Thuật toán phụ thuộc khá nhiều vào bộ dữ liệu. Nó đòi hỏi bộ dữ liệu phải sạch và ít điểm dữ liệu gây nhiễu vì ý tưởng chính của AdaBoost là fit từng điểm dữ liệu một cách hoàn hảo.

Một điểm trừ lớn của thuật toán này là sử dụng phương pháp học tuần tự. Ta chỉ có thể huấn luyện bộ dự đoán mới chỉ khi đã hoàn thành huấn luyện và đánh giá bộ dự đoán trước đó. Vì vậy, nhìn chung kĩ thuật này sẽ không mở rộng tốt bằng các thuật toán Ensemble khác (bagging, pasting).

Do đây là thuật toán cơ bản trong nhánh Boosting nên ngoài ra có những thuật toán trong nhánh này đạt hiệu quả và tốc độ nhanh hơn như XGBoost,...

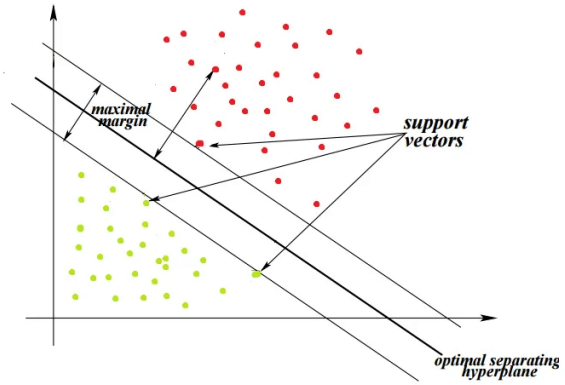
G. Support vector machine (SVM)

1. Giới thiệu sơ lược về SVM

Support Vector Machines là một trong số những thuật toán phổ biến và được sử dụng nhiều nhất trong học máy trước khi mạng nơ ron nhân tạo trở lại với các mô hình deep learning. Nó được biết đến rộng rãi ngay từ khi mới được phát triển vào những năm 1990.

Mục tiêu của SVM là tìm ra một siêu phẳng trong không gian N chiều (ứng với N đặc trưng) chia dữ liệu thành hai phần tương ứng với lớp của chúng. Nói theo ngôn ngữ của đại số tuyến tính, siêu phẳng này phải có lề cực đại và phân chia hai bao lồi và cách đều chúng.

Để phân chia hai lớp dữ liệu, rõ ràng là có rất nhiều siêu phẳng có thể làm được điều này. Mặc dù vậy, mục tiêu của chúng ta là tìm ra siêu phẳng có lề rộng nhất tức là có khoảng cách tới các điểm của hai lớp là lớn nhất

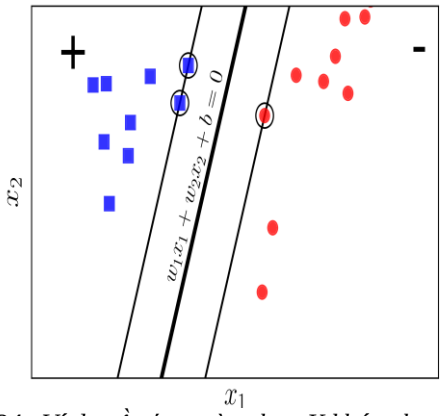


Hình 33. Ví dụ về các trường hợp K khác nhau ở KNN

Lưu ý: Số chiều của siêu phẳng phụ thuộc vào số đặc trưng

2. Xây dựng bài toán tối ưu cho SVM:

Giả sử rằng các cặp dữ liệu của training set là (x_1, y_1), (x_2, y_2), …, (x_N, y_N) với vector $x_i \in R^d$ thể hiện đầu vào của một điểm dữ liệu và y_i là nhãn của điểm dữ liệu đó. d là số chiều của dữ liệu và N là số điểm dữ liệu. Giả sử rằng nhãn của mỗi điểm dữ liệu được xác định bởi $y_i = 1$ (class 1) hoặc $y_i = -1$ (class 2).



Hình 34.. Ví dụ về các trường hợp K khác nhau ở KNN

Giả sử rằng các điểm vuông xanh thuộc class 1, các điểm tròn đỏ thuộc class -1 và mặt $w^Tx + b = w_1x_1 + w_2x_2 + b = 0$ là mặt phân chia giữa hai classes.

Khoảng cách từ điểm (x_n, y_n) đến mặt phân chia là:

$$\frac{y_n(w^Tx_n + b)}{\|w\|_2}$$

$$\Rightarrow \text{margin} = \min \frac{y_n(w^Tx_n + b)}{\|w\|_2}$$

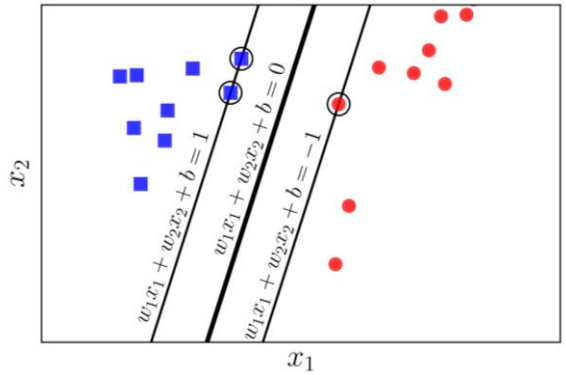
Bài toán tối ưu trong SVM chính là bài toán tìm w và b sao cho margin này đạt giá trị lớn nhất:

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \left\{ \min_n \frac{y_n(\mathbf{w}^T\mathbf{x}_n + b)}{||\mathbf{w}||_2} \right\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||_2} \min_n y_n(\mathbf{w}^T\mathbf{x}_n + b) \right\} \quad (1)$$

Nếu ta thay vector hệ số w bởi kw và b bởi kb trong đó k là một hằng số dương thì mặt phân chia không thay đổi, tức khoảng cách từ từng điểm đến mặt phân chia không đổi, tức margin không đổi. Dựa trên tính chất này, ta có thể giả sử:

$$y_n(w^Tx_n + b) = 1$$

$$\Rightarrow d_+ = d_- = \frac{1}{\|w\|_2}$$



Hình 35. Ví dụ về các trường hợp K khác nhau ở KNN

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

Như vậy, với mọi n, ta có:

$$y_n(w^Tx_n + b) \geq 1$$

Vậy bài toán tối ưu (1) có thể đưa về bài toán tối ưu có ràng buộc sau đây:

$$(\mathbf{w}, b) = \arg\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||_2} \text{ subject to: } y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \forall n = 1, 2, \ldots, N \quad (2)$$

Lấy nghịch đảo hàm mục tiêu, bình phương nó để được một hàm khả vi, và nhân với $\frac{1}{2}$, ta được:

$$(\mathbf{w}, b) = \arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||_2^2 \text{ subject to: } 1 - y_n(\mathbf{w}^T\mathbf{x}_n + b) \leq 0, \forall n = 1, 2, \ldots, N \quad (3)$$

