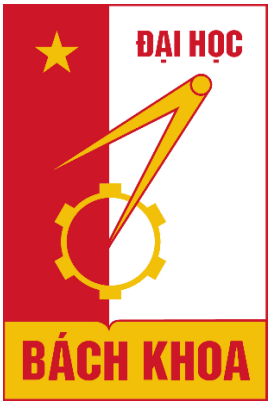
ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN TRUYỀN THÔNG



BÁO CÁO ĐỒ ÁN LẬP TRÌNH

Lập trình Python minh họa một số thuật toán Học máy

Họ tên sinh viên/ học viên thực hiện

Nguyễn Duy Cương 20191714

CLC HTTT&TT K64

Giảng viên hướng dẫn

Trịnh Văn Loan

Hà Nội 2022

DANH MỤC CÁC CÔNG VIỆC

Nội dung : Lập trình Python minh họa một số thuật toán Học máy

STT	Tên nội dung công việc	Người thực hiện
1	1.1 Tìm hiểu về Python và Machine Learning	
2	1.2 Thuật toán Machine Learning	

MỤC LỤC

MỞ ĐẦU **2**

CHƯƠNG 1: Tìm hiểu về Python và Machine Learning……. 3

1.1. Giới thiệu về ngôn ngữ lập trình Python 3

1.1.1. Giới thiệu về Python và môi trường cài đặt 3

1.1.2. Cấu trúc dữ liệu trong Python 3

1.1.3. Thư viện trong Python 4

1.2. Giới thiệu về Machine Learning 5

1.3. Dữ liệu trong Machine Learning 6

1.3.1. Làm việc với file .csv 6

1.3.2. Hình dung dữ liệu 7

1.3.3. Chuyển đổi dữ liệu 8

CHƯƠNG 2: Thuật toán Machine Learning 10

2.1. Đánh giá hiệu năng trong thuật toán Machine Learning 10

2.2. Các chỉ số đánh giá thuật toán Machine Learning 12

2.3. Linear Regression 14

2.4. Logistic Regression 16

2.5. K Nearest Neighbors 18

2.6. Naive Bayes 21

2.7. Decision Tree 25

KẾT LUẬN **31**

TÀI LIỆU THAM KHẢO **32**

MỞ ĐẦU

Những năm gần đây, AI - Artificial Intelligence (Trí Tuệ Nhân Tạo), và cụ thể hơn là Machine Learning (Học Máy hoặc Máy Học) nổi lên như một bằng chứng của cuộc cách mạng công nghiệp lần thứ tư (1 - động cơ hơi nước, 2 - năng lượng điện, 3 - công nghệ thông tin). Trí Tuệ Nhân Tạo đang len lỏi vào mọi lĩnh vực trong đời sống mà có thể chúng ta không nhận ra. Xe tự hành của Google và Tesla, hệ thống tự tag khuôn mặt trong ảnh của Facebook, trợ lý ảo Siri của Apple, hệ thống gợi ý sản phẩm của Amazon, hệ thống gợi ý phim của Netflix, máy chơi cờ vây AlphaGo của Google DeepMind, …, chỉ là một vài trong vô vàn những ứng dụng của AI/Machine Learning. Vì vậy việc tìm hiểu và áp dụng Machine Learning vào những bài toán thực tế đang ngày trở thành một lĩnh vực lớn mạnh và thu hút.
Trong chuyên đề này chúng ta sẽ minh họa một số giải thuật học máy bằng Python. Chuyên đề này được tổ chức như sau:

Chương 1: TÌM HIỂU VỀ PYTHON VÀ MACHINE LEARNING. Chương này nghiên cứu tìm hiểu ngôn ngữ lập trình Python gồm các cấu trúc dữ liệu và các thư viện sử dụng. Tiếp theo báo cáo giới thiệu về Machine Learning và phân loại thuật toán Học máy. Đồng thời tiếp cận cách sử dụng dữ liệu trong Machine Learning.

