

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐÀO TẠO CHẤT LƯỢNG CAO



ĐỒ ÁN 3

**XÂY DỰNG CÁC THUẬT TOÁN MACHINE
LEARNING TRÊN TẬP DỮ LIỆU TITANIC**

SVTH: NGUYỄN THỊ NGỌC TRÂN

MSSV: 16131104

Ngành : Công nghệ thông tin

GVHD: TS. HUỖNH XUÂN PHỤNG

TP. Hồ Chí Minh, tháng 07 năm 2020

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH

KHOA ĐÀO TẠO CHẤT LƯỢNG CAO



ĐỒ ÁN 3

**XÂY DỰNG CÁC THUẬT TOÁN MACHINE
LEARNING TRÊN TẬP DỮ LIỆU TITANIC**

SVTH: NGUYỄN THỊ NGỌC TRÂN

MSSV: 16131104

Ngành : Công nghệ thông tin

GVHD: TS. HUỲNH XUÂN PHỤNG

TP. Hồ Chí Minh, tháng 07 năm 2020

MỤC LỤC

Chương 1 : GIỚI THIỆU VỀ HỌC MÁY.....	1
1.1. Khái niệm về học máy	1
1.2. Các dạng thuật toán của học máy.....	1
1.2.1. Supervised learning	1
1.2.2. Unsupervised learning	2
1.3. Các bước học máy	3
Chương 2 : PHÂN TÍCH TẬP DỮ LIỆU	4
2.1. Feature	4
2.2. Label	5
Chương 3 : XỬ LÝ DỮ LIỆU	6
3.1. Import dữ liệu và tổng quan.....	6
3.2. Chuẩn bị dữ liệu	10
3.3. Chia tập dữ liệu train và test	11
Chương 4 : LOGISTIC REGRESSION	12
4.1. Giới thiệu	12
4.2. Hypothesis	12
4.3. Decision boundary.....	13
4.4. Cost function	15
4.5. Cost function & Gradient Descent	16
4.6. Multiclass Classification: One-vs-all	17

4.7.	Vấn đề khi gặp Overfitting	19
4.8.	Cost function – Regularization	21
4.9.	Code.....	23
4.9.1.	Thư viện.....	23
4.9.2.	Cài đặt thuật toán	23
Chương 5 : DECISION TREE.....		25
5.1.	Giới thiệu	25
5.2.	Ưu, nhược điểm.....	26
5.2.1.	Ưu điểm:	26
5.2.2.	Nhược điểm:	26
5.3.	Entropy	26
5.4.	Information Gain	27
5.5.	Code.....	27
5.5.1.	Thư viện.....	27
5.5.2.	Cài đặt thuật toán	27
Chương 6 : NEURAL NETWORK.....		32
6.1.	Giới thiệu	32
6.2.	Model representation	32
6.3.	Cost function	33
6.4.	Backpropagation algorithm:	34
6.5.	Random initialization:	35

6.6. Code.....	35
6.6.1. Thư viện.....	35
6.6.2. Cài đặt thuật toán	36
Chương 7 : SUPPORT VECTOR MACHINE (SVM)	38
7.1. Giới thiệu	38
7.2. Kernel SVM	38
7.3. Code.....	39
7.3.1. Thư viện.....	39
7.3.2. Cài đặt thuật toán	40
Chương 8 : KNN (K-nearest Neighbors)	42
8.1. Giới thiệu	42
8.2. Cách hoạt động.....	42
8.3. Code.....	44
8.3.1. Thư viện.....	44
8.3.2. Cài đặt thuật toán	44
Chương 9 : ADA BOOST	47
9.1. Giới thiệu	47
9.2. Cách thức hoạt động.....	47
9.3. Ưu, nhược điểm.....	48
9.4. Code.....	48
9.4.1. Thư viện.....	48

9.4.2. Cài đặt thuật toán	48
Chương 10 : NAIVE BAYES.....	50
10.1. Định nghĩa	50
10.2. Thuật toán	50
10.3. Các mô hình thuật toán Navie Bayes	51
10.4. Code	51
10.4.1. Thư viện.....	51
10.4.2. Cài đặt thuật toán	52
Chương 11 : K-MEAN	53
11.1. Giới thiệu.....	53
11.2. Cách hoạt động	53
11.3. Chọn giá trị k	54
11.4. Ứng dụng	55
11.5. Code	55
11.5.1. Thư viện.....	55
11.5.2. Cài đặt thuật toán	55
Chương 12 : RANDOM FOREST	57
12.1. Giới thiệu.....	57
12.2. Cách hoạt động	57
12.3. Code	58
12.3.1. Thư viện.....	58

12.3.2. Cài đặt thuật toán	58
Chương 13 : ĐÁNH GIÁ CÁC THUẬT TOÁN.....	60
TÀI LIỆU THAM KHẢO	61

DANH SÁCH CÁC HÌNH ẢNH, BIỂU ĐỒ

Hình 3.1	Biểu đồ tập titanic theo giới tính.....	8
Hình 3.2	Biểu đồ tập titanic theo Embarked.....	9
Hình 3.3	Biểu đồ tập titanic theo Pclass	9
Hình 4.1	Đồ thị biểu diễn Logistic Regression	12
Hình 4.2	Đồ thị linear decision boundaries	14
Hình 4.3	Đồ thị nonlinear decision boundaries.....	15
Hình 4.4	Bài toán One-vs-all.....	18
Hình 4.5	Các trường hợp LR.....	20
Hình 4.6	Kết quả độ chính xác của Logistic Regression.....	24
Hình 5.1	Biểu diễn Decision Tree	25
Hình 5.2	Mô hình biểu diễn tập dữ liệu Titanic.....	29
Hình 5.3	Cây nhị phân tập Titanic.....	30
Hình 5.4	Độ chính xác của thuật toán Decision Tree.....	31
Hình 6.1	Mô hình Neuron Network.....	32
Hình 6.2	Cài đặt thông số chạy Neuron Network	37
Hình 6.3	Độ chính xác của thuật toán Neuron Network	37
Hình 7.1	Biểu diễn thuật toán SVM	38
Hình 7.2	Cài đặt thông số train SVM	40

Hình 7.3 Độ chính xác của thuật toán SVM	41
Hình 8.1 Ví dụ thuật toán K-NN-1	43
Hình 8.2 Ví dụ thuật toán K-NN-2.....	43
Hình 8.3 Biểu đồ kiểm tra độ chính xác của k.....	45
Hình 8.4 Độ chính xác của thuật toán K-NN.....	46
Hình 9.1 Cách thức hoạt động của thuật toán Ada Boost.....	47
Hình 9.2 Độ chính xác của thuật toán Ada Boost	48
Hình 10.1 Độ chính xác của thuật toán Naive bayes.....	52
Hình 11.1 Chọn giá trị K theo phương pháp Elbow point.....	54
Hình 11.2 Độ chính xác của thuật toán K-mean	56
Hình 12.1 Cách thức hoạt động của thuật toán Random Forest	58
Hình 12.2 Độ chính xác của thuật toán Random Forest.....	59
Hình 13.1 Biểu đồ đánh giá mức độ chính xác của các thuật toán ML	60

Chương 1 : GIỚI THIỆU VỀ HỌC MÁY

1.1. Khái niệm về học máy

Học máy là một nhánh ứng dụng của trí tuệ nhân tạo (AI) cung cấp cho hệ thống khả năng tự động học hỏi và cải thiện từ kinh nghiệm mà không cần phải lập trình rõ ràng.

Chi tiết hơn, Tom Mitchell đã định nghĩa: “*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E* ”

Ví dụ: Nhận dạng chữ viết tay

E = Cơ sở dữ liệu chữ viết tay có trước và được phân loại

T = Nhận dạng và phân loại chữ viết tay trong ảnh

P = Tỷ lệ các chữ được nhận dạng chính xác

Các thuật toán của học máy thường được phân loại chính thành Supervised và Unsupervised.

1.2. Các dạng thuật toán của học máy

1.2.1. Supervised learning

Supervised learning là thuật toán dự đoán đầu ra (output) của dữ liệu mới (new input) dựa trên các cặp dữ liệu và đầu ra đã cho từ trước. Cặp dữ liệu này được gọi là data (hoặc feature), label. Supervised learning là thuật toán phổ biến nhất trong các thuật toán của học máy.

Thuật toán Supervised learning còn được phân loại thành 2 loại chính:

1.2.1.1 Classification

Classification được sử dụng khi bài toán của chúng ta có các labels của input data được chia thành các nhóm không liên tục với nhau hoặc là được chia thành một số hữu hạn. Ví dụ: Phân loại email là spam hay không spam. Dễ dàng thấy được 2 nhóm này hoàn toàn khác nhau và không liên tục.

1.2.1.2 Regression

Regression được sử dụng khi bài toán của chúng ta có các labels không được chia thành nhóm mà chia thành một giá trị số thực cụ thể. Ví dụ: Căn nhà rộng xm^2 , có y phòng và nằm gần trung tâm thành phố sẽ có giá là bao nhiêu? Có thể thấy giá căn nhà sẽ là một con số cụ thể.

1.2.2. Unsupervised learning

Với thuật toán Unsupervised learning, chúng ta sẽ không còn có sẵn các labels cho trước như supervised mà chỉ có dữ liệu đầu vào. Thuật toán sẽ dựa vào cấu trúc của dữ liệu để thực hiện công việc như phân nhóm (clustering) hoặc giảm chiều dữ liệu (dimensionality reduction). Thuật toán Unsupervised learning được phân thành các loại sau:

1.2.2.1 Clustering

Phân nhóm toàn bộ dữ liệu cho trước thành các nhóm nhỏ dựa trên sự tương quan giữa các dữ liệu trong nhóm. Ví dụ: Chúng ta đưa vào rất nhiều mảnh ghép có các hình thù tròn, tam giác, vuông, ngũ giác với màu sắc tương ứng là: xanh, đỏ, vàng, tím. Mặc dù chúng ta không đưa thêm bất kỳ gợi ý phân loại chúng như thế nào nhưng nhiều khả năng máy vẫn có thể phân loại các mảnh ghép đó theo màu hoặc hình dạng.

1.2.2.2 Dimensionality reduction

Dimensionality reduction là một cách để đơn giản hóa dữ liệu, giúp dữ liệu dễ trao đổi, tính toán nhanh hơn, và dễ lưu trữ hơn. Phương pháp đơn giản nhất trong thuật toán là phương pháp Principal Component Analysis (PCA).

1.2.2.3 Semi-supervised learning

Các bài toán khi chúng ta có một lượng lớn dữ liệu đầu vào x nhưng chỉ một phần trong chúng có labels y được gọi là Semi-supervised learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm trên. Ví dụ: Một số bài toán để có được các labels từ dữ liệu đầu vào cần có chuyên gia phân tích hay đầu tư rất nhiều thời gian và chi phí, chúng ta cần phải kết hợp cả supervised và unsupervised để tìm đầu ra.

1.2.2.4 Reinforcement learning

Reinforcement learning là thuật toán dạy cho máy tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất. Reinforcement learning chủ yếu được áp dụng vào trò chơi, các thuật toán cần xác định các nước đi tiếp theo để giành lấy điểm số cao nhất.

Ví dụ: AlphaGo Zero.

1.3. Các bước học máy

Một bài toán học máy cần trải qua 3 bước chính:

- Chọn mô hình: Chọn một mô hình thống kê cho tập dữ liệu.
- Tìm tham số: Các mô hình thống kê có các tham số tương ứng, nhiệm vụ lúc này là tìm các tham số này sao cho phù hợp với tập dữ liệu nhất có thể.
- Suy luận: Sau khi có được mô hình và tham số, ta có thể dựa vào chúng để đưa ra suy luận cho một đầu vào mới nào đó.

Chương 2 : PHÂN TÍCH TẬP DỮ LIỆU

Vào ngày 15 tháng 4 năm 1912, thảm họa Titanic là một thảm họa nổi tiếng nhất trong lịch sử, tàu Titanic bị chìm sau khi va chạm với tảng băng đã cướp đi 1502 sinh mạng trên tổng số 2224 hành khách và thủy thủ đoàn, số liệu thu thập được qua thảm họa được tổng hợp thành tập dataset titanic.

Dataset bao gồm các feature và label như sau:

2.1. Feature

- PassengerId: Unique Id of a passenger
- Pclass: Passenger Class (1 = 1st, 2 = 2nd, 3 = 3rd)
- Survival: Survival (0 = No, 1 = Yes)
- Name: Name
- Sex: Sex
- Age: Age
- Sibsp: Number of Siblings/ Spouses Abroad
- Parch: Number of Parents/Children Abroad
- Ticket: Ticket Number
- Fare: Passenger Fare (British pound)
- Cabin: Cabin
- Embarked: Port of Embarkation
(C= Chebourg, Q = Queenstown, S = Southampton)

2.2. *Label*

- Survival: Survival (0 = No, 1 = Yes)

Bài toán là cho một lượng dữ liệu training cho trước của 892 hành khách bao gồm các thông tin như tên, giới tính, tuổi, mã số vé, cabin, giá vé, quan hệ nhân thân, số lượng trẻ em, họ hàng, cha mẹ đi cùng, còn sống hay đã tử nạn.

Output: Dữ liệu đầu ra rất đơn giản đưa ra dự đoán về khả năng tử nạn của các hành khách còn lại dựa trên những yếu tố biết trước như tên, tuổi, giới tính, số người đi cùng, vị trí phòng,...

Ý nghĩa: Bài toán cho phép chúng ta có thể dựa vào đó để dự đoán khả năng sống sót của một người dựa trên một thảm họa, từ đó có thể rút ra những kinh nghiệm trong các chuyến hành trình trong tương lai, có thể là thiết kế lại tàu một cách an toàn hơn, dịch vụ chăm sóc, cảnh báo,...

Phân tích: Có thể dễ nhận thấy đây là một bài Two-Class Classification với kết quả đầu ra là NOT_SUVIVAL hoặc SUVIVAL.

Với một bài toán Two-Class Classification như thế này có thể giải bằng nhiều cách và nhiều thuật toán khác nhau như Decision Tree, Neuron network, KNN, SVM, Adaboost, Naive Bayes,...

Nếu bỏ qua các yếu tố may mắn thì có thể các dữ liệu sau có thể ảnh hưởng đến sự tồn vong của hành khách: giới tính, tuổi, vị trí boong tàu, loại vé, số người đi cùng. Những yếu tố có thể bỏ qua như: id của vé, tên.