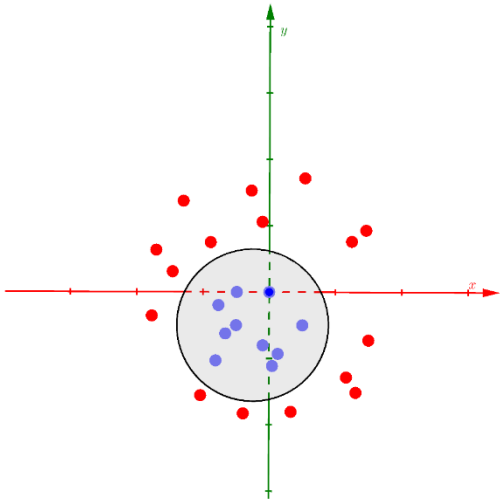
Trong bài toán (3), hàm mục tiêu là một norm, nên là một hàm lồi. Các hàm bất đẳng thức ràng buộc là các hàm tuyến tính theo w và b, nên chúng cũng là các hàm lồi. Vậy bài toán tối ưu (3) có hàm mục tiêu là lồi, và các hàm ràng buộc cũng là lồi, nên nó là một bài toán lồi. Hơn nữa, nó là một Quadratic Programming. Thậm chí, hàm mục tiêu là strictly convex vì $\|w\|_2 = w^TIw$ và I là ma trận đơn vị - là một ma trận xác định dương. Từ đây có thể suy ra nghiệm cho SVM là duy nhất.

Đến đây thì bài toán này có thể giải được bằng các công cụ hỗ trợ tìm nghiệm cho Quadratic Programing, ví dụ CVXOPT.

3. Giới thiệu về Kernel SVM:

Ý tưởng cơ bản của Kernel SVM và các phương pháp kernel nói chung là tìm một phép biến đổi sao cho dữ liệu ban đầu là không phân biệt tuyến tính được biến sang không gian mới. Ở không gian mới này, dữ liệu trở nên phân biệt tuyến tính.

Ở ví dụ trong slide trước, chúng ta đã coi chiều không gian thứ ba là một hàm số của hai chiều còn lại $z = x^2 + y^2$. Các điểm dữ liệu được phân bố trên 1 parabolic và đã trở nên phân biệt tuyến tính. Giao điểm của mặt phẳng tìm được và mặt parabolic là một đường ellipse, khi chiếu toàn bộ dữ liệu cũng như đường ellipse này xuống không gian hai chiều ban đầu, ta đã tìm được đường phân chia hai classses.



Hình 36. Ví dụ về các trường hợp K khác nhau ở KNN

Nói một cách ngắn gọn, Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu x từ không gian feature ban đầu thành dữ liệu trong một không gian mới bằng hàm số $\Phi(x)$. Trong ví dụ này, hàm $\Phi(x)$ đơn giản là giới thiệu thêm một chiều dữ liệu mới (một feature mới) là một hàm số của các features đã biết. Hàm số này cần thỏa mãn mục đích của chúng ta: trong không gian mới, dữ liệu giữa hai classes là phân biệt tuyến tính hoặc gần như phân biệt tuyến tính.

4. Những ràng buộc về bài toán Kernel SVM:

Không phải hàm k() bất kỳ nào cũng được sử dụng. Các hàm kerrnel cần có các tính chất:

+ Đối xứng: k(x, z) = k(z, x)
+ Thỏa mãn điều kiện Mercer:

$$\sum_{n=1}^{N}\sum_{m=1}^{N} k(\mathbf{x}_m, \mathbf{x}_n)c_nc_m \geq 0, \quad \forall c_i \in \mathbb{R}, i = 1, 2, \ldots, N$$

Từ những ràng buộc trên ta suy ra bài toán tối ưu phải là lồi và hàm mục tiêu là một hàm lồi (một quadratic form).

5. Dataset của nhóm

Bộ dataset được nhóm chọn là dự đoán bệnh suy tim, với 11 thuộc tính đầu vào, 1 nhãn gồm 2 giá trị Yes hoặc No và 918 dòng dữ liệu.

Với bộ dataset này thì ta chưa biết được rằng nó có phân biệt tuyến tính hay không nên tôi đã chọn làm bằng Linear SVM bình thường và SVM với Kernel là Polynomial

a) Linear:

Đây là trường hợp đơn giản với kernel chính tích vô hướng của hai vector:

$$k(x, z) = x^T z$$

Khi sử dụng hàm **sklearn.svm.SVC**, kernel này được chọn bằng cách đặt **kernel = 'linear'**

b) Polynomial:

$$k(x, z) = (r + \gamma x^T z)^d$$

Với d là một số dương để chỉ bậc của đa thức. d có thể không là số tự nhiên vì mục đích chính của ta không phải là bậc của đa thức mà là cách tính kernel. Polynomial kernel có thể dùng để mô tả hầu hết các đa thức có bậc không vượt quá d nếu d là một số tự nhiên.

Khi sử dụng thư viện sklearn, **kernel = 'poly'**

6. Ưu và nhược điểm

Ưu điểm

Là một kĩ thuật phân lớp khá phổ biến, SVM thể hiện được nhiều ưu điểm trong số đó có việc tính toán hiệu quả trên các tập dữ liệu lớn. Có thể kể thêm một số ưu điểm của phương pháp này như:

- 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 dịnh
- 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.

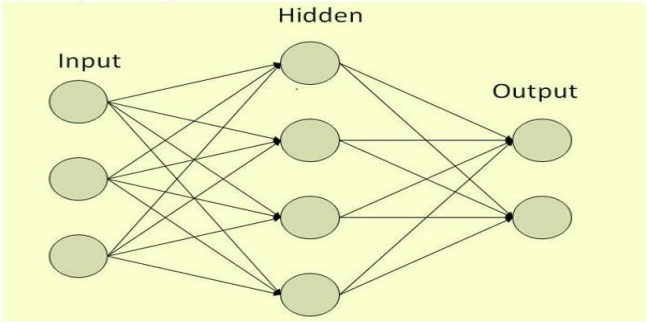
H. Neural Network

1. Giới thiệu sơ lược về Neural Network

Là một mô hình tính toán bắt chước cách thức hoạt động của các tế bào thần kinh trong não người. Mạng nơ-ron nhân tạo (ANN) sử dụng các thuật toán learning có thể thực hiện các điều chỉnh một cách độc lập – hoặc học theo một nghĩa nào đó – khi chúng nhận được giá trị input mới.

Mạng Neural Network là sự kết hợp của những tầng perceptron hay còn gọi là perceptron đa tầng. Và mỗi một mạng Neural Network thường bao gồm 3 kiểu tầng là:

- Tầng input layer (tầng vào): Tầng này nằm bên trái cùng của mạng, thể hiện cho các đầu vào của mạng.
- Tầng output layer (tầng ra): Là tầng bên phải cùng và nó thể hiện cho những đầu ra của mạng.
- Tầng hidden layer (tầng ẩn): Tầng này nằm giữa tầng vào và tầng ra nó thể hiện cho quá trình suy luận logic của mạng.