Chương 2: MACHINE LEARNING ALGORITHMS. Trong chương này giới thiệu minh họa về một số thuật toán học máy cũng như cách đánh giá hiệu năng của chúng.

CHƯƠNG 1: TÌM HIỂU VỀ PYTHON

1.1. Giới thiệu về ngôn ngữ lập trình Python

1.1.1. Giới thiệu về Python và môi trường cài đặt

Ngôn ngữ Python là loại ngôn ngữ cấp cao, nằm trong số các ngôn ngữ lập trình có mục đích chung phổ biến nhất. Đây là một trong những ngôn ngữ lập trình phát triển nhanh nhất thế giới và được sử dụng bởi các kỹ sư phần mềm, nhà toán học, nhà phân tích dữ liệu, nhà khoa học, kỹ sư mạng, sinh viên và kế toán.

Python là một ngôn ngữ lập trình được thông dịch, hướng đối tượng và cấp cao. Nó được gọi là ngôn ngữ thông dịch vì mã nguồn của nó được biên dịch thành mã bytecode sau đó sẽ được thông dịch.[1]. Trình thông dịch Python có sẵn trên một số hệ điều hành như Linux, macOS và Windows. Mở một dòng lệnh trên cmd và nhập: python –version để kiểm tra phiên bản Python hiện tại.

Python là một ngôn ngữ linh hoạt, có nghĩa là nó có thể được sử dụng trong nhiều ứng dụng khác nhau, từ Blender (phần mềm mô hình 3D) đến phát triển web. Chúng ta có thể viết mã Python bằng shell cho các dự án nhỏ. Tuy nhiên, nếu muốn làm việc trên các dự án lớn hơn, chắc chắn cần sử dụng trình soạn thảo mã chuyên dụng hoặc môi trường phát triển tích hợp (IDE). Mỗi trình soạn thảo mã code hoặc Python IDE khác nhau về các tính năng, giao diện người dùng, v.v. Trong báo cáo này, những minh họa về thuật toán được tiến hành trên Jupyter Notebook - một ứng dụng web mã nguồn mở mà bạn có thể sử dụng để tạo và chia sẻ tài liệu có chứa mã trực tiếp, phương trình, trực quan hóa và văn bản. [2]

1.1.2. Cấu trúc dữ liệu trong Python

Một số kiểu dữ liệu như :

- String : data = ‘ hello world ‘
- Number : value = 789
- Boolean : a = True

- Multiple Assignment : a,b,c = 1,2,3
- No Value : a = None

Các cấu trúc điều khiển như :

- If-Then-Else Conditional : if 1: continue else: break
- For-Loop : for x in range(10): print x
- While-Loop : for x < 10 : print x

Các cấu trúc dữ liệu như:

- Tuple : a = (1,2,3)
- List : my_list = {1,2,3}
- Dictionary : mydict = {'a': 1, 'b': 2, 'c': 3}
- Function : def myfunc(): return 0

1.1.3. Một số thư viện trong Python

NumPy cung cấp các cấu trúc dữ liệu nền tảng, các hoạt động cho SciPy. Có thể dễ dàng chuyển đổi một list trong Python thành một numpy array. Ký hiệu mảng và phạm vi có thể được sử dụng để truy cập dữ liệu một cách hiệu quả trong mảng NumPy và mảng NumPy có thể được sử dụng trực tiếp trong số học.

Ví dụ,

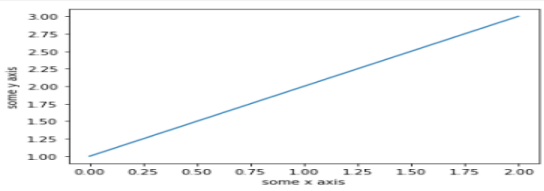
```
# define an array
import numpy
mylist = [1, 2, 3]
myarray = numpy.array(mylist)
myarray2 = numpy.array([3, 3, 3])

print(myarray)
print(myarray.shape)
print(("Specific row and col: %s") % myarray[-1])
print(("Addition: %s") % (myarray + myarray2))
```

```
[1 2 3]
(3,)
Specific row and col: 3
Addition: [4 5 6]
```

Matplotlib có thể được sử dụng để tạo các đồ thị và biểu đồ.