Chương 3 : XỬ LÝ DỮ LIỆU

3.1. Import dữ liệu và tổng quan

Bước 1: Import các thư viện cần sử dụng

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Bước 2: Tiến hành đọc train_dataset và test_dataset.

```
df_train =
pd.read_csv('C:/Users/Administrator/Desktop/titanic/titanic/train.csv')
df_test =
pd.read_csv('C:/Users/Administrator/Desktop/titanic/titanic/test.csv')
# Explore the data by printing the head of the dataset.
df_train.head()

print('Column names and data of first twenty passengers - training
set:\n{}'.format(df_train.head(20)))
```

Bước 3: Tìm ra mối quan hệ giữa survived và các thuộc tính khác

```
#Use corr function to find out the liner
corr_matrix = df_train.corr()
print('Correlation between survived and other numerical
features:\n{}'.format(corr_matrix["Survived"].sort_values(ascending =
False)))
#visualize the correlation of Survived and other non numerical features.
df_train.info
```

Kết quả thu được:

Correlation between survived and other numerical features:

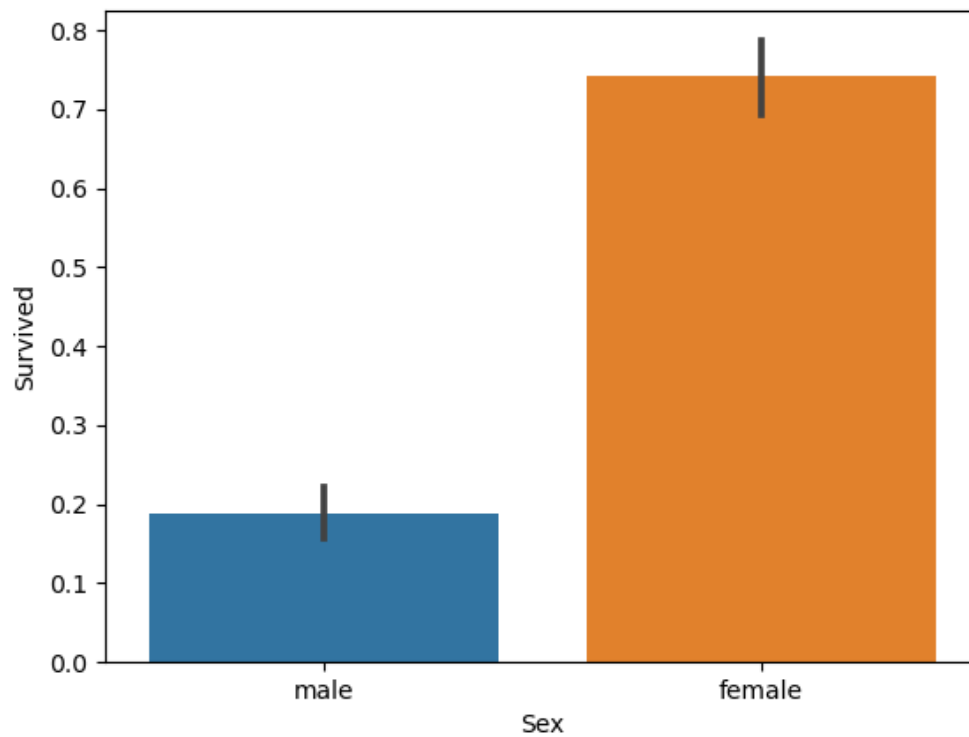
Survived	1.000000
Fare	0.257307
Parch	0.081629
PassengerId	-0.005007
SibSp	-0.035322
Age	-0.077221

```
Pclass          -0.338481  
Name: Survived, dtype: float64
```

Bước 4: Tiến hành vẽ một số biểu đồ dựa vào dữ liệu được đưa vào để thấy rõ sự tương quan giữa các thuộc tính

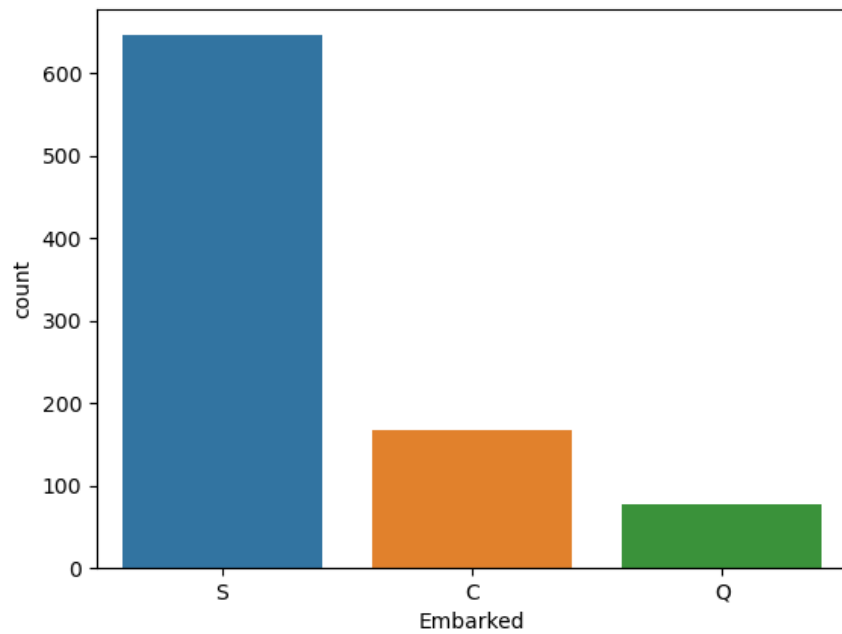
```
sns.barplot(  
    data = df_train,  
    x = 'Sex',  
    y = 'Survived'  
)  
plt.show()  
sns.countplot(df_train[ 'Embarked' ])  
plt.show()  
sns.barplot (  
    data = df_train,  
    x = 'Pclass',  
    y = 'Survived'  
)  
plt.show()
```


Các biểu đồ thu được:

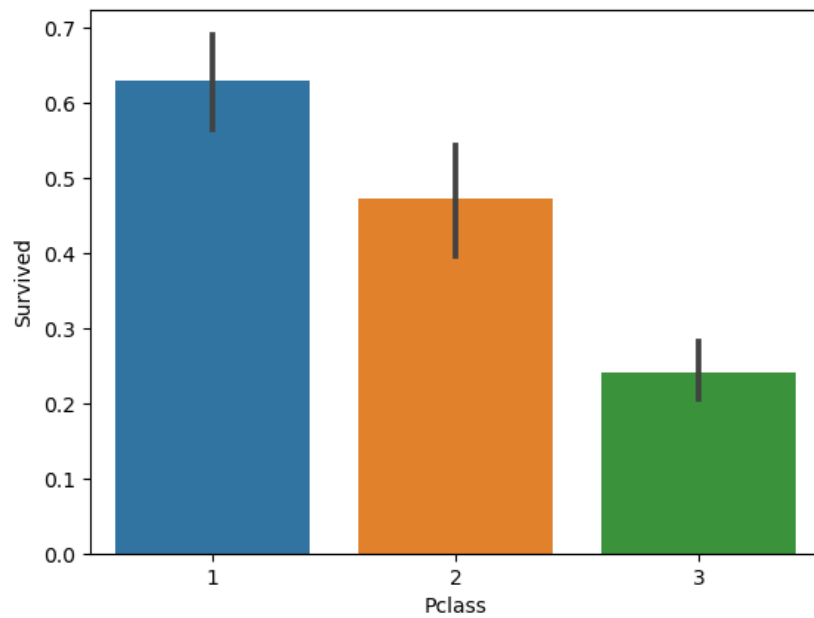


Hình 3.1 Biểu đồ tập titanic theo giới tính

Nhận xét: Ở biểu đồ này, có thể thấy rằng tỉ lệ sống của giới tính nữ cao hơn của nam.



Hình 3.2 Biểu đồ tập titanic theo Embarked



Hình 3.3 Biểu đồ tập titanic theo Pclass

Nhận xét: Hành khách ở các class trên có tỉ lệ sống cao hơn gần 65%. Những hành khách ở các class thấp hơn có tỉ lệ sống thấp hơn (27%).

3.2. Chuẩn bị dữ liệu

Bây giờ, chúng ta phải lựa chọn các features để xây dựng mô hình dự đoán. Có thể thấy rằng, các feature như Pclass, Sex, Fare và Embarked có sự ảnh hưởng cao.

Bước 1: Chuẩn bị cho dữ liệu đầu vào, loại bỏ một số thuộc tính không cần thiết trong tập train và tập test

```
#features are Pclass, Sex, Fare and Embarked
q = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp', 'Parch', 'Age']
df_train_set = df_train.drop(q, axis = 1)
df_train_set.head()
z = ['Name', 'Ticket', 'Cabin', 'SibSp', 'Parch', 'Age']
df_test_set = df_test.drop(z, axis = 1)
df_test_set.head()
```

Bước 2: Điền các giá trị null của feature Fare trong tập test bằng cách sử dụng function Mean() của python và tính giá trị trung bình

```
mean = df_test_set["Fare"].mean()
df_test_set["Fare"] = df_test_set["Fare"].fillna(mean)
df_test_set.info()
```

Bước 3: Chuyển đổi các thuộc tính string sang number

```
#Then translate non-numerical features to numerical features.
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()

df_train_set.iloc[:,2] =
labelencoder.fit_transform(df_train_set.iloc[:,2].values)
df_train_set.iloc[:,4] =
labelencoder.fit_transform(df_train_set.iloc[:,4].values)

df_test_set.iloc[:,2] =
labelencoder.fit_transform(df_test_set.iloc[:,2].values)
```

```
df_test_set.iloc[:,4] =
labelencoder.fit_transform(df_test_set.iloc[:,4].values)
```

```
df_train_set.info()
```

Kết quả thu được sau khi chuyển

```
>>> df_train_set.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 5 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Survived    891 non-null    int64
1   Pclass      891 non-null    int64
2   Sex         891 non-null    int32
3   Fare        891 non-null    float64
4   Embarked    891 non-null    int32
dtypes: float64(1), int32(2), int64(2)
memory usage: 28.0 KB
```

3.3. Chia tập dữ liệu train và test

Bước 1: Chia dữ liệu train thành 4 tập con: X_train, X_test, y_train, y_test

```
X = df_train_set.iloc[:, 1:5].values
Y = df_train_set.iloc[:, 0].values
```

Bước 2: Tiến hành lấy 40% dữ liệu để test, và 60% dữ liệu để train

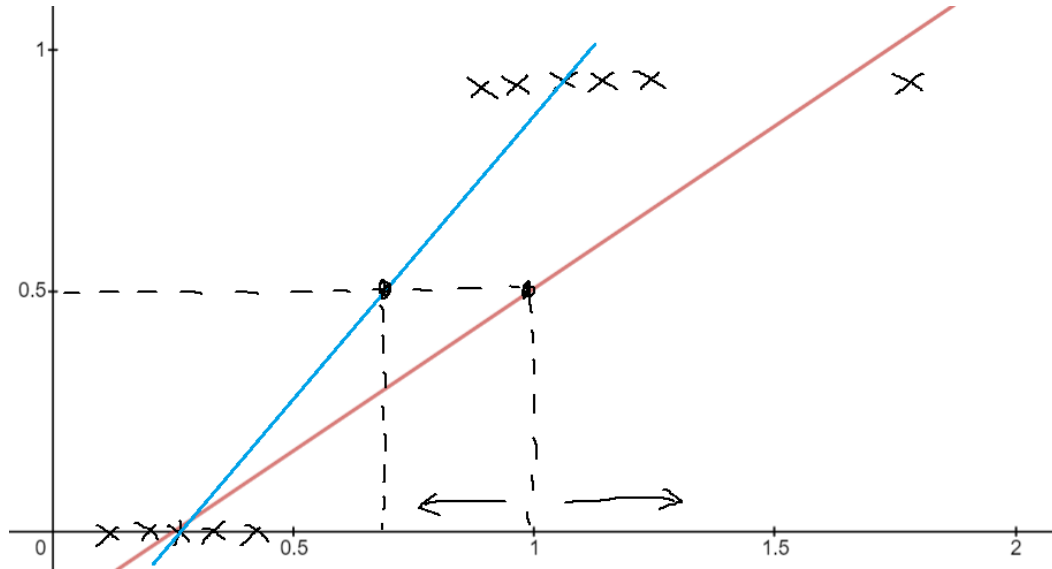
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (X,Y,test_size= 0.4,
random_state = 4)
```

Tổng quan về tập test và tập train:

```
>>> print (X_train.shape)
(534, 4)
>>> print (X_test.shape)
(357, 4)
>>> print (y_train.shape)
(534,)
>>> print (y_test.shape)
(357,)
```

Chương 4 : LOGISTIC REGRESSION

4.1. Giới thiệu



Hình 4.1 Đồ thị biểu diễn Logistic Regression

Thuật toán Logistic Regression được sử dụng nhiều để giải quyết các bài toán có Binary Classification. Ví dụ bài toán chuẩn đoán bệnh nhân ung thư:

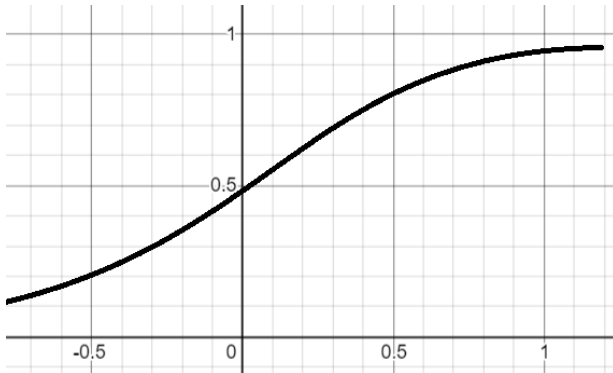
Ta có thể thấy như trên hình, nếu sử dụng Linear Regression thì ta sẽ có đường màu xanh. Chiều xuống trục x ta sẽ dễ dàng thấy được nếu data thuộc từ 0.7 trở đi sẽ bị bệnh ($y=1$), trở xuống sẽ không bị bệnh ($y=0$). Tuy nhiên, nếu data training của chúng ta có điểm cách xa nhau thì lúc đó Linear Regression sẽ cho ra đường màu đỏ. Lần này khi chúng ta chiếu xuống trục x sẽ thấy không còn chuẩn nữa, có một số data training bị chuẩn đoán sai. Lúc này đây, chúng ta nên sử dụng Logistic Regression để có được kết quả tốt nhất.

4.2. Hypothesis

Mục tiêu: $0 \leq h_{\theta}(x) \leq 1$

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\text{với } z = \theta^T x \rightarrow g(z) = \frac{1}{1 + e^{-z}} \text{ (sigmoid function)}$$



$h_{\theta}(x)$ tính toán xác suất khả năng $y = 1$ của dữ liệu input x

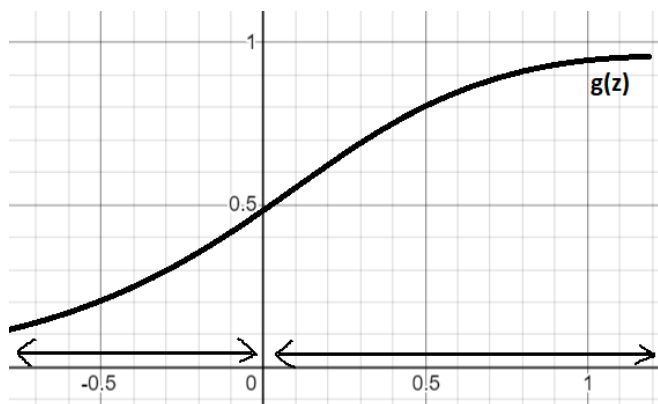
Ví dụ: $h_{\theta}(x) = 0.7 \rightarrow$ khả năng $y = 1$ là 70% và $y = 0$ sẽ là 30%

Từ đó ta có: $h_{\theta}(x) = P(y = 1|x, \theta)$

$\rightarrow P(y = 0|x, \theta) = 1 - h_{\theta}(x)$

4.3. Decision boundary

Ta có: $h_{\theta}(x) = g(\theta^T x)$



Predict:

$y = 1$ nếu $h_{\theta}(x) \geq 0.5 \rightarrow \theta^T x \geq 0$

$y = 0$ nếu $h_{\theta}(x) < 0.5 \rightarrow \theta^T x < 0$

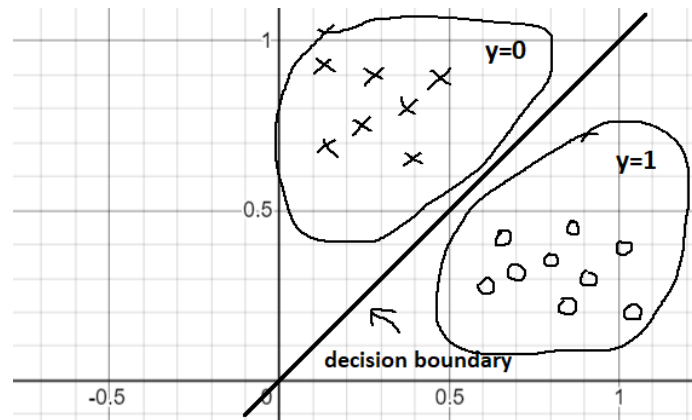
Cụ thể ví dụ:

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) \text{ với } (\theta = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix})$$

$$= g(0 + x_1 - x_2)$$

$$\rightarrow y = 1 \leftrightarrow x_1 - x_2 \geq 0 \leftrightarrow x_1 = x_2$$

Đồ thị:



Hình 4.2 Đồ thị linear decision boundaries

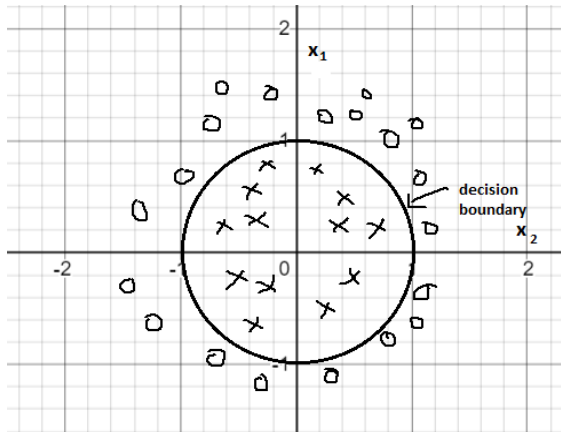
Ví dụ nonlinear decision boundaries:

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2) \text{ với } \theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$= g(-1 + x_1^2 + x_2^2)$$

$$y = 1 \leftrightarrow x_1^2 + x_2^2 \geq 1$$

4.4. Cost function



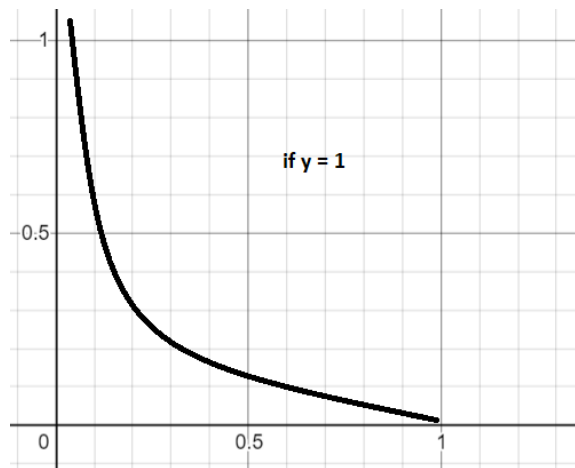
Hình 4.3 Đồ thị nonlinear decision boundaries

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

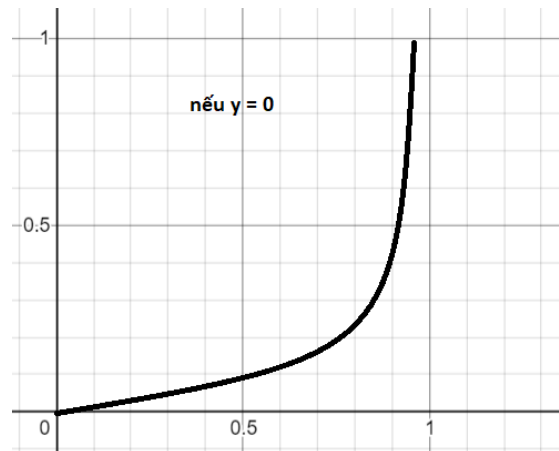
$$\text{cost}(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(h_{\theta}(x)) \text{ nếu } y = 1$$

$$\text{cost}(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(1 - h_{\theta}(x)) \text{ nếu } y = 0$$

Với $y = 1$, đồ thị liên hệ giữa $J(\theta)$ và $h_{\theta}(x)$ là:



Tương tự, với $y = 0$, đồ thị liên hệ giữa $J(\theta)$ và $h_\theta(x)$ là:



4.5. Cost function & Gradient Descent

Vì y luôn thường chỉ có 2 giá trị là 0 hoặc 1 nên chúng ta có thể ghép 2 trường hợp đó lại thành một trường hợp. Khi đó:

$$\text{cost}(h_\theta(x), y^{(i)}) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

Ta có cost function:

$$\begin{aligned}
J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{cost}(h_{\theta}(x^{(i)}), y^{(i)}) \\
&= -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))
\end{aligned}$$

Như linear regression, chúng ta cũng sử dụng Gradient Descent để tìm $\min_{\theta} J(\theta)$.

Repeat until convergence:

{

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \text{ với } for j = 0, \dots, n$$

$$= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \text{ với } h_{\theta}(x^{(i)}) = g(\theta^T x)$$

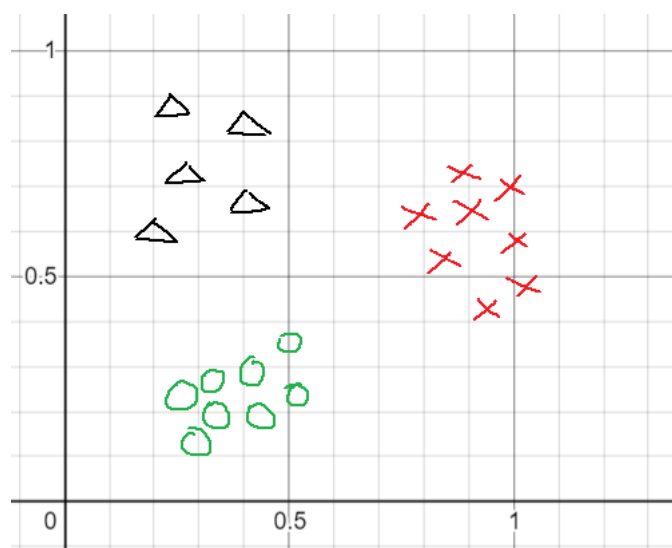
}

4.6. Multiclass Classification: One-vs-all

Chúng ta vẫn có thể sử dụng Logistic Regression cho bài toán có nhiều hơn 2 trường hợp. Thay vì $y = \{0, 1\}$, chúng ta sẽ mở rộng thành $y = \{0, 1, \dots, n\}$.

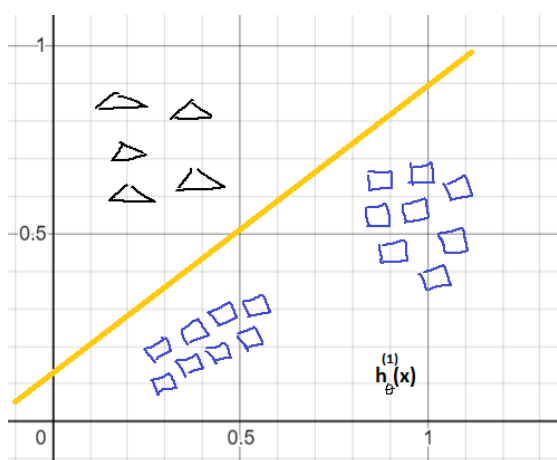
Để chi tiết hơn nghĩa là chúng ta sẽ chia bài toán thành $n + 1$ trường hợp (index bắt đầu đếm từ số 0) binary classification. Với mỗi class i , chúng ta sẽ dự đoán xác suất mà $y = i$.

Ví dụ:



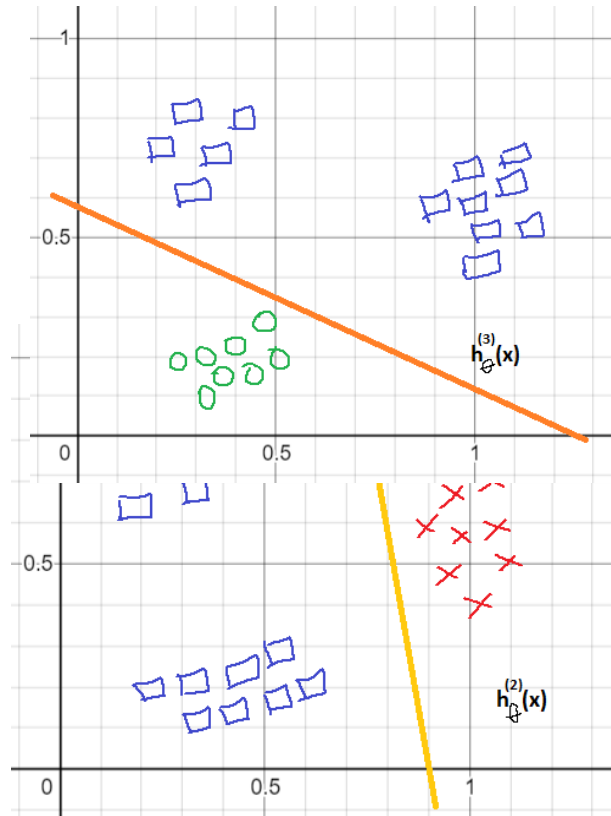
Hình 4.4 Bài toán One-vs-all

Với: $h_{\theta}^1(x) = P(y = 1|x, \theta)$



Với: $h_{\theta}^2(x) = P(y = 2|x, \theta)$

Với: $h_{\theta}^3(x) = P(y = 3|x, \theta)$

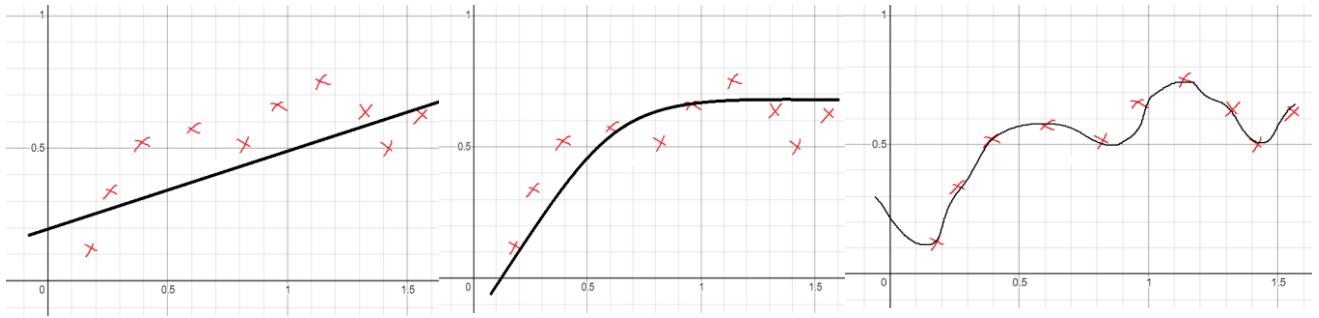


Sau khi training xong, với dữ liệu input x mới, để dự đoán, ta sẽ chọn class i có xác suất cao nhất.

$$y = \max_{(i)} h_{\theta}^i(x)$$

4.7. Vấn đề khi gặp Overfitting

Khi gặp một bài toán có data training không có xu hướng rõ ràng, chúng ta thường sẽ gặp các trường hợp dưới đây:



Hình 4.5 Các trường hợp LR

Với trường hợp 1: Sử dụng $y = \theta_0 + \theta_1 x$, có thể thấy được đường thẳng không fit được với các điểm data training tốt lắm \rightarrow cost function sẽ có giá trị cao và khi predict x new sẽ không cho kết quả tốt. Trường hợp này được gọi là underfitting hay high bias.

Với trường hợp 2: Sử dụng $y = \theta_0 + \theta_1 x + \theta_2 x^2$, có thể thấy đường curve fit hơn, đi qua được nhiều điểm dữ liệu hơn \rightarrow cost function có sai số nhỏ và cho ra kết quả predict x new tốt hơn.

Với trường hợp 3: Sử dụng $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$, có thể thấy đường cong fit rất hoàn hảo với các điểm data dữ liệu \rightarrow cost function gần như bằng 0. Tuy nhiên, với mọi dữ liệu x new thì sẽ không chắc rằng cho ra được kết quả predict tốt nhất. Trường hợp này được gọi là overfitting hay high variance.

Kết luận:

- Underfitting: Xảy ra khi hàm hypothesis của chúng ta quá đơn giản và cho ra cost function có giá trị cao.
- Overfitting: Xảy ra khi hàm hypothesis fit rất tốt với các data training (cost function gần như bằng 0 hay quá phức tạp, nhiều bậc nhưng lại thất bại khi predict một data dữ liệu mới.

Vậy khi gặp overfitting, chúng ta sẽ có 2 cách để giải quyết nó:

- Giảm số lượng các feature.
- Giảm bậc của θ_j (Regularization).

4.8. Cost function – Regularization

Xem lại ví dụ ở trên, ta có thể thấy được khi bậc đa thức bằng 2 ($y = \theta_0 + \theta_1 x + \theta_2 x^2$) là fit tốt nhất trong khi đó tăng lên bậc cao hơn thì $h_\theta(x)$ sẽ gặp vấn đề overfitting. Vậy vấn đề ta nên làm là loại bỏ đi các feature x^3, x^4, x^5 .

Trên thực tế khi làm việc với các vấn đề của Machine Learning ta phải sử dụng số lượng feature không chỉ một hay một vài, thậm chí là nhiều hơn hàng trăm feature. Khi đó việc lựa chọn ra feature nào bỏ đi là rất phức tạp. Thay vì bỏ trực tiếp, chúng ta sẽ thay đổi cost function để khi minimize sẽ thu được θ của x^3, x^4, x^5 xấp xỉ bằng 0 (penalty). Chẳng hạn:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 1000\theta_4^2 + 1000\theta_5^2$$

Khi ta thêm một lượng lớn $\theta_3, \theta_4, \theta_5$ sẽ dẫn đến việc muốn giảm giá trị của cost function đồng nghĩa với việc $\theta_3, \theta_4, \theta_5$ phải nhỏ, cụ thể là càng gần 0 càng tốt (do ta thêm vào θ^2 nên giá trị nhỏ nhất khi chúng = 0). Khi $\theta_3, \theta_4, \theta_5$ xấp xỉ 0 thì các feature x^3, x^4, x^5 càng ít ảnh hưởng tới $h_\theta(x)$. Phần được thêm được gọi là Regularization

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

Tham số λ được dùng để thay đổi mức ảnh hưởng của toàn bộ feature. Vì vậy, khi chọn λ cần lưu ý nếu λ quá nhỏ sẽ không giải quyết được overfitting và nếu chọn λ quá lớn sẽ dẫn tới tình trạng underfitting.

Chúng ta có thể apply Regularized cho cả Linear Regression và Logistic Regression:

Với Regularized Linear Regression:

Khi ứng dụng Gradient Descent, bước giảm trên từng θ sẽ như sau:

Repeat

{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \text{ với } j = 1, 2, 3, \dots, n$$

}

Nếu như ta nhóm θ_j lại với nhau sẽ có:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Normal Equation: non – interactive

$$\theta = (X^T X + \lambda \cdot L)^{-1} X^T y$$

$$\text{Với } L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ dimension } (n+1) \times (n+1)$$

Tương tự ta có Regularization cho Logistic Regression:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Gradient Descent:

Repeat

{

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \text{ với } j = 1, 2, 3, \dots, n$$

}

4.9. Code

4.9.1. Thư viện

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
```