Hình 37. Mô hình Ann đơn giản

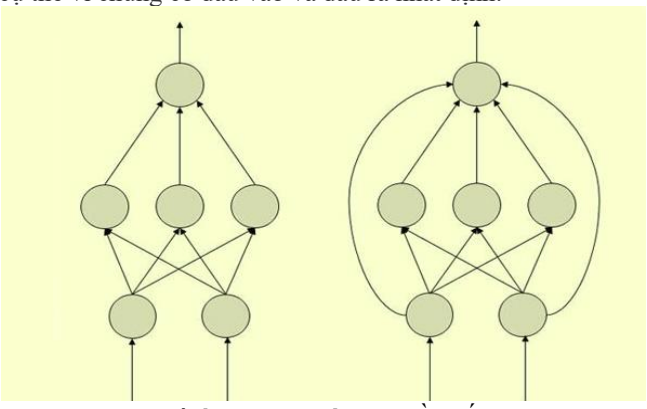
Lưu ý: Mỗi một Neural Network chỉ có duy nhất một tầng vào và 1 tầng ra nhưng lại có rất nhiều tầng ẩn.

Trong ANN, trừ input layer thì tất cả các node thuộc các layer khác đều full-connected với các node thuộc layer trước nó. Mỗi node thuộc hidden layer nhận vào ma trận đầu vào từ layer trước và kết hợp với trọng số để ra được kết quả.

2. Các loại mạng trong Neural Network.

a) Mạng lan truyền tiến (Feed Forward)

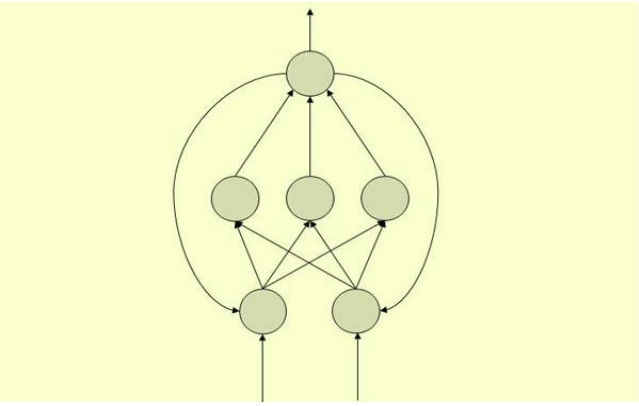
Mạng lan truyền tiến (FeedForward) có luồng thông tin 1 chiều, một đơn vị sẽ được sử dụng để gửi thông tin cho một đơn vị khác mà không nhận được bất kỳ thông tin nào. Chúng không có vòng phản hồi, thường dùng để nhận dạng một mẫu cụ thể vì chúng có đầu vào và đầu ra nhất định.



Hình 38. Mạng lan truyền tiến

b) Mạng lan truyền ngược (Back Forward)

Trong mạng Nơron nhân tạo này, chúng sẽ cho phép các vòng lặp phản hồi. Chúng ta thường sử dụng mô hình này trong các bộ nhớ có thể giải quyết nội dung.



Hình 39. Mô hình Feedback ANN

3. Cách hoạt động của Neural Network

Mạng neural nhân tạo có khả năng sử dụng được như một loại cơ chế xấp xỉ hàm tùy ý mà học được từ việc dữ liệu quan sát. Tuy nhiên, việc sử dụng chúng khá khó và cần phải có sự hiểu biết tương đối về những lý thuyết cơ bản về mạng nơron này.

Lựa chọn mô hình: Phụ thuộc vào cách trình bày dữ liệu và các ứng dụng của nó. Đây là mô hình khá phức tạp nên có thể dẫn đến nhiều thách thức cho quá trình học.

Thuật toán học: Thường sẽ có rất nhiều thỏa thuận giữa các thuật toán học. Và hầu hết, chúng sẽ làm việc tốt với những tham số đúng nhằm huấn luyện trên dữ liệu mà không nhìn thấy yêu cầu một số lượng đáng kể các thử nghiệm.

Mạnh mẽ: Nếu như các mô hình, thuật toán học và hàm chi phí được lựa chọn một cách thích hợp thì Neural Network có thể cho ra kết quả vô cùng hợp lý.

4. Ưu và nhược điểm của Neural Nerwork

Ưu điểm:

- Phân tích dữ liệu một cách trực quan hiệu quả: Mạng nơ ron nhân tạo tương tự như mạng thần kinh của con người nên nó có thể xử lý một số các tác vụ phức tạp hơn so với các phương pháp khác. Mạng neural có thể phân tích các dữ liệu một cách hiệu qủa và tách chúng thành nhiều loại khác nhau.
- Xử lý hiệu quả các dữ liệu không có cấu trúc.
- Cấu trúc có tính thích ứng: bất kỳ mục đích nào mà ANN được áp dụng, nó sẽ thay đổi tiến trình cấu trúc của nó theo mục đích đó.

Nhược điểm:

- Yêu cầu phần cứng.
- Cấu trúc phức tạp và khó triển khai.
- Cần nhiều dữ liệu: Dữ liệu sẽ tỉ lệ thuận với độ chính xác.

5. Ứng dụng của Neural Network

Mạng ANN hiện nay được ứng dụng rất nhiều trong rất nhiều ngành, nó có thể làm những việc mà con người gặp khó khăn một cách rất dễ dàng. Một số lĩnh vực sử dụng mạng nơron nhân tạo như: Hàng không vũ trụ, phần mềm, giao thông vận tải, viễn thông.

IV. PROTOCOL ĐÁNH GIÁ

A. Cross Validation

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

Khi k bằng với số lượng phần tử trong tập *training* ban đầu, tức mỗi tập con có đúng 1 phần tử, ta gọi kỹ thuật này là **leave-one-out**.

B. Thang đo đánh giá

1. Thang đo Accuracy.

Khi xây dựng mô hình phân loại chúng ta sẽ muốn biết một cách khái quát tỷ lệ các trường hợp được dự báo đúng trên tổng số các trường hợp là bao nhiêu. Tỷ lệ đó được gọi là độ chính xác. Độ chính xác giúp ta đánh giá hiệu quả dự báo của mô hình trên một bộ dữ liệu. Độ chính xác càng cao thì mô hình của chúng ta càng chuẩn xác.