Ví dụ,

```
import matplotlib.pyplot as plt
import numpy
myarray = numpy.array([1, 2, 3])
plt.plot(myarray)
plt.xlabel('some x axis')
plt.ylabel('some y axis')
plt.show()
```



Pandas cung cấp cấu trúc dữ liệu và chức năng để thao tác và phân tích dữ liệu một cách nhanh chóng. Chìa khóa để hiểu Pandas cho học máy là hiểu cấu trúc dữ liệu Series và DataFrame. Series là mảng một chiều trong đó các hàng và cột có thể được gắn nhãn, DataFrame là một mảng đa chiều trong đó các hàng và cột có thể được gắn nhãn.

Ví dụ,

```
import numpy
import pandas

yarray = numpy.array([1, 2, 3])
rownames = ['a', 'b', 'c']
myseries = pandas.Series(myarray, index=rownames)

myarray = numpy.array([[1, 2, 3], [4, 5, 6]])
rownames = ['a', 'b']
colnames = ['one', 'two', 'three']
mydataframe = pandas.DataFrame(myarray, index=rownames, columns=colnames)
print(myseries, mydataframe)
```

```
a    1
b    2
c    3
dtype: int32    one  two  three
a    1    2      3
b    4    5      6
```

1.2. Giới thiệu về Machine Learning

Machine Learning là một tập con của AI. Theo định nghĩa của Wikipedia, Machine learning is the subfield of computer science that "gives computers the ability to learn without being explicitly programmed". Nói đơn giản, Machine Learning là một lĩnh vực nhỏ của Khoa Học Máy Tính, nó có khả năng tự học hỏi dựa trên dữ liệu đưa vào mà không cần phải được lập trình cụ thể. [3]

Có hai cách phổ biến phân nhóm các thuật toán Học máy. Một là dựa trên phương thức học (learning style), hai là dựa trên chức năng (function) (của mỗi thuật toán).

A, Theo phương thức học, các thuật toán Machine Learning thường được chia làm 4 nhóm: Supervised learning, Unsupervised learning, Semi-supervised learning và Reinforcement learning. Có một số cách phân nhóm không có Semi-supervised learning hoặc Reinforcement learning.

- Supervised learning là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới (new input) dựa trên các cặp (input, outcome) đã biết từ trước. Cặp dữ liệu này còn được gọi là (data, label), tức (dữ liệu, nhãn). Supervised learning là nhóm phổ biến nhất trong các thuật toán Machine Learning
- Trong thuật toán Unsupervised learning, chúng ta không biết được *outcome* hay *nhãn* mà chỉ có dữ liệu đầu vào. Thuật toán unsupervised learning sẽ dựa vào cấu trúc của dữ liệu để thực hiện một công việc nào đó,

ví dụ như phân nhóm (clustering) hoặc giảm số chiều của dữ liệu (dimension reduction) để thuận tiện trong việc lưu trữ và tính toán.

- Các bài toán khi chúng ta có một lượng lớn dữ liệu X nhưng chỉ một phần trong chúng được gán nhãn được gọi là Semi-Supervised Learning. Những bài toán thuộc nhóm này nằm giữa hai nhóm được nêu bên trên.
- Reinforcement learning là các bài toán giúp cho một hệ thống tự động xác định hành vi dựa trên hoàn cảnh để đạt được lợi ích cao nhất (maximizing the performance). Hiện tại, Reinforcement learning chủ yếu được áp dụng vào Lý Thuyết Trò Chơi (Game Theory), các thuật toán cần xác định nước đi tiếp theo để đạt được điểm số cao nhất.

B, Có một cách phân nhóm thứ hai dựa trên chức năng của các thuật toán.