4.9.2. Cài đặt thuật toán

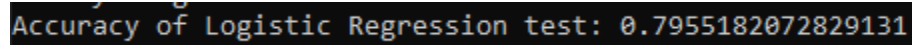
Bước 1:

```
# Instantiate our model
logreg = LogisticRegression()
# Fit our model to the training data
logreg.fit(X_train, y_train)
# Predict on the test data
logreg_predictions = logreg.predict(X_test)
```



```
print('Accuracy of Logistic Regression test:
{}'.format(metrics.accuracy_score(y_test, logreg_predictions)))
```

Độ chính xác của thuật toán Logistic Regression thu được là:



```
Accuracy of Logistic Regression test: 0.7955182072829131
```

Hình 4.6 Kết quả độ chính xác của Logistic Regression

Bước 2: Sử dụng mô hình này để tiến hành các dự đoán ngoài dữ liệu mẫu.

```
df_test_set.info()
test = df_test_set.iloc[:,1:5].values
# Instantiate our model
logreg = LogisticRegression()
logreg.fit (X,Y)
logreg_pred = logreg.predict(test)
```

Bước 3: Xuất tập dữ liệu mới có tên là LogisticRegression_SS_OH_FE2.csv

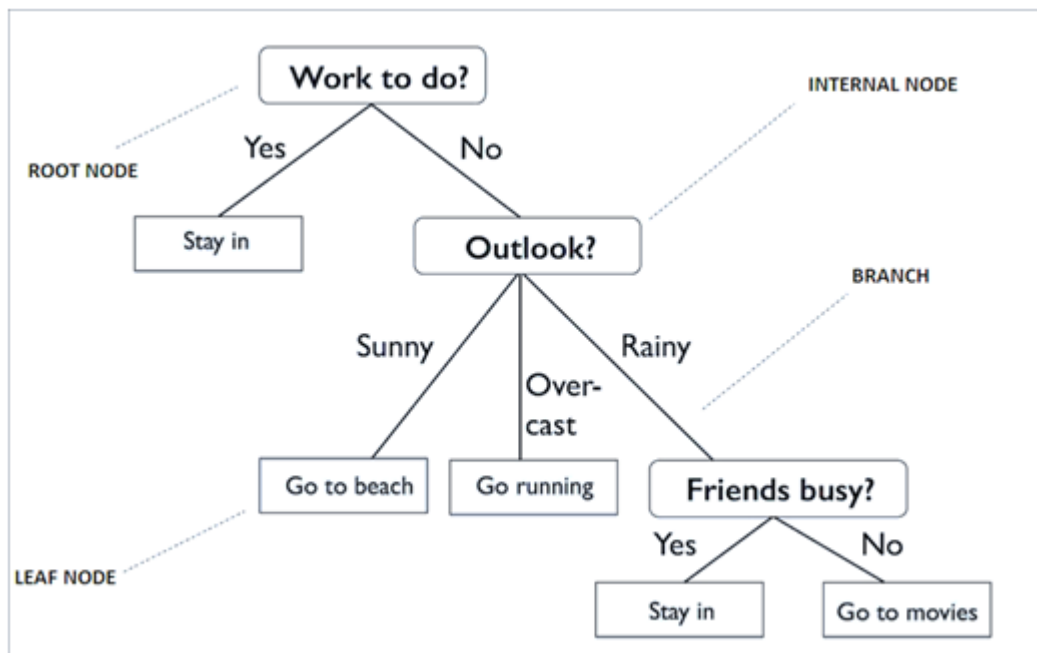
```
logreg_submission = pd.DataFrame({
    "PassengerId" : df_test_set["PassengerId"],
    "Survived" : logreg_pred
})
logreg_submission.to_csv('C:/Users/Administrator/Desktop/titanic/titanic/LogisticRegression_SS_OH_FE2.csv')
```

Chương 5 : DECISION TREE

5.1. Giới thiệu

Decision Tree là một thuật toán thuộc loại Supervised Learning, phương pháp học có giám sát, là một cây phân cấp có cấu trúc được dùng để phân lớp các đối tượng dựa trên các luật.

Decision Trees gồm 3 phần chính: 1 node gốc (root node), những node lá (leaf nodes) và các nhánh của nó (branches). Node gốc là điểm bắt đầu của cây quyết định và cả hai node gốc và node chứa câu hỏi hoặc tiêu chí để được trả lời. Nhánh biểu diễn các kết quả của kiểm tra trên nút. Ví dụ câu hỏi ở node đầu tiên yêu cầu câu trả lời là “yes” hoặc là “no” thì sẽ có 1 node con chịu trách nhiệm cho phản hồi là “yes”, 1 node là “no”.



Hình 5.1 Biểu diễn Decision Tree

- Root node: điểm ngọn chứa giá trị của biến đầu tiên được dùng để phân nhánh

- Internal node: các điểm bên trong thân cây là các biến chứa các thuộc tính, giá trị dữ liệu dùng để xét cho các phân nhánh tiếp theo
- Leaf node: là các lá cây chứa giá trị của biến phân loại sau cùng
- Branch: là các quy luật phân nhánh, nói đơn giản là mối quan hệ giữa các giá trị của biến độc lập (internal node) và các giá trị của biến mục tiêu (leaf node)

5.2. Ưu, nhược điểm

5.2.1. Ưu điểm:

Mô hình sinh ra các quy tắc dễ hiểu cho người đọc, tạo ra bộ luật với mỗi nhánh lá là một luật của cây.

Dữ liệu đầu vào có thể là dữ liệu missing, không cần chuẩn hóa hoặc tạo biến giả.

Có thể làm việc với cả dữ liệu số và dữ liệu phân loại.

Có thể xác thực mô hình bằng cách sử dụng các kiểm tra thống kê.

Có khả năng làm việc với dữ liệu lớn.

5.2.2. Nhược điểm:

Mô hình cây quyết định phụ thuộc rất lớn vào dữ liệu đầu vào. Thậm chí, với một sự thay đổi nhỏ trong bộ dữ liệu, cấu trúc mô hình cây quyết định có thể thay đổi hoàn toàn.

Cây quyết định hay gặp vấn đề overfitting.

5.3. Entropy

Entropy trong học máy và lý thuyết thông tin nói chung là thước đo tính ngẫu nhiên của thông tin đang được xử lý. Entropy càng cao, càng khó rút ra bất kỳ kết luận nào từ thông tin đó. Tung một đồng xu là một ví dụ về thông tin ngẫu nhiên, trong trường hợp này

Entropy đạt cực đại bằng 1, không có kết luận nào giúp dự đoán được kết quả tung đồng xu.

Công thức tính entropy:

$$H(\text{Goal}) = -B(p) / (p+n)$$

5.4. Information Gain

Information Gain dựa trên sự giảm của hàm [Entropy](#) khi tập dữ liệu được phân chia trên một thuộc tính. Để xây dựng một cây quyết định, ta phải tìm tất cả thuộc tính trả về Information gain cao nhất.

5.5. Code

5.5.1. Thư viện

```
using Accord;  
using Accord.IO;  
using Accord.MachineLearning.DecisionTrees;  
using Accord.MachineLearning.DecisionTrees.Learning;  
using Accord.Math;  
using Accord.Statistics.Analysis;  
using AForge;  
using Components;  
using System;  
using System.Data;  
using System.Drawing;  
using System.IO;  
using System.Windows.Forms;  
using ZedGraph;
```

5.5.2. Cài đặt thuật toán

Bước 1: Đọc dữ liệu đầu vào gồm 5 cột đầu tiên lượt là: Pclass, Sex, Parch, Fare, Age, Embarked

```
// Creates a matrix from the entire source data table
```

```

        double[, ] table = (dgvLearningSource.DataSource as
DataTable).ToMatrix(out columnNames);

        // Get only the input vector values (first two columns)
        double[][] inputs = table.GetColumns( 0, 1, 2, 3, 4,
5).ToJagged();

        // Get the expected output labels (last column)
        int[] outputs = table.GetColumn(6).ToInt32();

        // Specify the input variables
        DecisionVariable[] variables =
        {
            new DecisionVariable("Pclass",
DecisionVariableKind.Continuous),
            new DecisionVariable("Sex",
DecisionVariableKind.Continuous),
            new DecisionVariable("Parch",
DecisionVariableKind.Continuous),
            new DecisionVariable("Fare",
DecisionVariableKind.Continuous),
            new DecisionVariable("Age",
DecisionVariableKind.Continuous),
            new DecisionVariable("Embarked",
DecisionVariableKind.Continuous),
        };

```

Bước 2: Sử dụng thuật toán C45 để train mô hình

```

        // Create the C4.5 learning algorithm
        var c45 = new C45Learning(variables);
        // Learn the decision tree using C4.5
        tree = c45.Learn(inputs, outputs);

        // Show the learned tree in the view
        decisionTreeView1.TreeSource = tree;

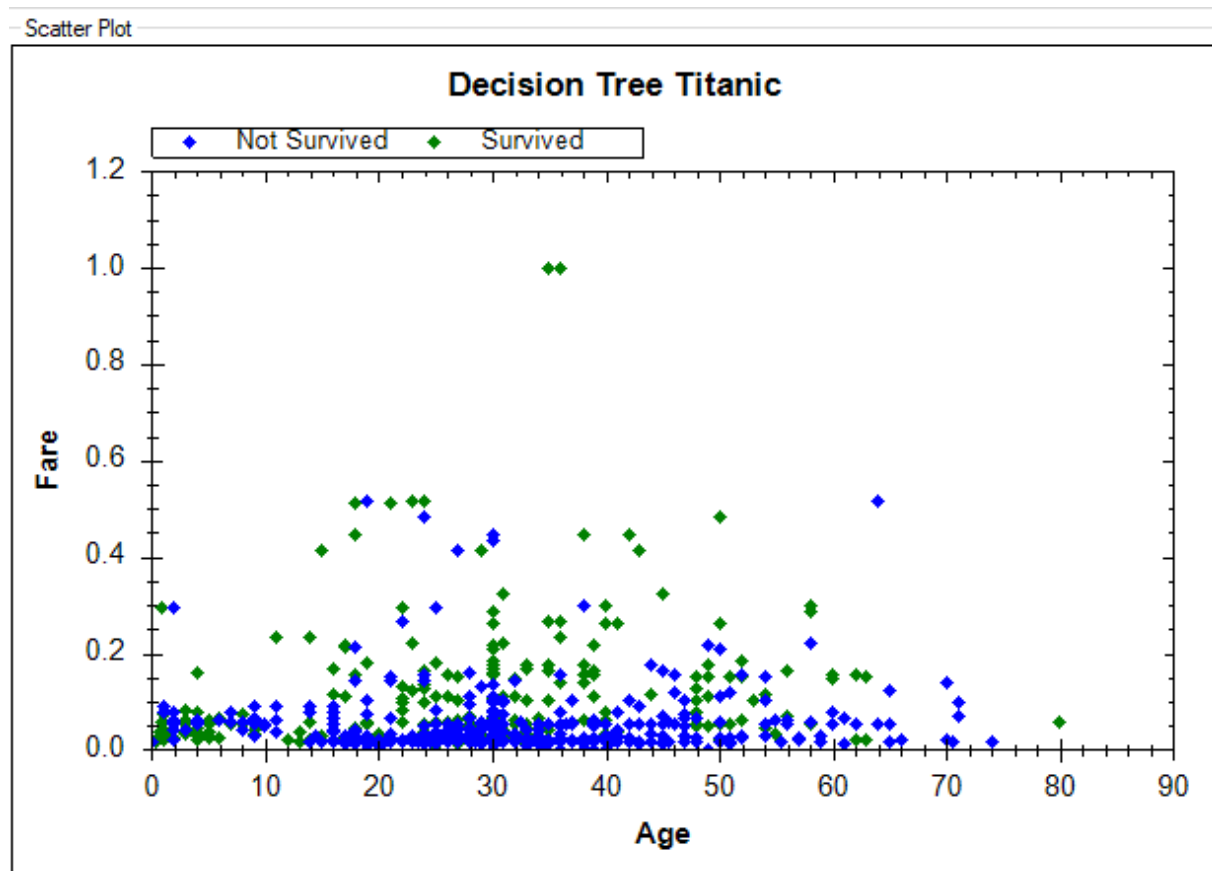
        // Get the ranges for each variable (X and Y)
        DoubleRange[] ranges = table.GetRange(0);

        CreateScatterplot(zedGraphControl2, table);

        lbStatus.Text = "Learning finished! Click the other tabs to
explore results!";

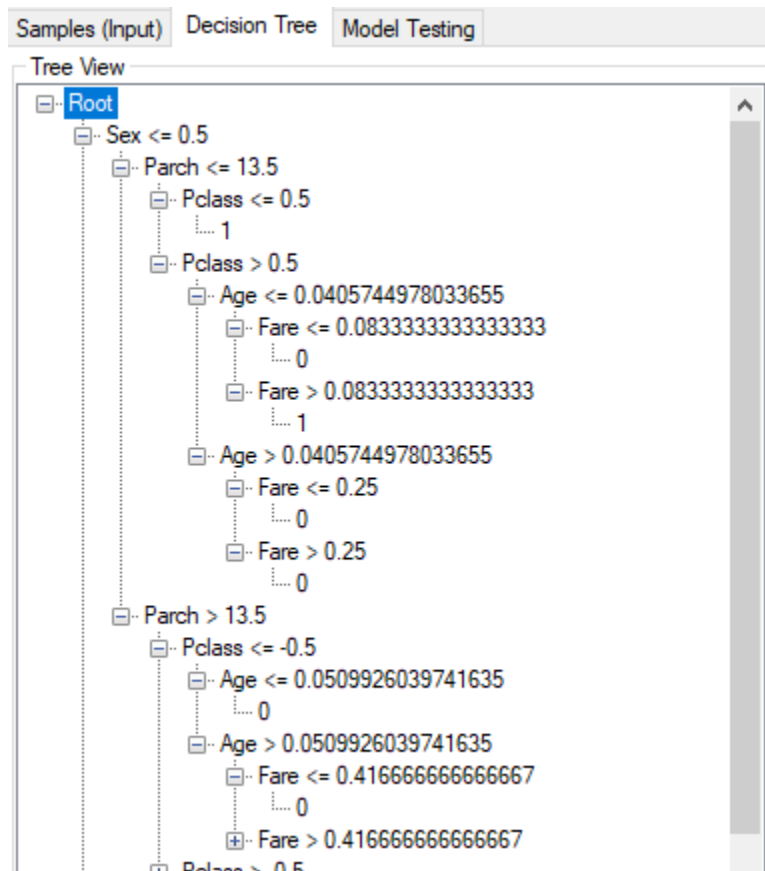
```