Công thức:

$$Accuracy = \frac{TP + TN}{total\ sample}$$

Trong các metrics đánh giá mô hình phân loại thì độ chính xác là metric khá được ưa chuộng vì nó có công thức tường minh và dễ diễn giải ý nghĩa. Tuy nhiên hạn chế của nó là đo lường trên *tất cả* các nhãn mà không quan tâm đến độ chính xác trên từng nhãn. Do đó nó không phù hợp để đánh giá những tác vụ mà *tầm quan trọng* của việc dự báo các nhãn không còn như nhau. Hay nói cách khác, như trong ví dụ phân loại nợ xấu, việc chúng ta phát hiện đúng một hồ sơ nợ xấu quan trọng hơn việc chúng ta phát hiện đúng một hồ sơ thông thường.

2. Thang đo Precision và Recall.

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

Xét bài toán phân loại nhị phân, ta xem một lớp là positive, một lớp là negative.

Với một cách xác định một lớp là ***positive*** :

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

Công thức :

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Với hai thang đo ta phải đánh đổi giữa độ lớn giữa 2 thang đo này. Sự đánh đổi giữa precision và recall khiến cho kết quả của mô hình thường là : precision cao, recall thấp hoặc precision thấp, recall cao. Khi đó rất khó để lựa chọn đâu là một mô hình tốt vì không biết rằng đánh giá trên precision hay recall sẽ phù hợp hơn. Chính vì vậy chúng ta sẽ tìm cách kết hợp cả precision và recall trong một chỉ số mới, đó chính là f1 score.

3. Thang đo F1 Score.

F1 Score là trung bình điều hòa giữa precision và recall. Do đó nó đại diện hơn trong việc đánh giá độ chính xác trên đồng thời precision và recall.

Công thức:

$$\frac{2}{F_1} = \frac{1}{precision} + \frac{1}{recall}$$

$$hay\ F_1 = 2\frac{1}{\frac{1}{precision} + \frac{1}{recall}} = 2\frac{precision \cdot recall}{precision + recall}$$

F1 score có giá trị nằm trong nửa khoảng (0,1]. F1 càng cao thì bộ phân lớp càng tốt. Khi cả recall và precision đều bằng 1 thì F1 = 1.

4. Thang đo F1 macro

Macro-average precision là trung bình cộng của các precision theo class, tương tự với Macro-average recall.

F1 macro là trung bình cộng không trọng số của F1 score của từng class.

Công thức:

$$Macro\ F1\ score\ = \frac{sum(F1\ scores)}{number\ of\ class}$$

V. QUÁ TRÌNH HUẤN LUYỆN

Accuracy, Precision, Recall, F1 (macro) sẽ là 4 thang đo chúng tôi sử dụng để đánh giá quá trình huấn luyện các mô hình. Tại bước này, chúng tôi sẽ thực hiện huấn luyện và kiểm thử các mô hình để xuất ở mỗi phương pháp bằng phương pháp đánh giá mô hình *Kfold Cross Validation (với CV=5, shuffle=False)* bằng công cụ hỗ trợ KFold của thư viện Sklearn.

Đối với những bài toán dự đoán bệnh tật, yêu cầu của bài toán thường phải là dự đoán được số lượng trường hợp có bệnh nhiều nhất hơn là số dự đoán trường hợp có bệnh đúng. Có nghĩa là ta cần ưu tiên độ phủ (recall) hơn là chuẩn xác (accuracy) khi thực hiện dự đoán.

Do đặc trưng của bài toán dự đoán suy tim của chúng tôi cũng là bài toán dự đoán bệnh tật, chính vì điều đó chúng tôi sẽ ưu tiên chọn Recall làm than đo chính để so sánh hiệu quả của mô hình, ngoài ra đi kèm theo đó chúng tôi cũng sẽ ưu tiên kết quả đánh giá mô hình theo than đo F1 (macro) bởi nó là trung bình điều hòa của cả 2 than đo Recall và Accuracy.

A. Logistic Regression:

Thực hiện huấn luyện và đánh giá thực nghiệm bộ dữ liệu đã được qua tiền xử lý (loại bỏ đặc trưng có độ tương quan kém, chuẩn hóa dữ liệu bằng phương pháp StandardScaler) bằng 3 mô hình :

+Logisctic Regression với Binary Cross Entropy loss function được xây dựng thủ công

+Logistic Regression với alpha-beta Binary Cross Entropy loss function được tuning hệ số alpha-beta tự động

+Logistic Regression của thư viện sklearn (bao gồm penalty là L2, L1, Elastic Net, và None) được tuning parameters bằng GridSearchCV

1. Thực nghiệm

a) Logistic Regression được xây dựng thủ công với Binary Cross Entropy loss function

Kết quả:

Accuracy	Precision	Recall	F1_Score	Execution time(s)
83.8679	82.4429	81.6461	81.9901	6.5564

Nhận xét:

- Kết quả đánh giá của mô hình với điểm số của cả 4 thang đo tốt.
- Thời gian thực thi quá lâu đối với một thuật toán đơn giản, dễ huấn luyện như Logistic Regression, đây là hạn chế ở việc cài đặt thủ công tối ưu thuật toán chưa thực sự tốt.

b) Logistic Regression với alpha-beta Binary Cross Entropy loss function

KFold Cross Validation sẽ thực hiện chia tập train và test ngẫu nhiên cho mỗi fold nên tỉ lệ phân bố class giữa các fold sẽ khác nhau nên hệ số alpha và beta cần phải xác định độc lập cho mỗi fold. Ta sẽ xác định 2 hệ số này dự trên tỉ lệ nghịch của 2 phân lớp trong tập dữ liệu huấn luyện ở mỗi fold mục đích là để can thiệp vào mức độ ảnh hưởng của 2 nhãn khi huấn luyện.

Ví dụ: Ở fold 1, ta sẽ giảm bớt mức độ ảnh hưởng của lớp 1 bằng cách đánh trọng số alpha=0.67198 và giữ nguyên mức độ ảnh hưởng của lớp 0. Mục đích của việc đánh trọng số này là nhằm giảm mức độ ảnh hưởng của nhãn dễ học (trường hợp này là nhãn 1 do nhãn 1 có số lượng điểm dữ liệu lớn hơn so với nhãn 0)

	Số nhãn 0	Số nhãn 1	alpha, beta
Fold 1	295	439	alpha= 0.67198, beta=1
Fold 2	330	404	alpha= 0.81683, beta=1
Fold 3	375	359	alpha= 1.044, beta=1
Fold 4	328	407	alpha= 0.8059, beta=1
Fold 5	312	423	alpha= 0.73759, beta=1

Kết quả:

Accuracy	Precision	Recall	F1_Score	Execution time(s)
83.9790	82.8822	81.5913	82.0489	7.2104

Nhận xét:

- So với mô hình Logistic Regression sử dụng Binary Cross Entropy phía trên, các than điểm nhìn chung có tăng nhưng không thật sự nhiều, riêng theo than đo recall thì giảm. Lí giải cho điều này là do bộ dữ liệu ban đầu có sự phân bổ lệch về phía class 1 nhiều, dẫn đến khi chia fold có 4 trên 5 fold có tập train phân bổ bị lệch về class 1. Khi huấn luyện mô hình Logistic Regression đơn thuần với mức độ ảnh hưởng của 2 class là như nhau, mô hình sẽ thiên về hướng dự đoán lớp có số lượng nhiều hơn, thế nên 4 trên 5 mô hình khi thực hiện kfold sẽ dự đoán cho nhãn 1 tốt hơn, dẫn đến Recall có phần cao hơn so với khi ta đánh trọng số để điều chỉnh lại độ cân bằng của 2 nhãn.
- Xét về thời gian thực thi, ở trường hợp thực nghiệm này thời gian thực thi vẫn lâu và có phần lâu hơn so với thực nghiệm trên.

c) Logistic Regression của thư viện Sklearn được tuning parameters bằng GridSearchCV

Logistic Regression có một số siêu tham số và việc tìm kiếm siêu tham số tối ưu là một nhiệm vụ rất khó giải quyết. Nhưng nó có thể được tìm thấy bằng cách thử tất cả các kết hợp và xem thông số nào hoạt động tốt nhất. Ý tưởng chính đằng sau nó là tạo một lưới các siêu tham số và chỉ cần thử tất cả các kết hợp của chúng .Do đó, phương pháp này được gọi là Gridsearch, và thư viện Sklearn có một công cụ là

GridSearchCV hỗ trợ cho chúng ta có thể tìm được bô siêu tham số tốt nhất cho mô hình theo ý tưởng như trên.

GridSearchCV lấy một từ điển mô tả các tham số có thể được thử trên một mô hình để huấn luyện nó. Lưới tham số được định nghĩa như một từ điển, trong đó các khóa là các tham số và các giá trị là cài đặt cần kiểm tra.

Thực hiện Tuning siêu tham số theo các giác trị của mỗi tham số như sau:

+Tham số penalty chuyền vào cho mô hình sẽ được để mặc định là "elasticnet" mà không cần thay đổi. Do Elastic Net có 1 hệ số để điều chỉnh Norm 1, như đã nói ở trên, nếu hệ số này =1 thì hàm sẽ là norm chuẩn bậc 1 và =0 thì hàm sẽ về norm chuẩn bậc 2.

+l1_ratio chính là hệ số nói trên, ta sẽ cho hệ số này chạy trong khoảng từ 0 đến 1, bước chạy là 0.05

+C chạy từ 0 đến 5, bước chạy 0.1

GridSearchCV sẽ thực hiện huấn luyện và đánh giá tổng cộng 1020 mô hình với tham số truyền vào khác nhau để chọn ra được đâu là tham số tốt nhất cho mô hình.

Kết quả thu về mô hình sẽ đạt kết quả đánh giá cao nhất với bộ tham số:

C= 0.2,
l1_ratio= 0.7368421052631579,
penalty='elasticnet'

Đánh giá lại với Kfold Cross Validation thu được kết quả:

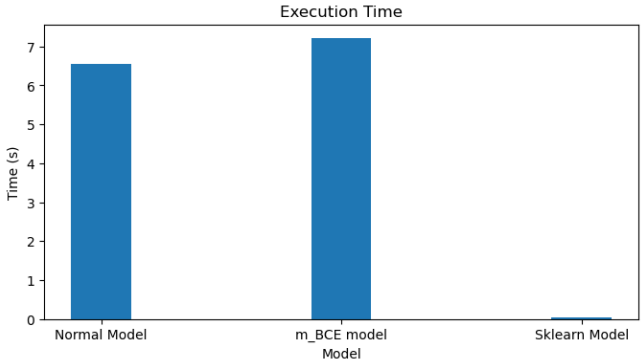
Accuracy	Precision	Recall	F1_Score	Execution time(s)
84.4126	83.1463	82.842	82.5836	0.0349

Nhận xét:

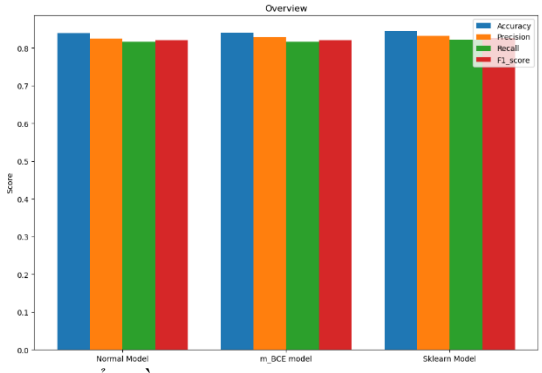
- Tuy số lượng bộ tham số phải thử nghiệm lớn (1020 bộ tham số) nhưng thời gian chạy thuật toán vẫn tương đối nhanh chỉ mất khoảng 19.1204s.
- Mô hình thu về khi thực hiện tuning bằng GridSearchCV có kết quả đánh giá tốt nhất trên cả 4 than đo, nhưng độ chênh lệch cũng không thật sự đáng kể.
- Xét về thời gian thực thi, Logistic Regression của thư viện Sklearn có thời gian thực thi nhanh hơn rất nhiều so với mô hình Logistic Regression được em xây dựng thủ công. Đó cũng là lí giải tại sao GridSearchCV phải tuning không gian nghiệm lớn hơn rất nhiều nhưng thời gian thực hiện vẫn nhanh hơn rất nhiều.

2. Tổng kết

Cả 3 mô hình đều cho kết quả thực nghiệm tốt, nhưng xét riêng về thời gian thực thi thì chỉ có mô hình của thư viện Sklearn là có thời gian thực thi tốt. Ưu điểm của thuật toán là dễ thực hiện, dễ cài đặt và cũng hoạt động hiệu quả đối với tập dữ liệu có phân tách tuyến tính như bộ dữ liệu Heart Disease. Tuy nhiên, Logistic Regression lai nhạy cảm với những điểm dữ liệu outlier và dữ liệu không phân tách tuyến tính, rõ có thể nói bài toán tối ưu hồi quy logistic cũng chính là bài toán tối ưu dữ liệu. Để mô hình có được điểm đánh giá cao hơn, ta có thể thực hiện thêm 1 vài bước tối ưu hóa dữ liệu như xóa bỏ đi outlier, tăng cường dữ liệu, …



Hình 40. Biểu đồ so sánh về thời gian huấn luyện của các mô hình Logistic Regression



Hình 41. Biểu đồ so sánh các mô hình Logistic Regression trên các than đo

Bài toán ban đầu của ta là dự đoán suy tim bằng mô hình máy học nên giữa 4 than đo, bên cạnh F1 score ta sẽ ưu tiên chọn mô hình dựa theo than đo Recall. Mô hình đề xuất cuối cùng của em chính là Logistic Regression sử dụng thư viện Sklearn với tham số đầu vào là:

C= 0.2, l1_ratio= 0.7368421052631579,
penalty='elasticnet'.