- Regression Algorithms
- Classification Algorithms
- Clustering Algorithms
- Instance-based Algorithms,….[3]

1.3. Dữ liệu trong Machine Learning

1.3.1. Làm việc với file .csv

Trong báo cáo này, các dữ liệu được sử dụng trong việc minh họa thuật toán đều ở dưới dạng file .csv. Vì vậy, báo cáo sẽ chỉ ra cách làm việc với những file có định dạng .csv như bước tiền xử lý dữ liệu trong áp dụng thuật toán Học máy.

A, Đọc file .csv với Standard Library.

Python cung cấp CSV mô-đun và trình đọc hàm reader () có thể được sử dụng để tải tệp CSV. Sau khi được tải, chúng ta có thể chuyển đổi dữ liệu CSV sang mảng NumPy và sử dụng nó để học máy.

Ví dụ,

```
# Load CSV Using Python Standard Library
import csv
import numpy
filename = 'Salary_Data.csv'
raw_data = open(filename, 'rb')
reader = csv.reader(raw_data, delimiter=',')
x = list(reader)
data = numpy.array(x).astype('float')
print(data.shape)
#YearsExperience,Salary
```

B, Đọc file .csv với Numpy

Có thể tải dữ liệu CSV của chúng ta bằng NumPy và hàm numpy.loadtxt (). Hàm này giả định không có hàng tiêu đề và tất cả dữ liệu có cùng một định dạng.

Ví dụ,

```
# Load CSV using NumPy
from numpy import loadtxt
filename = 'Salary_Data.csv'
raw_data = open(filename, 'rb')
data = loadtxt(raw_data, delimiter=",")
print(data.shape)
```

```
(30, 2)
```

C, Đọc file .csv với Pandas

Có thể tải dữ liệu CSV bằng cách sử dụng Pandas và hàm pandas.read csv (). Chức năng này rất linh hoạt và có lẽ là cách tiếp cận được khuyến nghị trong bài báo cáo này để tải dữ liệu máy học. Hàm trả về một pandas.DataFrame mà bạn có thể bắt đầu tóm tắt và vẽ biểu đồ ngay lập tức.

Ví dụ,

```
# Load CSV Using Pandas
from pandas import read_csv
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
data = read_csv(filename, names = names)
print(data.shape)
```

```
(30, 2)
```

Ngoài ra, người dùng cũng có thể sửa đổi để tải dữ liệu CSV trực tiếp từ một URL.

Ví dụ,

```
# Load CSV using Pandas from URL
from pandas import read_csv
url = 'https://data.api.trade.gov.uk/v1/datasets/uk-trade-quotas/
names = ['column_a', 'column_b', 'column_c']
data = read_csv(url, names=names)
print(data.shape)
```

```
(2833, 3)
```

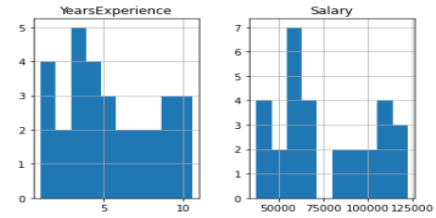
1.3.2. Hình dung dữ liệu

Người dùng phải hiểu dữ liệu để có được kết quả tốt nhất từ các thuật toán máy học. Cách nhanh nhất để tìm hiểu thêm về dữ liệu chính là sử dụng trực quan hóa dữ liệu. Trong phần này, báo cáo sẽ chỉ ra cách chúng có thể hình dung dữ liệu học máy bằng Python bằng Pandas.

- Univariate Plot:
  - ˆ Histograms.
  - ˆ Density Plots.
  - ˆ Box and Whisker Plots
- Multivariate Plots:
  - ˆ Correlation Matrix Plot.
  - ˆ Scatter Plot Matrix.