Bước 3: Sử dụng thư viện hỗ trợ Zedgraph để xuất ra được mô hình dữ liệu



Hình 5.2 Mô hình biểu diễn tập dữ liệu Titanic

Bước 4: Tiến hành tạo cây nhị phân, kết quả cây nhị phân thu được là



Hình 5.3 Cây nhị phân tập Titanic

Bước 5: Tiến hành test dữ liệu

```
private void btnTestingRun_Click(object sender, EventArgs e)
{
    if (tree == null || dgvTestingSource.DataSource == null)
    {
        MessageBox.Show("Please create a machine first.");
        return;
    }
    // Creates a matrix from the entire source data table
    double[][] table = (dgvLearningSource.DataSource as
    DataTable).ToJagged(out columnNames);
    // Get only the input vector values (first two columns)
    double[][] inputs = table.GetColumns(0, 1, 2, 3, 4, 5);
    // Get the expected output labels (last column)
    int[] expected = table.GetColumn(6).ToInt32();
    // Compute the actual tree outputs
```

```

        int[] actual = tree.Decide(inputs);
        // Use confusion matrix to compute some statistics.
        ConfusionMatrix confusionMatrix = new ConfusionMatrix(actual,
expected, 1, 0);
        dgvPerformance.DataSource = new[] { confusionMatrix };
        // Create performance scatter plot
        CreateResultScatterplot(zedGraphControl1, inputs,
expected.ToDouble(), actual.ToDouble());
    }

```

Kết quả thu được độ chính xác của Decision Tree

Performance Measures							
True Positives	False Negatives	True Negatives	False Positives	Sensitivity	Specificity	Efficiency	Accuracy
249	93	500	49	0.7280701754...	0.9107468123...	0.8194084939...	0.8406285072...

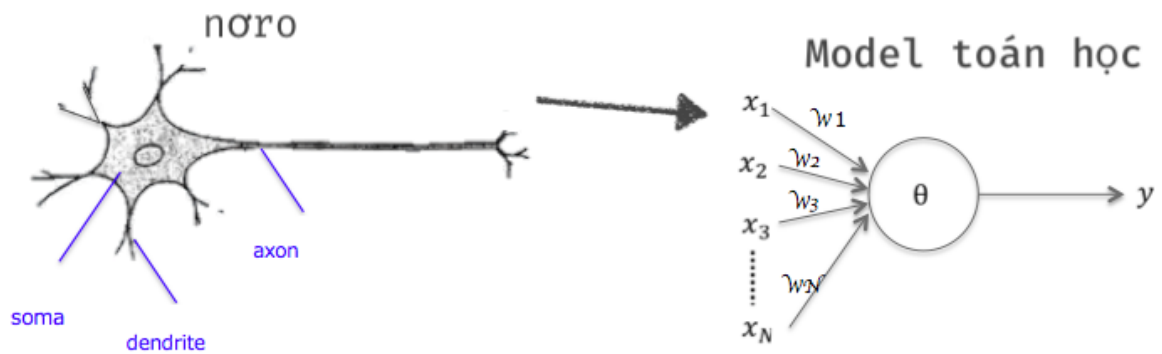
Hình 5.4 Độ chính xác của thuật toán Decision Tree

Nhận xét: có thể thấy độ chính xác của thuật toán decision tree khá cao, rơi vào khoảng 84%.

Chương 6 : NEURAL NETWORK

6.1. Giới thiệu

Neuron là các đơn vị tính toán nhận các input (các sợi nhánh) như các input điện tử (gọi là “spikes”), được di chuyển đến các output (các axon).



Hình 6.1 Mô hình Neuron Network

Trong neural network, các sợi nhánh đóng vai trò như là các input feature x_1, x_2, \dots, x_n và output là kết quả của hypothesis function. Chúng ta sử dụng logistic function như trong phân nhóm (classification), $\frac{1}{1+e^{-\theta^T x}}$. Tuy nhiên, chúng ta đôi khi gọi nó là sigmoid (logistic) activation function. Trong trường hợp này các tham số “theta” này được gọi là “weight”.

Input node x_0 đôi khi được gọi là “bias unit” và luôn có giá trị bằng 1.

6.2. Model representation

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow [\] \rightarrow h_{\theta}(x)$$

Các input node (lớp 1), gọi là “lớp input”, đi qua một lớp node khác (lớp 2) và cuối cùng xuất ra hypothesis function (lớp output).

Neural network có thể có nhiều lớp trung gian giữa lớp input và output. Chúng được gọi là “hidden layers”. Các node trong các lớp hidden layer được label là $a_0^2, a_1^2, \dots, a_n^2$ và được gọi là các “activation node” với:

- $a_i^{(j)}$ là “activation” của đơn vị i trong lớp j .
- $\theta^{(j)}$ là ma trận các “weight” điều khiển control mapping từ lớp j đến lớp $j+1$.

Giá trị của từng “activation” node sẽ được tính như sau (với một hidden layer có 3 activation node):

- $a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$
 - $a_2^{(2)} = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$
 - $a_3^{(2)} = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$
- $\Rightarrow h_{\theta}(x) = a_1^{(3)} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$

Nếu một network có s_j unit trong lớp j và s_{j+1} unit trong lớp $j+1$ thì $\theta^{(j)}$ sẽ có chiều $s_{j+1} \times (s_j + 1)$ (+1 có từ $\theta^{(j)}$ của các “bias node”).

6.3. Cost function

Từ cost function của regularized logistic regression:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Với một network có số lớp là L , có s_j unit trong lớp j và số output unit/class là K thì ta có cost function của neural network như sau:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \left[y_k^{(i)} \log((h_{\theta}(x^{(i)}))_k) + (1 - y_k^{(i)}) \log(1 - (h_{\theta}(x^{(i)}))_k) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\theta_{j,i}^{(l)})$$

6.4. Backpropagation algorithm:

“Backpropagation” là một thuật ngữ của neural network cho việc giảm cost function, như chúng ta đang làm với gradient descent trong logistic và linear regression. Mục tiêu của chúng ta là tính được $\min_{\theta} J(\theta)$ (có nghĩa là tính được đạo hàm từng phần của $J(\theta)$ là $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta)$).

Để làm được điều đó, chúng ta sẽ sử dụng thuật toán như sau:

Với training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} := 0$ với mọi (l, i, j)

Với training example $t = 1$ đến m :

1. Set $a^{(1)} := x^{(t)}$
2. Thực hiện forward propagation để tính $a^{(l)}$ với $l=2, 3, \dots, L$.
3. Sử dụng $y^{(t)}$, tính $\delta^{(L)} = a^{(L)} - y^{(t)}$ với:

L : tổng số lớp trong neural network

$a^{(L)}$: vector của các output của các activation unit cho lớp cuối cùng.

4. Tính $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ bằng cách sử dụng $\delta^{(l)} = ((\theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$

$$5. \quad \Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

Cuối cùng chúng ta cập nhật ma trận Δ mới

- $D_{i,j}^{(l)} := \frac{1}{m} (\Delta_{i,j}^{(l)} + \lambda \theta_{i,j}^{(l)})$ nếu $j \neq 0$.
- $D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)}$ nếu $j = 0$.

$$\Rightarrow \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{i,j}^{(l)}$$

6.5. Random initialization:

Khởi tạo các theta weight với giá trị bằng 0 không phù hợp với các neural network. Khi chúng ta backpropagate, tất cả các node sẽ cập nhật các giá trị không đổi liên tục.

Thay vào đó, chúng ta có thể khởi tạo các weight cho các ma trận θ bằng cách khởi tạo từng $\theta_{ij}^{(l)}$ với một giá trị ngẫu nhiên trong khoảng $[-\epsilon, \epsilon]$.

6.6. Code

6.6.1. Thư viện

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Threading;
using System.Windows.Forms;
using Accord.IO;
using Accord.Math;
using Accord.Neuro;
using Accord.Neuro.Learning;
using Accord.Statistics.Analysis;
using ZedGraph;
```

6.6.2. Cài đặt thuật toán

Bước 1:

```
// Creates a matrix from the source data table
double[,] sourceMatrix = (dgvTestingSource.DataSource as
DataTable).ToMatrix();
double[][] inputs = new double[sourceMatrix.GetLength(0)][];
for (int i = 0; i < inputs.Length; i++)
    inputs[i] = new double[] { sourceMatrix[i, 0],
sourceMatrix[i, 1], sourceMatrix[i, 2],
    sourceMatrix[i, 3], sourceMatrix[i, 4], sourceMatrix[i,
5]};

// Get only the label outputs
int[] expected = new int[sourceMatrix.GetLength(0)];
for (int i = 0; i < expected.Length; i++)
    expected[i] = (int)sourceMatrix[i, 6];
// Compute the machine outputs
int[] output = new int[expected.Length];
for (int i = 0; i < expected.Length; i++)
    output[i] = System.Math.Sign(ann.Compute(inputs[i])[0]);
double[] expectedd = new double[expected.Length];
double[] outputd = new double[expected.Length];
for (int i = 0; i < expected.Length; i++)
{
    expectedd[i] = expected[i];
    outputd[i] = output[i];
}
// Use confusion matrix to compute some statistics.
ConfusionMatrix confusionMatrix = new ConfusionMatrix(output,
expected, 1, -1);
dgvPerformance.DataSource = new List<ConfusionMatrix> {
confusionMatrix };
foreach (DataGridViewColumn col in dgvPerformance.Columns)
col.Visible = true;
Column1.Visible = Column2.Visible = false;