B. K-nearest neigbors:

1. Dataset

Bộ dataset nhóm đã chọn là dự đoán bệnh suy tim, với 11 thuộc tính đầu vào, 1 nhãn gồm 2 giá trị Yes hoặc No và 918 dòng dữ liệu. Với bộ dữ liệu này thì thuật toán KNN gần như không bị ảnh hưởng nhiều về tốc độ cũng như bộ nhớ khi dự đoán giá trị mới vì số dữ liệu là khá ít (tuy nhiên vẫn sẽ chậm hơn các thuật toán khác). Đồng thời, như trong phân tích EDA, dataset này cân bằng về số nhãn của 2 lớp, vì vậy KNN cũng chạy khá tốt trong trường hợp này. Và qua boxplot của các numerical feature, thuộc tính có nhiều outlier "RestingBP" cũng được lược bỏ đi sau khi cân nhắc nhiều yếu tố. nên dataset cũng không còn nhiều outlier.

Như đã nêu trong phần nhược điểm, để mô hình KNN đạt kết quả tốt thì dữ liệu cần phải được chuẩn hóa, đối với bộ dataset này thì sau khi thử nghiệm 2 hàm chuẩn hóa Standard Scaler và Min Max Scaler thì ta nhận thấy hàm Standard Scaler cho ra kết quả tốt hơn.

Bộ dữ liệu dùng cho mô hình là bộ dữ liệu đã được loại bỏ đi 2 thuộc tính "RestingBP" và "RestingECG", do sau bước EDA, nhóm nhận thấy 2 thuộc tính này không ảnh

hưởng nhiều đến kết quả đầu ra và còn gây nhiễu. Nếu dùng thư viện sklearn.feature_selection cũng sẽ nhận thấy được rằng 2 thuộc tính này ít liên hệ đến kết quả cuối cùng (mô tả trong phần code).

2. Kết quả thực nghiệm

Do mô hình KNN có nhiều tham số có thể thay đổi làm ảnh hưởng đến accuracy nên ở phần mô tả thực nghiệm, ta sẽ so sánh dựa trên các tham số có ảnh hưởng lớn đến độ chính xác mô hình là k, p (công thức tính khoảng cách theo chuẩn 1 hoặc 2 theo độ đo minkowski) và weights (cách tính trọng số cho các điểm gần nhất).

p	weights	k	Accuracy (%)	F1-score (%)
1	uniform	5	84.62	83.09
		15	**86.05**	**84.45**
		25	85.94	84.31
		35	85.72	84.08
	distance	5	84.52	82.99
		15	86.59	85.04
		25	**86.59**	**85.07**
		35	86.16	84.54
2	uniform	5	84.84	83.24
		15	**85.50**	**83.82**
		25	84.96	83.27
		35	84.42	82.66
	distance	5	84.74	83.04
		15	85.39	83.72
		25	**85.50**	**83.85**
		35	84.96	83.23

Nhận xét:

- Kết quả cao nhất với bộ tham số {p=1, weights='distance', k=25}.
- Ở cả p bằng 1 và 2, với weights='uniform' thì k=15 cao nhất, còn 'distance' thì k=25 cao nhất.

Sau khi chạy với nhiều bộ tham số, ta dùng đến hàm GridSeachCV để tìm ra được bộ tham số tối ưu cho mô hình, kết quả cho mô hình tốt nhất theo các tham số: ***{p=1, k=24, weights='distance', algorithm='ball_tree'}***. Kết quả tốt nhất đạt được là: accuracy=86.48, f1-score=84.92.

→ Như vậy kết quả chạy thực nghiệm của ta so với kết quả hàm GridSeachCV đưa ra là gần như tương đồng, cho thấy đánh giá của ta là chính xác.

Ngoài ra, nếu áp dụng phương pháp PCA (giảm chiều dữ liệu), thay vì bỏ 2 thuộc tính đã nêu trên thủ công thì PCA sẽ giảm đi các thuộc tính không có sự đa dạng về giá trị, với các giá trị p=5, 7, 9 đều cho ra kết quả tệ hơn so với bỏ thủ công.

C. Decision Tree:

Đánh giá mô hình Decision Tree dựa trên protocol của nhóm

Với các cài đặt splitter="best",criterion = "gini", random_state=0

(*) giới hạn số lượng mẫu chia (min_samples_split)

(*)	Không có			
Độ sâu tối đa mỗi cây	**Accuracy (%)**	**F1_score (%)**	**Precision (%)**	**Recall (%)**
2	77.77	75.49	76.58	75.47
4	**82.34**	**80.34**	**80.66**	**80.45**
6	79.18	77.17	77.06	77.69
8	79.07	77.05	76.85	77.46
10	76.57	74.22	74.00	74.84

Bảng C.1 Bảng kết quả của mô hình Cây quyết định với không giới hạn mẫu chia

(*)	100			
Độ sâu tối đa mỗi cây	**Accuracy (%)**	**F1_score (%)**	**Precision (%)**	**Recall (%)**
2	77.77	75.49	76.58	75.47
4	79.62	77.25	77.85	77.11
6	79.51	77.10	77.74	76.92
8	79.51	77.10	77.74	76.92
10	79.51	77.10	77.74	76.92

Bảng C.2 Bảng kết quả của mô hình Cây quyết định với giới hạn 100 mẫu chia

(*)	200			
Độ sâu tối đa mỗi cây	**Accuracy (%)**	**F1_score (%)**	**Precision (%)**	**Recall (%)**
2	77.77	75.49	76.58	75.47
4	77.11	74.76	75.57	74.84
6	77.01	74.63	75.40	74.65
8	77.01	74.63	75.40	74.65
10	77.01	74.63	75.40	74.65

Bảng C.3 Bảng kết quả của mô hình Cây quyết định với giới hạn 200 mẫu chia

Nhận xét:

Sau quá trình chạy thử khi đổi 2 tham số là độ sâu của cây (max_depth) và (min_samples_split), chúng ta đưa ra được một số nhận xét sau đây về mô hình khi dự đoán kết quả trong bộ dữ liệu đang xét:

- Với độ sâu của cây = 4 và không giới hạn số lượng mẫu chia, mô hình cho ra kết quả tốt nhất.
- Với không giới hạn số lượng mẫu chia:
  + Việc tăng độ sâu làm cho mô hình giảm dần độ chính xác. Qua đó thấy đc tình trạng mô hình đang bị overfitting với tập train nên làm giảm kết quả khi so với tập test.
- Tăng số lượng giới hạn mẫu lên 100:
  + Kết quả của độ sâu 4 bị giảm (giảm tầm 3%).
  + Kết quả của độ sâu 10 tăng (tăng tầm 3%).
  + Ở các độ sâu khác thì có sự thay đổi nhỏ giữ các điểm.
  + Kết quả độ sâu 2 vẫn giữ nguyên.

- Tăng số lượng giới hạn mẫu lên 200:

+ Kết quả ở các độ sâu giảm mạnh ngoại trừ mức 2 (giảm hơn 2%).