Ví dụ Histograms,

```
# Univariate Histograms
from matplotlib import pyplot
from pandas import read_csv
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
data = read_csv(filename, names=names)
data.hist()
pyplot.show()
```



1.3.3.Chuyển đổi dữ liệu

Chúng ta hầu như luôn cần xử lý trước dữ liệu và đây là một bước bắt buộc.Một khó khăn là các thuật toán khác nhau đưa ra các giả định khác nhau về dữ liệu và có thể yêu cầu các phép chuyển đổi. Hơn nữa, khi chúng tuân theo tất cả các quy tắc và đã chuẩn bị dữ liệu, đôi khi các thuật toán có thể mang lại kết quả tốt hơn mà không cần xử lý trước. Nói chung, trong báo cáo này người dùng được khuyên nên tạo nhiều chế độ xem và chuyển đổi dữ liệu khác nhau, sau đó thực hiện một số thuật toán trên mỗi chế độ xem của tập dữ liệu.

Trong bài báo cáo này, chúng ta sẽ làm việc với 3 công thức xử lý pre-processing khác nhau cho máy học.

- Rescale Data
- Standardize Data
- Normalize Data

A, Rescale Data

Khi tập dữ liệu bao gồm các thuộc tính với các tỷ lệ khác nhau, nhiều thuật toán học máy có thể cho ra kết quả tốt hơn từ việc thay đổi tỷ lệ các thuộc tính để tất cả các thuộc tính có cùng tỷ lệ, thường thuộc phạm vi từ 0 đến 1. Đây là việc hữu ích cho các thuật toán tối ưu hóa được sử dụng các thuật toán học máy như gradient descent. Chúng ta có thể định lại dữ liệu của mình sử dụng scikit-learning bằng MinMaxScaler class.

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:1]
Y = array[:,1]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Ví dụ,

```
[[0.   ]
 [0.021]
 [0.043]
 [0.096]
 [0.117]]
```

B, Standardize Data

Standardization là một kỹ thuật hữu ích để biến đổi các thuộc tính với phân phối Gaussian với giá trị trung bình và độ lệch chuẩn khác nhau so với phân phối Gaussian chuẩn với giá trị trung bình là 0 và độ lệch chuẩn là 1. Nó phù hợp nhất cho các kỹ thuật giả định một phân phối Gaussian trong các biến đầu vào và hoạt động tốt hơn với dữ liệu được thay đổi tỷ lệ, chẳng hạn như hồi quy tuyến tính, hồi quy logistic. Bạn có thể chuẩn hóa dữ liệu bằng cách sử dụng scikit-learning với class StandardScaler.

$$\textbf{standardized value}_i = (\textbf{value}_i - \textbf{mean}) / \textbf{stdev}$$

$$\textbf{stdev} = (P_{(i=1,n)} (\textbf{value}_i - \textbf{mean})^2 / (\textbf{count(values)} - 1))^{1/2}$$

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:1]
Y = array[:,1]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

Ví dụ,

```
[[-1.51 ]
 [-1.438]
 [-1.367]
 [-1.187]
 [-1.116]]
```

C, Normalize Data

Normalizing trong scikit-learning đề cập đến việc thay đổi tỷ lệ mỗi quan sát (hàng) để có độ dài là 1 đơn vị. Phương pháp tiền xử lý này có thể hữu ích cho các tập dữ liệu thưa thớt (nhiều số 0) với các thuộc tính của các tỷ lệ khác nhau khi sử dụng các thuật toán coi trọng các giá trị đầu vào chẳng hạn như mạng nơ-ron và các

thuật toán sử dụng khoảng cách các biện pháp như k-Nearest Neighbors. Chúng ta có thể chuẩn hóa dữ liệu bằng Python với scikit-learning sử dụng class Normalizer.

Scaled value = (value – min) / (max – min)