// Create performance scatterplot
CreateResultScatterplot(zedGraphControl1, inputs, expectedd,
outputd);
```

Bước 2: Chạy chương trình và cài đặt các thông số theo ý muốn

Settings

Learning rate: 0.1

Sigmoid's alpha value: 2

Neurons in first layer: 10

☐ Use Bayesian Regularization

☒ Use Nguyen-Widrow Weights

☒ Use always same initialization

Iterations: 100
(0 - infinity)

Current iteration

Iteration: 101

Error: 0.1043503698

Elapsed: 00:00:05.3954602

Start Stop

Hình 6.2 Cài đặt thông số chạy Neuron Network

Độ chính xác và mô hình thu được sau khi train Neural network là:

Performance Measures							
True Positives	False Negatives	True Negatives	False Positives	Sensitivity	Specificity	Efficiency	Accuracy
302	40	533	16	0.883040935...	0.970856102...	0.9269485188...	0.937149270...

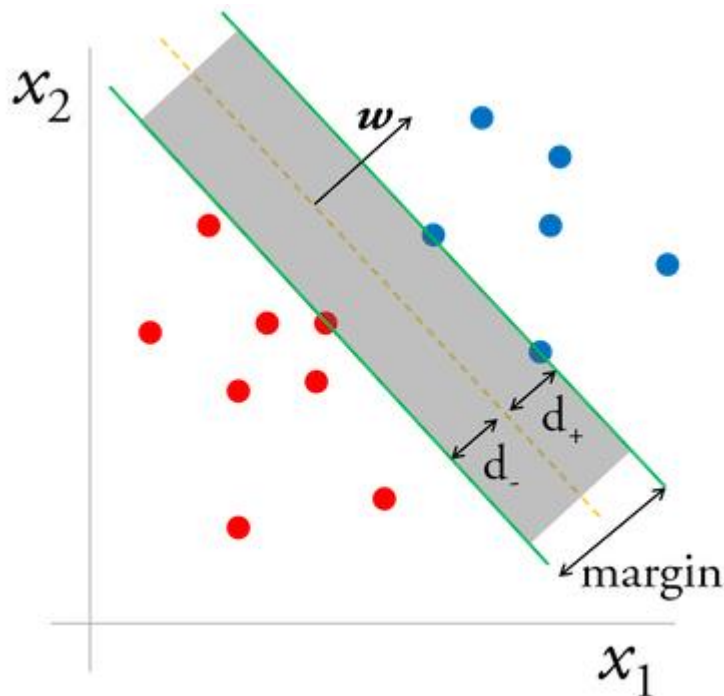
Hình 6.3 Độ chính xác của thuật toán Neuron Network

Nhân xét: có thể thấy rằng độ chính xác của Neuron Network đạt được rất cao, lên đến 93%.

Chương 7 : SUPPORT VECTOR MACHINE (SVM)

7.1. Giới thiệu

Support vector machine (SVM) là thuật toán thuộc nhóm Supervised Learning để phân chia dữ liệu thành các nhóm riêng biệt. Khi phân chia dữ liệu, SVM sẽ học ra được để chia đường tốt nhất cách xa 2 nhóm dữ liệu để khi dữ liệu mới vào sẽ dễ đoán hơn. Đường chia đó được gọi là large-margin classifier.



Hình 7.1 Biểu diễn thuật toán SVM

7.2. Kernel SVM

Khi gặp bài toán phức tạp có nonlinear decision boundary, chúng ta sẽ sử dụng kernel SVM để giải bài toán.

Giải thuật kernel là giải thuật biến đổi không gian của các feature (feature space).

Giả sử dữ liệu training input là: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$. Chọn tất cả \rightarrow dùng điểm đặc biệt (landmark) $\rightarrow mx = ml$

$$(x^{(1)} = l^{(1)}, x^{(2)} = l^{(2)} \dots, x^{(m)} = l^{(m)})$$

Chọn datum $x = f_1 = \text{similarity}(x, l^{(1)}) = e^{\frac{\|x - l_1\|}{2\sigma^2}}$

$$x = f_2 = \text{similarity}(x, l^{(2)})$$

...

$$x = f_m = \text{similarity}(x, l^{(m)})$$

Như vậy chúng ta có dữ liệu training mới:

$(f^{(1)}, y^{(1)}), (f^{(2)}, y^{(2)}), \dots, (f^{(m)}, y^{(m)}) \rightarrow$ Dùng Gradient descent để học

Dự đoán dữ liệu mới: $x \rightarrow f \rightarrow h_{\theta}(x) = y(\theta^T f)$

$$\text{với } (y = 1 \leftrightarrow \theta^T f \gg 0, (y = 0 \leftrightarrow \theta^T f \ll 0))$$

Chú ý: Chọn σ^2 bằng nhau hết (σ^2 nhỏ \rightarrow overfitting, σ^2 lớn \rightarrow underfitting).

7.3. Code

7.3.1. Thư viện

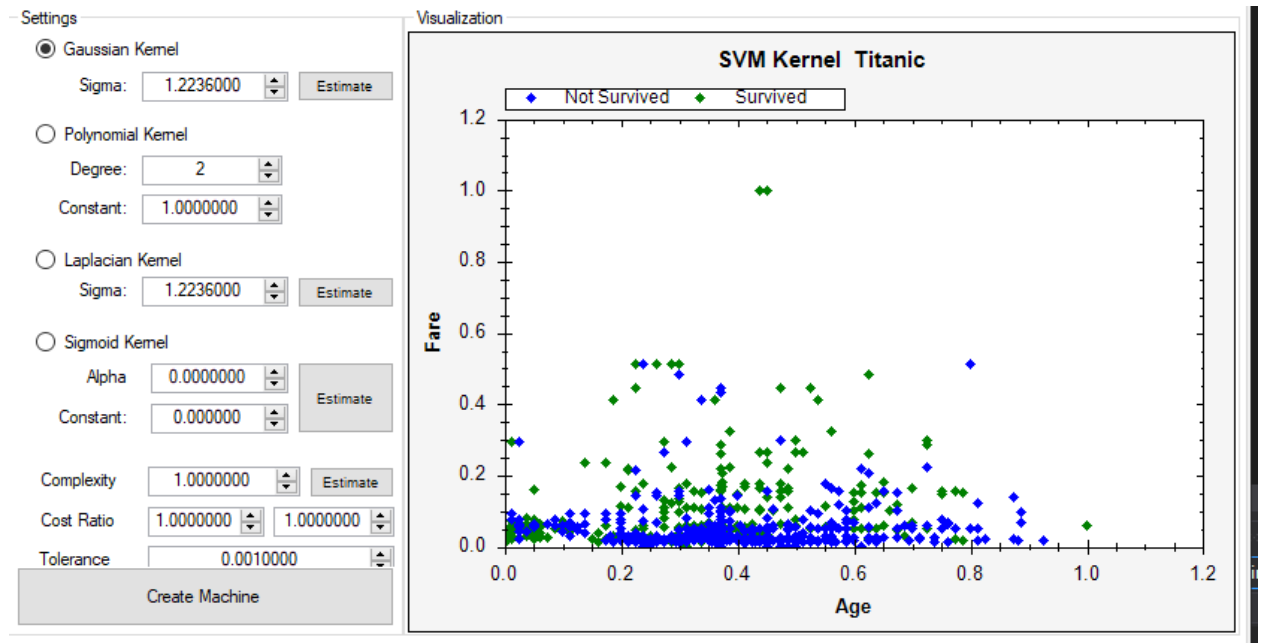
```
using Accord;
using Accord.Controls;
using Accord.IO;
using Accord.MachineLearning.VectorMachines;
using Accord.MachineLearning.VectorMachines.Learning;
using Accord.Math;
using Accord.Statistics;
using Accord.Statistics.Analysis;
using Accord.Statistics.Kernels;
using AForge;
using System;
using System.Data;
```



```
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using ZedGraph;
```

7.3.2. Cài đặt thuật toán

Bước 1: Cài đặt các setting để tiến hành train model



Hình 7.2 Cài đặt thông số train SVM

Bước 2: Test model với tập dữ liệu đã được train

```
// Creates a matrix from the source data table
double[,] table = (dgvTestingSource.DataSource as
DataTable).ToMatrix();
// Extract the first and second columns (X and Y)
double[][] inputs = table.GetColumns(0, 1, 2, 3, 4,
5).ToJagged();
// Extract the expected output labels
bool[] expected = Classes.Decide(table.GetColumn(6));
// Compute the actual machine outputs
```

```

bool[] output = svm.Decide(inputs);
// Use confusion matrix to compute some performance metrics
dgvPerformance.DataSource = new [] { new
ConfusionMatrix(output, expected) };
// Create performance scatter plot
CreateResultScatterplot(zedGraphControl1, inputs,
    expected.ToMinusOnePlusOne().ToDouble(),
    output.ToMinusOnePlusOne().ToDouble());

```

Kết quả thu được:

Performance Measures							
True Positives	False Negatives	True Negatives	False Positives	Sensitivity	Specificity	Efficiency	Accuracy
264	78	488	61	0.771929824...	0.888888888...	0.830409356...	0.843995510...

Hình 7.3 Độ chính xác của thuật toán SVM

Nhận xét: độ chính xác thu được của thuật toán SVM đạt kết quả khá cao ~84%

Chương 8 : KNN (K-nearest Neighbors)

8.1. Giới thiệu

K-nearest neighbor là một thuật toán đơn giản nằm trong nhóm supervised-learning trong Machine Learning. Khi training, K-NN không học một điều gì từ dữ liệu training (thuật toán lazy learning), mọi tính toán thực hiện của thuật toán chỉ nhằm mục đích dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại bài toán Supervised learning là Classification và Regression. Vậy K-nearest neighbor là gì ?

Thuật toán K-NN là thuật toán dành cho máy học. Nó làm việc dựa trên khoảng cách nhỏ nhất từ Object được dự đoán đến từ mẫu training để xác định k-nearest neighbor. Sau đó thuật toán sẽ dựa trên k-nearest neighbor để đưa ra dự đoán object mới cần dự đoán.

8.2. Cách hoạt động

Bước 1: Xác định tham số K = số láng giềng gần nhất

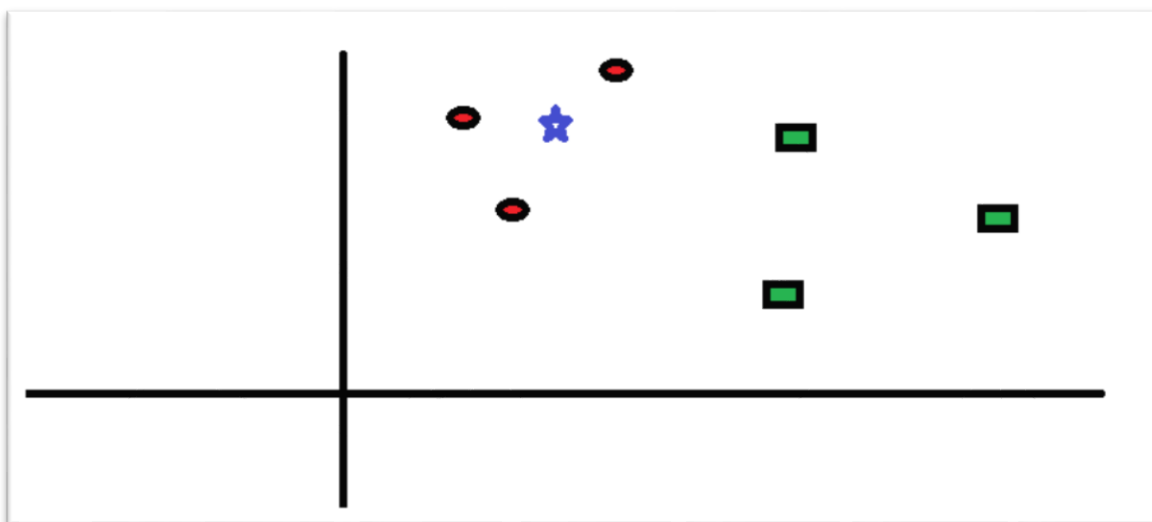
Bước 2: Tính toán khoảng cách giữa mẫu thử và những mẫu training

Bước 3: Sắp xếp khoảng cách và xác định K khoảng cách nhỏ nhất

Bước 4: Thu nhập giá trị thuộc tính của K láng giềng gần nhất

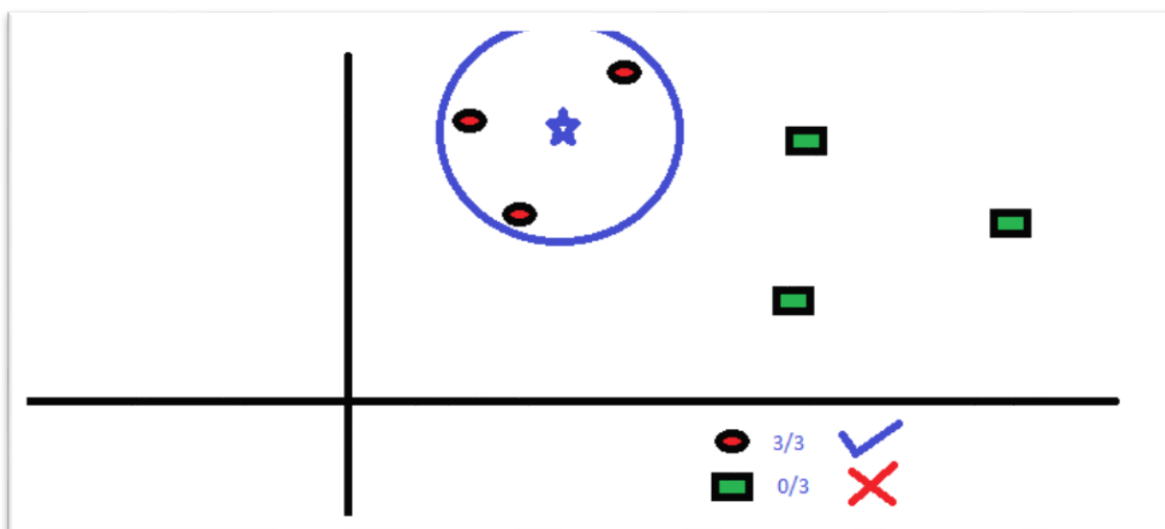
Bước 5: Sử dụng giá trị trung bình của K láng giềng gần nhất để phán đoán giá trị của Object mới

Để hiểu rõ hơn về thuật toán K-NN, ta sẽ thông qua một ví dụ đơn giản sau:



Hình 8.1 Ví dụ thuật toán K-NN-1

Trên hình ta sẽ dự đoán xem ngôi sao xanh (BS) là một vòng tròn đỏ (RC) hay hình vuông xanh (GS) hoặc không là gì. Bây giờ ta xác định K-nearest neighbor với $K = 3$. Do đó, ta sẽ tính khoảng cách và khoanh vùng lại 3 giá trị gần BS nhất (như hình bên dưới).



Hình 8.2 Ví dụ thuật toán K-NN-2

Như hình vẽ 3 điểm gần nhất với BS đều là RC. Do đó, qua thuật toán K-NN chúng ta có thể xác định BS là một hình tròn đỏ (RC).

8.3. Code

8.3.1. Thư viện

```
#k-nearest neighbors (KNN) model
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
```

8.3.2. Cài đặt thuật toán

Bước 1: Cho k chạy từ 1 đến 26 vì chưa xác định được k mang giá trị nào sẽ tối ưu nhất.