+ Xét trong trường hợp giới hạn trên, kết quả giảm dần theo độ sâu (tới mức 4) và sau đó không thay đổi.

Chốt lại, để mô hình Decision Tree cho ra kết quả tốt nhất cần điều chỉnh các tham số như độ sâu của cây, giới hạn số lượng mẫu chia, ... Việc thay đổi quá mức cần thiết có thể dẫn đến tình trạng overfitting làm giảm kết quả. Vì vậy ngoài tìm và thay đổi tham số, chúng ta có thể sử dụng các kĩ thuật tỉa cành (Purning) để tối ưu kết quả.

D. Random Forest:

Kết quả đánh giá mô hình Random Forest:

Số cây (n_estimator)	Độ sâu tối đa mỗi cây	Accuracy (%)	Precision (%)	Recall (%)	F1_score (%)	Time inference (ms)
100	**5**	85.06	84.15	82.88	83.39	0.06
	10	84.30	83.06	82.22	82.54	0.06
	20	83.32	82.58	81.24	81.7	0.07
1000	**5**	85.17	84.06	83.10	83.49	0.6
	10	84.19	83.30	82.03	82.51	0.72
	20	83.75	82.67	81.66	82.03	0.68
10000	**5**	**85.17**	**84.07**	**83.10**	**83.49**	**5.57**
	10	84.30	83.39	82.16	82.63	6.09
	20	84.30	83.38	82.17	82.63	0.68

Nhận xét:

- Số lượng cây càng nhiều thì mô hình random forest càng ổn định, độ chính xác tăng. Nhưng đánh đổi lại, cần tài nguyên tính toán và tốn nhiều thời gian training.
- Độ sâu tối đa của cây lớn quá dễ làm cho các Decision tree bị overfit, dẫn đến kết quả của random forest cũng tệ theo.

E. Naïve Bayes:

	accuracy	precision	recall	f1-score	Time inference(ms)
Gaussian Naive Bayes	84.63	84.23	80.81	81.85	0.00152
Multinomial Naive Bayes	80.16	77.84	78.02	77.88	0.00141
Bernoulli Naive Bayes	80.71	79.02	76.51	77.04	0.00135

Nhận xét:

- Trong dữ liệu có cả các cột chứa biến liên tục, có cột chứa biến mang giá trị 0-1, có cột chứa biến phân loại .Gaussian Naive Bayes đạt được kết quả tốt nhất, vì có nhiều cột chứa biến liên tục nhất.
- Độ chính xác của Naive bayes trên bộ data này thấp hơn so với các phương pháp khác như neurol network bởi vì các yếu tố gây ra bệnh tim thường liên quan chặt chẽ tới nhau, nhưng giả thiết của Naive Bayes lại cho rằng chúng độc lập. Ví dụ như chế độ ăn chay FastingBS có liên quan chặt chẽ tới Cholesterol,…

F. AdaBoost:

1. AdaBoost với DecisionTree

Cài đặt trên bộ dữ liệu Heart Failure, xây dựng mô hình AdaBoost sử dụng DecisionTree làm base model với cùng các tham số và tăng dần n_estimators (số lượng tối đa cho phép của Weak Classifiers)

	Accuracy	Precision	Recall	F1_Score	Time
1	0.93	0.92	0.94	0.93	0.02
5	0.87	0.86	0.86	0.86	0.01
10	0.90	0.89	0.89	0.89	0.03
50	0.88	0.87	0.87	0.87	0.15
100	0.89	0.88	0.88	0.88	0.47
200	0.87	0.86	0.86	0.86	0.53
300	0.85	0.84	0.84	0.84	0.76
500	0.82	0.80	0.81	0.81	1.29
1000	0.82	0.80	0.81	0.81	2.00
5000	0.79	0.77	0.78	0.78	8.12

Hình 43. Bảng kết quả của AdaBoost sử dụng DecisionTree

Nhận xét: Khi tăng dần số lượng của các Weak Classifiers, hiệu suất của mô hình ở bộ dữ liệu này chưa chắc đã tốt hơn. Việc tăng các Weak Classifiers dẫn đến mô hình học chậm hơn, đồng thời cũng tốn tài nguyên hơn rất nhiều.

2. *AdaBoost với SVM*

	Accuracy	Precision	Recall	F1_Score	Time
1	0.82	0.82	0.79	0.80	0.07
5	0.82	0.81	0.79	0.79	0.44
10	0.82	0.81	0.79	0.79	0.89
50	0.82	0.81	0.79	0.79	4.99
100	0.82	0.81	0.79	0.79	12.81
200	0.38	0.19	0.50	0.27	21.67
300	0.38	0.19	0.50	0.27	21.58
500	0.38	0.19	0.50	0.27	35.58
1000	0.38	0.19	0.50	0.27	63.76
5000	0.38	0.19	0.50	0.27	787.22

Hình x.x Bảng kết quả của AdaBoost sử dụng SVM

Nhận xét:

- Có thể thấy rõ ràng SVM không phải là một bộ dự đoán cơ sở tốt cho AdaBoost. SVM chậm hơn, không ổn định và cũng không mang lại hiệu quả cao bằng DecisionTree

G. Neural Network:

Kết quả thực nghiệm

Số hidden layer	Số node mỗi hidden layer	Hàm kích hoạt của hidden layer	ACC	Precision	Recall	F1_score
1	(6)	Relu	85.06	83.75	82.99	83.33
2	**(6,6)**	**Relu**	**86.70**	**84.99**	**85.51**	**85.19**
3	(6,6)	Sigmoid	84.30	83.14	82.03	82.45
4	(6,10)	Relu	84.08	83.47	81.68	82.28
5	(6,10)	Sigmoid	84.30	83.14	82.03	82.45
6	(6,15)	Relu	85.17	84.31	82.78	83.41
7	(6,6,10)	Relu	83.43	81.75	81.37	81.43
8	(6,10,10)	Relu	85.50	83.74	83.95	83.82
9	(6,10,10,10)	Relu	86.37	84.94	84.84	84.82

Nhận xét :

- Ta thấy với bộ dữ liệu heart.csv, mô hình có độ chính xác khi có 2 tầng ẩn với số node lần lượt là 6, 6.
- Ta thấy mô hình có xu hướng tăng độ chính xác khi tăng số node hoặc số tầng ẩn.

H. SVM

- Kết quả thực nghiệm:

	Accuracy	Precision	Recall	F1 Score
Linear SVM	84.19	82.06	82.19	82.50
Polynomial Kernel SVM	73.63	74.18	73.69	71.62

Nhận xét: Ta có thể đoán ra được là với bộ dataset này thì dữ liệu đã có sự phân biệt tuyến tính nên đối với thuật toán Linear SVM nó sẽ tối ưu hơn so với Polynomial Kernel SVM.