```
k_range = range(1,26)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit (X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append (metrics.accuracy_score(y_test, y_pred))
print(scores)
```

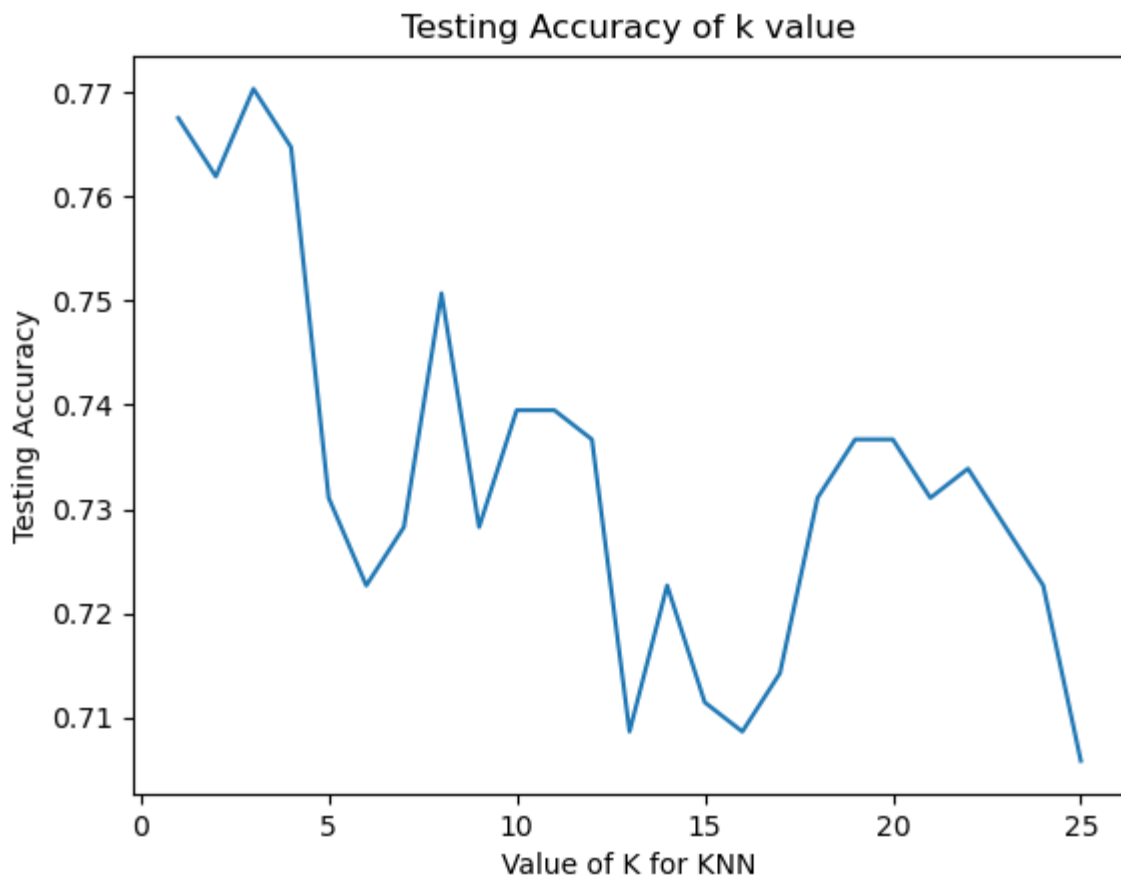
Kết quả: nhận được các giá trị về độ chính xác của k khi chạy từ 1 đến 26

```
[0.7675070028011205,      0.7619047619047619,      0.7703081232492998,
0.7647058823529411,      0.7310924369747899,      0.7226890756302521,
0.7282913165266106,      0.7507002801120448,      0.7282913165266106,
0.7394957983193278,      0.7394957983193278,      0.7366946778711485,
0.7086834733893558,      0.7226890756302521,      0.711484593837535,
0.7086834733893558,      0.7142857142857143,      0.7310924369747899,
0.7366946778711485,      0.7366946778711485,      0.7310924369747899,
0.7338935574229691,      0.7282913165266106,      0.7226890756302521,
0.7058823529411765]
```

Bước 2: Sử dụng thư viện matplotlib.pyplot để kiểm tra độ chính xác của k và tiến hành tìm giá trị k tối ưu nhất

```
import matplotlib.pyplot as plt
# Data for plotting
fig, ax = plt.subplots()
ax.plot(k_range, scores)
ax.set(xlabel='Value of K for KNN', ylabel='Testing Accuracy',
       title='Testing Accuracy of k value')
fig.savefig("test.png")
plt.show()
```

Biểu đồ nhận được về giá trị k và độ chính xác của k thu được là:



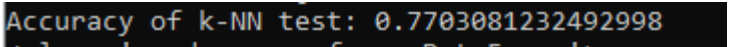
Hình 8.3 Biểu đồ kiểm tra độ chính xác của k

Nhận xét: qua biểu đồ trên, ta có thể thấy rằng k mang giá trị 3 đem lại độ chính xác cao nhất

Bước 3: Sau khi đã xác định được giá trị tối ưu của k, ta tiến hành train dữ liệu

```
#K value equal 3 has the highest accuracy rate.
knn = KNeighborsClassifier (n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print('Accuracy of k-NN test: {}'.format(metrics.accuracy_score(y_test,
y_pred)))
```

Kết quả: độ chính xác của thuật toán sau khi train dữ liệu nhận được là



```
Accuracy of k-NN test: 0.7703081232492998
```

Hình 8.4 Độ chính xác của thuật toán K-NN

Bước 4: Sử dụng mô hình này để tiến hành các dự đoán ngoài dữ liệu mẫu.

```
#Finally, we can use this model to make predictions on out of sample data.
And generate a new data frame 'submission'.
df_test_set.info()
test = df_test_set.iloc[:,1:5].values
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit (X,Y)
Y_pred = knn.predict(test)
```

Bước 5: Xuất tập dữ liệu mới có tên là knn_submission

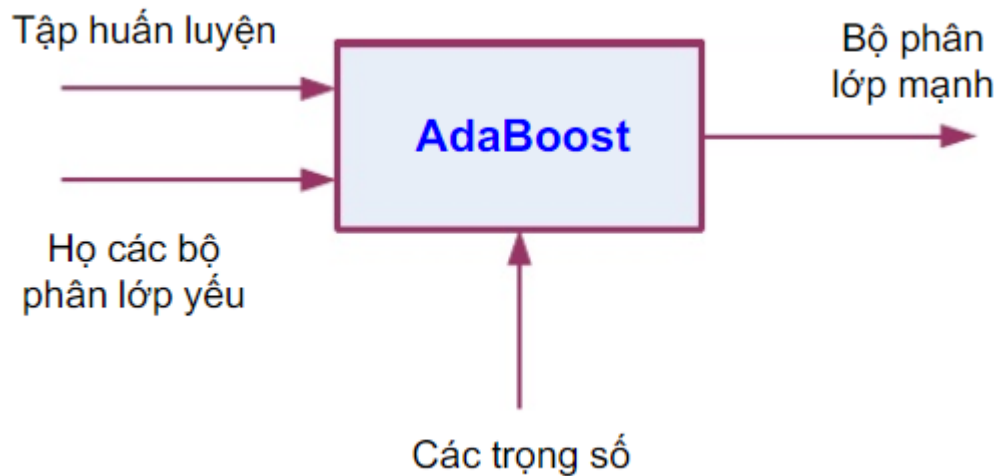
```
knn_submission = pd.DataFrame({
    "PassengerId" : df_test_set["PassengerId"],
    "Survived" : Y_pred
})
#export to csv
knn_submission.to_csv("C:/Users/Administrator/Desktop/titanic/titanic/knn_s
ubmission.csv", encoding='utf-8')
```

Chương 9 : ADA BOOST

9.1. Giới thiệu

AdaBoost là một trong những thuật toán ensemble learning, một thuật toán boosting dùng để xây dựng bộ phân lớp (classifier). Ý tưởng của thuật toán này là kết hợp các thuật toán yếu thành một thuật toán mạnh, ban đầu các instance được gán trọng số như nhau, nhưng sau khi classifier bằng một thuật toán đơn giản như (các thuật toán tree chẳng hạn) những instance nào có classifier sai sẽ được gán trọng số cao hơn và ngược lại.

9.2. Cách thức hoạt động



Hình 9.1 Cách thức hoạt động của thuật toán Ada Boost

9.3. Ưu, nhược điểm

Đây là thuật toán đơn giản và dễ dàng cài đặt. Thêm vào đó, tốc độ học rất nhanh. Các weak learner đơn giản hơn rất nhiều các strong learner, nhờ vậy thuật toán chạy nhanh hơn.

Một điều nữa, AdaBoost là phương pháp có khả năng điều chỉnh các classifier rất tinh tế. Vì mỗi lần chạy AdaBoost lại tinh chỉnh lại các trọng số cho các learner tốt nhất.

Cuối cùng, đây là thuật toán linh hoạt và đa năng. AdaBoost có thể kết hợp với bất kỳ thuật toán học máy nào và nó có thể làm việc với một lượng lớn dữ liệu khác nhau.

9.4. Code

9.4.1. Thư viện

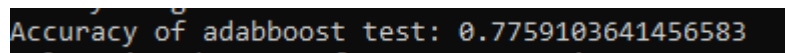
```
#Adaboost
from sklearn.ensemble import AdaBoostClassifier
```

9.4.2. Cài đặt thuật toán

Bước 1:

```
# Instantiate our model
adaboost = AdaBoostClassifier()
# Fit our model to the training data
adaboost.fit(X_train, y_train)
# Predict on the test data
adaboost_predictions = adaboost.predict(X_test)
print('Accuracy of adaboost test: {}'.format(metrics.accuracy_score(y_test,
adaboost_predictions)))
```

Kết quả: độ chính xác của thuật toán Ada Boost thu được là:



```
Accuracy of adaboost test: 0.7759103641456583
```

Hình 9.2 Độ chính xác của thuật toán Ada Boost

Bước 2: Sử dụng mô hình này để tiến hành các dự đoán ngoài dữ liệu mẫu

```
df_test_set.info()
test = df_test_set.iloc[:,1:5].values
adaboost = AdaBoostClassifier()
adaboost.fit (X,Y)
adaBoost_pred = adaboost.predict(test)
```

Bước 3: Xuất ra tập dữ liệu mới có tên là AdaptiveBoosting_SS_OH_FE.csv

```
adaboost_submission = pd.DataFrame({
    "PassengerId" : df_test_set["PassengerId"],
    "Survived" : adaBoost_pred
})
adaboost_submission.to_csv(

'C:/Users/Administrator/Desktop/titanic/titanic/AdaptiveBoosting_SS_OH_FE.csv')
```

Chương 10 : NAIVE BAYES

10.1. Định nghĩa

Thuật toán Naïve Bayes Classification (NBC) là một thuật toán phân loại dựa trên tính toán xác suất áp dụng định lý Bayes, thuật toán này thuộc nhóm Supervised Learning (Học có giám sát).

10.2. Thuật toán

Theo định lý Bayes, ta có công thức tính xác suất ngẫu nhiên của sự kiện y khi biết x như sau:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Giả sử ta phân chia 1 sự kiện x thành n thành phần khác nhau x_1, x_2, \dots, x_n . Từ đó ta có thể tính được:

$$P(x|y) = P(x_1 \cap x_2 \cap \dots \cap x_n | y) = P(x_1 | y) P(x_2 | y) \dots P(x_n | y)$$

Do đó ta có:

$$P(y|x) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

- α là phép tỉ lệ thuận

Cách xác định các thành phần của dữ liệu dựa trên giả thiết này có tên là Naive Bayes

10.3. Các mô hình thuật toán Navie Bayes

Một trong các bài toán nổi tiếng hiệu quả khi sử dụng Navie Bayes là bài toán phân loại text, trong đó phổ biến nhất là bài toán phân loại thư rác. Do đó hiện nay có 2 mô hình thuật toán Navie Bayes thường được sử dụng là:

- Mô hình Gaussian
- Mô hình Bernoulli
- Mô hình Multinomial

➤ Công thức Bernoulli

Ở mô hình này, các feature vector là các giá trị nhị phân 0,1. Trong đó 1 thể hiện từ có xuất hiện trong văn bản, 0 thể hiện từ đó không xuất hiện trong văn bản.

Xác suất $P(x_i | y)$ được tính bằng:

$$P(x_i | y) = P(i | y) \times x_i + (1 - P(i | y)) \times (1 - x_i)$$

Với $P(i | y)$ là tỉ lệ số lần từ x_i xuất hiện trong toàn bộ tập training data có nhãn y .

10.4. Code

10.4.1. Thư viện

```
using Accord.Controls;  
using Accord.IO;  
using Accord.MachineLearning.Bayes;  
using Accord.Math;  
using Accord.Statistics.Analysis;  
using Accord.Statistics.Distributions.Univariate;  
using Components;  
using System;  
using System.Collections.Generic;  
using System.Data;  
using System.Drawing;
```

```
using System.IO;
using System.Windows.Forms;
using ZedGraph;
```

10.4.2. Cài đặt thuật toán

```
// Creates a matrix from the source data table
double[,] table = (dgvLearningSource.DataSource as
DataTable).ToMatrix();
// Get only the input vector values
double[][] inputs = table.Get(null, 0, 6).ToJagged();
// Get only the label outputs
int[] expected = new int[table.GetLength(0)];
for (int i = 0; i < expected.Length; i++)
    expected[i] = (int)table[i, 6];
// Compute the machine outputs
int[] output = bayes.Decide(inputs);
// Use confusion matrix to compute some statistics.
ConfusionMatrix confusionMatrix = new ConfusionMatrix(output,
expected, 1, 0);
dgvPerformance.DataSource = new List<ConfusionMatrix> {
confusionMatrix };