VI. ĐÁNH GIÁ TỔNG QUAN

Sau quá trình thực nghiệm huấn luyện nhiều trường hợp của mô hình để tìm ra bộ tham số tốt nhất cho mô hình ở mỗi phương pháp, chúng tôi đã tổng hợp được bản kết quả thực nghiệm với mô hình tốt nhất ở mỗi phương pháp theo protocol chung như sau:

Method	*Accuracy*	*Precision*	*Recall*	*F1(macro)*
Logistic Regression	84.4126	83.1463	82.842	82.5836
KNN	86.59	85.21	84.94	85.07
Decision Tree	82.34	80.66	80.45	80.34
Random Forest	85.17	84.07	83.10	83.49
Gaussian Naive Bayes	84.63	84.23	80.81	81.85
AdaBoost	**93**	**92**	**94**	**93**
Neural Network	86.70	84.99	85.51	85.19
SVM	84.19	82.06	82.19	82.50

Qua đó, chúng tôi nhận thấy rằng mô hình AdaBoost đạt hiệu quả khá cao trong phạm vi đề tài này theo thang đo F1 và recall. Lí do chính là vì cơ sở toán học vững chắc của nhóm thuật toán Boosting nói chung và AdaBoost nói riêng. Từ việc sửa chữa lỗi lầm của bộ phân loại phía trước đến việc kết hợp các Weak Classifiers để tạo ra một model hoàn chỉnh. Ngoài ra, điểm yếu của AdaBoost chính là một bộ dữ liệu có nhiều ngoại lai, tuy nhiên ở bộ dữ liệu Heart Failure sau khi tiền xử lí lại khá sạch, ít nhiễu đã khắc phục nhược điểm này của AdaBoost, góp phần không nhỏ cho hiệu suất của mô hình này.

Ngoài ra, chúng tôi thấy rằng mô hình Decision Tree cho ra kết quả thấp nhất trong các mô hình. Nguyên nhân chủ yếu là vì mô hình Decision Tree là một mô hình đơn giản, tuy đã bộ dữ liệu Heart Failure đã được xử lí và ít trường hợp nhiễu nhưng mô hình Decision Tree vẫn rất phụ thuộc vào các điều kiện giới hạn tạo cây cũng như sử dụng các phương pháp tỉa cành để có thể cho ra kết quả tốt hơn.

VII. TỔNG KẾT

Dựa vào các kết quả chuẩn đoán y khoa của bệnh nhân, và bằng cách kết hợp sử dụng các phương pháp máy học tiêu biểu hiện hiện nay, chúng tôi đã xây dựng thành công những hệ thống giúp hỗ trợ dự đoán suy tim ở bệnh nhân, điều này cũng chứng tỏ được sức mạnh vực trội của những phương pháp máy học phía trên trong cuộc sống thực tiễn. Tuy những thực nghiệm trên chỉ dừng lại ở mức ở đề tài môn học, nhưng

trong tương lai chúng tôi sẽ tìm cách cải tiến chúng, tìm hiểu thêm những mô hình mới, những cách cải tiến mới, cũng như là tìm thêm những nguồn dữ liệu uy tín khác, kết hợp thêm nhiều bước xử lý dữ liệu, để từ đó có thể hiện thực được bài toán của nhóm vào thực tiễn, áp dụng thực tế vào cuộc sống của con người.

Cảm ơn sự đóng góp của các thành viên trong nhóm trong viện hỗ trợ tìm hiểu, thực nghiệm với bộ dữ liệu trên nhiều phương pháp khác nhau, và những đóng góp chung trong đề tài chung của nhóm.

- **Nguyễn Hoàng Tuấn** (Nhóm trưởng): mô hình Logistic Regression
- **Lê Viết Lâm Quang**: mô hình K-nearest neigbors
- **Trần Đăng Khoa**: mô hình Decision Tree
- **Nguyễn Minh Lý**: mô hình Random Forest
- **Nguyễn Đặng Quang Tuấn**: mô hình Naïve Bayes
- **Mai Duy Ngọc**: mô hình AdaBoost
- **Đào Danh Đăng Phụng**: Mô hình SVM
- **Phan Tấn Thương**: Mô hình Neural Network

Link tổng hợp source code:
https://drive.google.com/drive/folders/1Eh4YALCvTaZHELJkMbqO4cJ9SPXR7z9V?usp=share_link

VIII. TRÍCH DẪN

[1] Heart Failure Prediction Dataset | Kaggle
[2] Machine Learning cơ bản (machinelearningcoban.com)
[3] Types of Regularization in Machine Learning | by Aqeel Anwar | Towards Data Science
[4] Implement Logistic Regression with L2 Regularization from scratch in Python | by Tulrose Deori | Towards Data Science
[5] What is Logistic regression? | IBM
[6] sklearn.linear_model.LogisticRegression — scikit-learn 1.2.0 documentation
[7] Machine Learning cơ bản (machinelearningcoban.com)
[8] https://viblo.asia/p/support-vector-machine-trong-hoc-may-mot-cai-nhin-don-gian-hon-XQZkxoQmewA#_uu-diem-cua-svm-la-gi-4
[9] https://trituenhantao.io/kien-thuc/svm-qua-kho-hieu-hay-doc-bai-nay/
[10]
[11] https://machinelearningcoban.com/2017/04/09/smv/
[12] https://machinelearningcoban.com/2017/04/22/kernelsmv/
[13] https://www.geeksforgeeks.org/naive-bayes-classifiers/
[14] https://machinelearningcoban.com/2017/08/08/nbc/
[15] https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
[16] https://viblo.asia/p/thuat-toan-phan-lop-naive-bayes-924lJWPm5PM
[17] https://scikit-learn.org/stable/modules/tree.html
[18] https://www.geeksforgeeks.org/decision-tree/
[19] https://machinelearningcoban.com/tabml_book/ch_model/decision_tree.html
[20] https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/#:~:text=CART%20Algorithm&text=It%20is%20a%20decision%20tree,threshold%20value%20of%20an%20attribute.
[21] https://miai.vn/2021/09/12/tim-hieu-va-code-cay-quyet-dinh-decision-tree-mi-ai/
[27] https://www.geeksforgeeks.org/differences-between-random-forest-and-adaboost/
[28] https://https://phamdinhkhanh.github.io/deepai-book/ch_ml/Boosting.html#tai-lieu-tham-khao
[29] https://blog.paperspace.com/adaboost-optimizer/.
[30] https://machinelearningcoban.com/tabml_book/ch_model/random_forest.html
[31] https://medium.com/analytics-vidhya/bootstrapping-and-oob-samples-in-random-forests-6e083b6bc341
[32] https://roboticsbiz.com/pros-and-cons-of-random-forest-algorithm/
[33] https://phamdinhkhanh.github.io/deepai-book/ch_ml/RandomForest.html
[34] https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
[35] https://towardsdatascience.com/tree-algorithms-explained-ball-tree-algorithm-vs-kd-tree-vs-brute-force-9746debcd940
[36] https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/
[37] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html