foreach (DataGridViewColumn col in dgvPerformance.Columns)
    col.Visible = true;
Column1.Visible = Column2.Visible = true;
// Create performance scatter plot
CreateResultScatterplot(zedGraphControl1, inputs,
expected.ToDouble(), output.ToDouble());
```

Kết quả:

Performance Measures									
R ² (r-squared)	Error m-of-square	True Positives	False Negatives	True Negatives	False Positives	Sensitivity	Specificity	Efficiency	Accuracy
	0.218855...	241	101	455	94	0.704678...	0.828779...	0.766728...	0.781144...

Hình 10.1 Độ chính xác của thuật toán Naive bayes

Nhận xét: độ chính xác thu được sau khi train tập dữ liệu titanic với thuật toán Naïve Bayes là trung bình ~ 78%

Chương 11 : K-MEAN

11.1. Giới thiệu

K-means là một thuật toán phân cụm thuộc nhóm unsupervised learning (thuật toán không có label, chỉ có dữ liệu đầu vào). Mục đích của thuật toán là tìm ra được tất cả các nhóm trong dữ liệu dựa theo thông số K (số lượng nhóm cần phân).

Thuật toán hoạt động dựa trên sự lặp đi lặp lại để gán từng điểm dữ liệu cho một trong các nhóm K dựa trên các giá trị cung cấp. Điểm dữ liệu được phân cụm dựa trên tính tương tự giữa các features. Kết quả mong muốn của thuật toán K-means là :

- Tìm được điểm dữ liệu trung tâm của K nhóm, để sau này sử dụng như là một label cho dữ liệu mới.
- Tìm được labels cho dữ liệu training (mỗi điểm dữ liệu là một nhóm đơn).

11.2. Cách hoạt động

Thuật toán K-means sử dụng vòng lặp để sàng lọc tạo ra kết quả cuối cùng. Đầu vào của thuật toán là số lượng cụm và tập dữ liệu. thuật toán bắt đầu với k điểm trọng tâm có thể được tạo ngẫu nhiên hoặc chọn ngẫu nhiên từ tập dữ liệu được chia thành k nhóm. Sau đó thuật toán lặp lại các bước sau:

- Bước 1: Tính khoảng cách từ mỗi điểm đến trọng tâm (cx, cy) của từng nhóm sử dụng công thức tính khoảng cách:

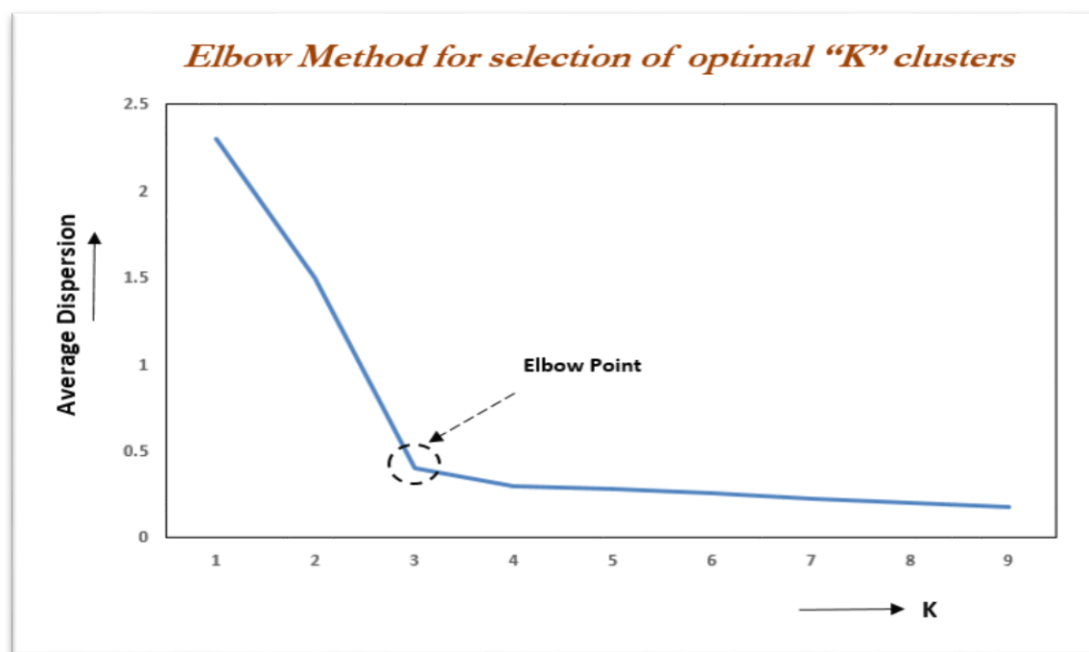
$$d(a, b)^2 = (ax - bx)^2 + (ay - by)^2$$

- Bước 2: Đối với mỗi điểm A, cho vào nhóm có khoảng cách từ trọng tâm nhóm đó đến điểm A là gần nhất
- Bước 3: Tính lại tọa độ trọng tâm cho mỗi nhóm.

11.3. Chọn giá trị k

Để tìm số cụm (giá trị K) trong dữ liệu, người dùng cần chạy thuật toán phân cụm K-means trong một phạm vi giá trị K và so sánh kết quả giữa chúng. Nói chung, không có phương pháp nào để xác định giá trị chính xác của K , nhưng có thể thu được ước tính chính xác bằng các kỹ thuật sau:

Một trong những số liệu để so sánh kết quả giữa các giá trị K là khoảng cách trung bình giữa các điểm trọng tâm và các điểm dữ liệu của chúng. Vì tăng số lượng cụm nhóm sẽ luôn giảm khoảng cách đến các điểm dữ liệu, đến khi mỗi điểm là một nhóm. Do đó, số lượng này không thể sử dụng để so sánh. Thay vào đó, khoảng cách trung bình đến trọng tâm của K nhóm sẽ được vẽ ra một hàm mà ở đó ta chọn điểm “khủy tay” có tốc độ giảm mạnh đáng kể, được chọn làm giá trị K với độ chính xác là cao nhất.



Hình 11.1 Chọn giá trị K theo phương pháp Elbow point

11.4. Ứng dụng

Thuật toán phân cụm K-means được dùng để tìm các nhóm chưa có label rõ ràng trong dữ liệu. Khi thuật toán được chạy và các nhóm được xác định, mọi dữ liệu mới có thể dễ dàng được gán cho nhóm chính xác. Một số ví dụ về các trường hợp sử dụng là:

- Phân tích hành vi
- Phân nhóm theo lịch sử mua hàng
- Xác định tính cách con người dựa trên sở thích
- Phân loại hàng tồn kho

11.5. Code

11.5.1. Thư viện

```
#kmean
from sklearn import datasets, cluster
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

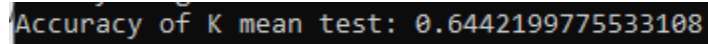
11.5.2. Cài đặt thuật toán

```
kmeans = cluster.KMeans(n_clusters=2) # You want cluster the passenger
records into 2: Survived or Not survived
kmeans.fit(X)
cluster.KMeans(algorithm='auto', copy_x=True, init='k-means++',
max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
```



```
if prediction[0] == Y[i]:  
    correct += 1  
print('Accuracy of K mean test: {}'.format(correct/len(X)));
```

Nhận xét: Độ chính xác của thuật toán K-mean thu được thấp hơn các thuật toán khác ~ 65%

A screenshot of a terminal window with a black background and white text. The text reads: "Accuracy of K mean test: 0.6442199775533108".

```
Accuracy of K mean test: 0.6442199775533108
```

Hình 11.2 Độ chính xác của thuật toán K-mean

Chương 12 : RANDOM FOREST

12.1. Giới thiệu

Random Forest là một thuật toán học có giám sát Supervised Learning, tập hợp của các Decision Tree, mà mỗi cây được chọn theo một thuật toán dựa vào ngẫu nhiên

Điểm mạnh:

Random Forest algorithm có thể sử dụng cho cả bài toán Classification và Regression

Random Forest làm việc được với dữ liệu thiếu giá trị

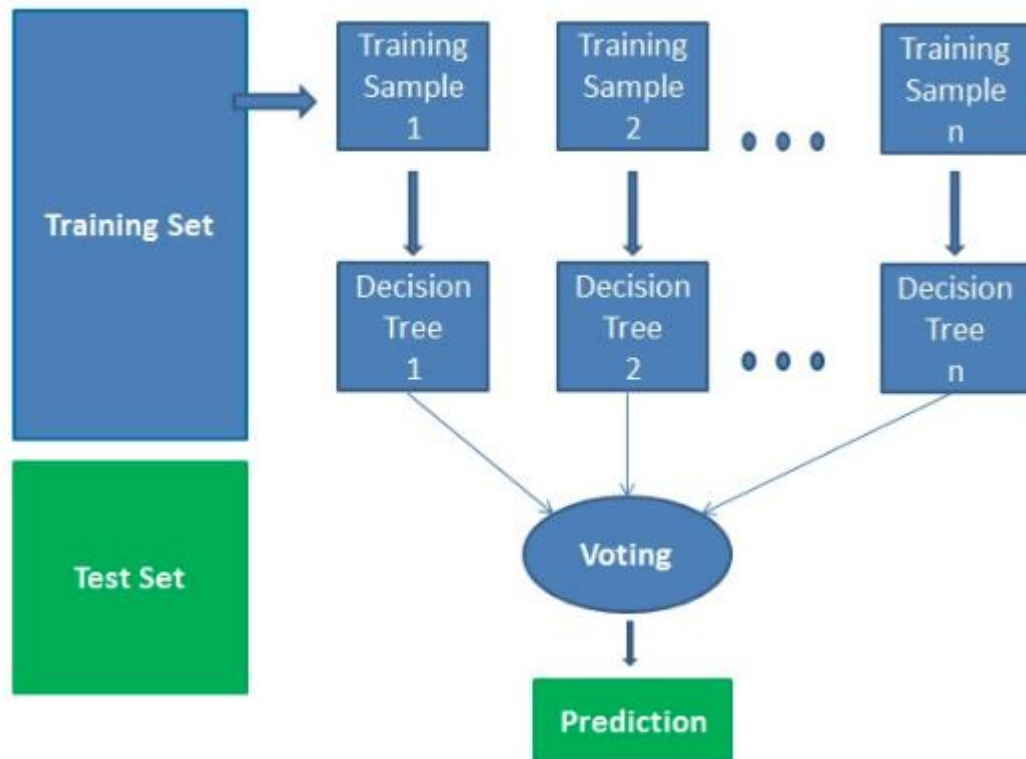
Khi Forest có nhiều cây hơn, ta có thể tránh được việc Overfitting với tập dữ liệu

Có thể tạo mô hình cho các giá trị phân loại

12.2. Cách hoạt động

Nó hoạt động theo bốn bước:

- Bước 1: Chọn các mẫu ngẫu nhiên từ tập dữ liệu đã cho.
- Bước 2: Thiết lập cây quyết định cho từng mẫu và nhận kết quả dự đoán từ mỗi quyết định cây.
- Bước 3: Hãy bỏ phiếu cho mỗi kết quả dự đoán.
- Bước 4: Chọn kết quả được dự đoán nhiều nhất là dự đoán cuối cùng.



Hình 12.1 Cách thức hoạt động của thuật toán Random Forest

12.3. Code

12.3.1. Thư viện

```
#random forest  
from sklearn.ensemble import RandomForestClassifier
```

12.3.2. Cài đặt thuật toán

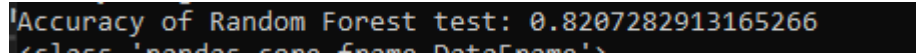
Bước 1: Chọn các thông số, $n_estimators = 100$

```
random_forest = RandomForestClassifier(n_estimators=100)
```

Bước 2: Tiến hành train dataset

```
random_forest.fit(X_train, y_train)
random_forest_predictions = random_forest.predict(X_test)
print('Accuracy of Random Forest test: {}'.format(
    metrics.accuracy_score(y_test, random_forest_predictions)))
```

Kết quả thu được:



```
'Accuracy of Random Forest test: 0.8207282913165266
<class 'pandas.core.frame.DataFrame'>
```

Hình 12.2 Độ chính xác của thuật toán Random Forest

Nhận xét: độ chính xác thu được của thuật toán Random forest đạt được khá cao ~82%

Bước 3: Sử dụng mô hình này để tiến hành các dự đoán ngoài dữ liệu mẫu

```
df_test_set.info()
test = df_test_set.iloc[:,1:5].values
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit (X,Y)
random_forest_pred = random_forest.predict(test)
```

Bước 4: Xuất ra tập dữ liệu mới có tên là RandomForest_SS_OH.csv

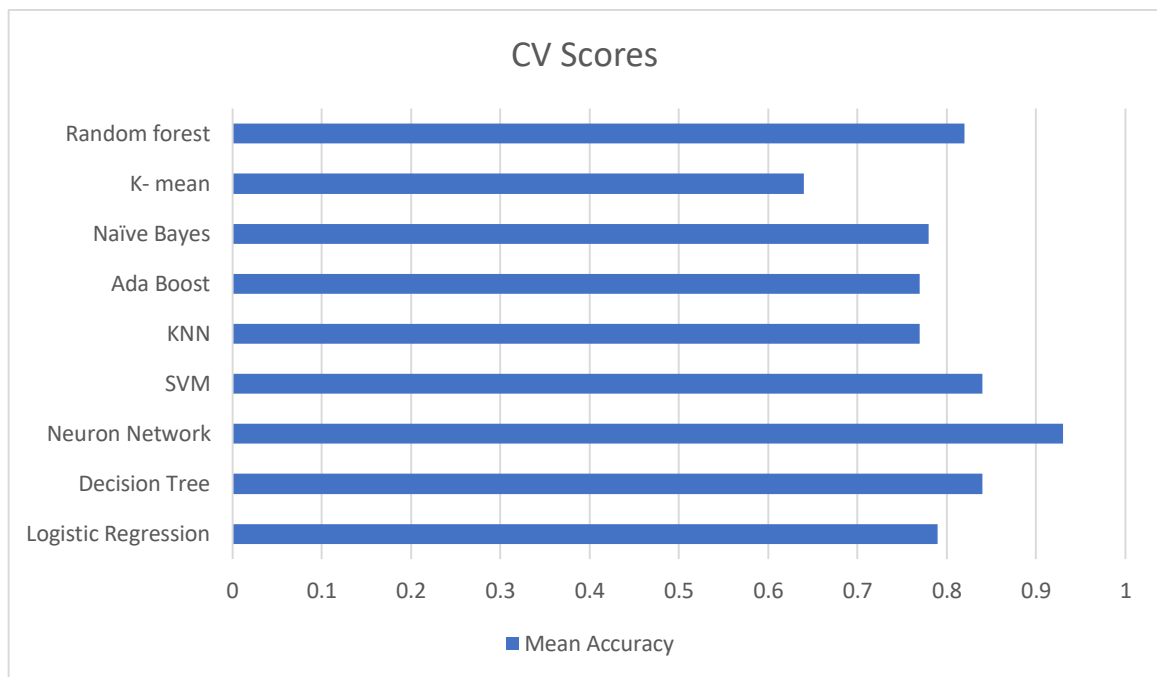
```
random_forest_submission = pd.DataFrame({
    "PassengerId" : df_test_set["PassengerId"],
    "Survived" : random_forest_pred
})
random_forest_submission.to_csv(

'C:/Users/Administrator/Desktop/titanic/titanic/RandomForest_SS_OH.csv')
```

Chương 13 : ĐÁNH GIÁ CÁC THUẬT TOÁN

Thông qua việc train tập dữ liệu titanic với 9 thuật toán đã học, có thể thấy rằng thuật toán K-Mean cho độ chính xác thấp nhất, thuật toán cho độ chính xác cao nhất là Neuron Network

Các thuật toán khác như Random Forest, SVM, Decision Tree cho độ chính xác cao, các thuật toán còn lại như Naive Bayes, Ada Boost, KNN, Logistic Regression cho độ chính xác trung bình



Hình 13.1 Biểu đồ đánh giá mức độ chính xác của các thuật toán ML

TÀI LIỆU THAM KHẢO

1. <https://medium.com/analytics-vidhya/titanic-machine-learning-by-k-nearest-neighbors-knn-algorithm-530d8bdd8323>
2. <https://medium.com/@praveen.orvakanti/this-will-help-you-score-95-percentile-in-the-kaggle-titanic-ml-competition-aa2b3fd1b79b>