Ví dụ,

```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:1]
Y = array[:,1]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

CHƯƠNG 2. MACHINE LEARNING ALGORITHMS

2.1. Đánh giá hiệu năng của thuật toán

Đánh giá là một ước tính mà có thể sử dụng để hình dung về thuật toán thực sự có thể làm trong thực tế. Nó không phải là một đảm bảo về hiệu suất. Sau khi ước tính hiệu suất của thuật toán, chúng ta có thể đào tạo lại thuật toán cuối cùng trong toàn bộ quá trình đào tạo tập dữ liệu và chuẩn bị sẵn sàng để sử dụng trong hoạt động. Báo cáo này sẽ xem xét hai kỹ thuật mà có thể sử dụng để chia nhỏ tập dữ liệu đào tạo và tạo các ước tính hữu ích về hiệu suất cho các thuật toán học máy:

- Train and Test Sets.
- k-fold Cross Validation

A, Train and Test Sets.

Phương pháp đơn giản nhất mà chúng ta có thể sử dụng để đánh giá hiệu suất của thuật toán học máy là sử dụng các bộ dữ liệu đào tạo và thử nghiệm khác nhau. Chúng ta có thể lấy tập dữ liệu gốc và chia nó thành hai phần. Đào tạo thuật toán ở phần đầu tiên, đưa ra dự đoán trong phần thứ hai và đánh giá các dự đoán so với kết quả mong đợi. Kích thước của sự phân tách có thể phụ thuộc vào kích thước và chi tiết cụ thể của tập dữ liệu ban đầu, kỹ thuật đánh giá thuật toán này rất nhanh, phù

hợp cho các tập dữ liệu lớn (hàng triệu bản ghi). Nhược điểm của kỹ thuật này là nó có thể có một phương sai cao, do có sự khác biệt trong tập dữ liệu đào tạo và kiểm tra từ đó dẫn đến sự khác biệt lớn trong ước tính độ chính xác.

Ví dụ,

```
# Evaluate using a train and a test set
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:1]
Y = array[:,1]
test_size = 0.2
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,random_state=seed)
model = LinearRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print(("Accuracy: %.3f%%") % (result*100.0))
```

```
Accuracy: 81.430%
```

B, k-fold Cross Validation

Xác thực chéo (Cross Validation) là một cách tiếp cận mà bạn có thể sử dụng để ước tính hiệu suất của thuật toán học máy với ít phương sai hơn so với một phân tách tập hợp kiểm tra huấn luyện duy nhất. Nó hoạt động bằng cách chia dữ liệu thành các folds(nếp gấp) và đảm bảo rằng mỗi fold được sử dụng như một bộ thử nghiệm tại một số thời điểm. Trong lần lặp đầu tiên, fold đầu tiên được sử dụng để kiểm tra mô hình và phần còn lại được sử dụng để huấn luyện mô hình. Quá trình này được lặp lại cho đến khi mỗi fold trong số k folds đã được sử dụng làm bộ thử nghiệm. Sau khi chạy xác thực chéo, chúng sẽ nhận được k điểm hiệu suất khác nhau mà có thể tóm tắt lại bằng cách sử dụng giá trị trung bình và độ lệch chuẩn. Kết quả là một ước tính đáng tin cậy hơn về hiệu suất của thuật toán trên dữ liệu mới vì thuật toán được huấn luyện và đánh giá nhiều lần trên các dữ liệu khác nhau.

Ví dụ,

```
# Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = 'Salary_Data.csv'
names = ['YearsExperience','Salary']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:1]
Y = array[:,1]
num_folds = 3
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed,shuffle = True)
model = LinearRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print(("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 95.243% (1.887%)
```

2.2. Các chỉ số đánh giá thuật toán học máy

Chúng ta phải ước tính chất lượng của một tập hợp các dự đoán khi đào tạo một mô hình học máy. Các chỉ số hiệu suất có thể cung cấp một ý tưởng khách quan rõ ràng về mức độ phù hợp của một tập hợp các dự đoán và mô hình dự đoán học máy. Trong báo cáo này chúng ta đề cập tới chỉ số phân loại(classification accuracy, confusion matrix) và chỉ số hồi quy(mean absolute error, mean squared error).

A, Classification Accuracy

Độ chính xác của phân loại(classification accuracy)là tỷ lệ giữa số lượng các dự đoán đúng trong số tất cả các dự đoán đã được đưa ra.

accuracy = (correct predictions / total predictions) *100

```
# Calculate accuracy percentage between two lists
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

Ví dụ,

```
# Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'Social_Network_Ads.csv'
names = ['Age','EstimatedSalary','Purchased']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:2]
Y = array[:,2]
num_folds = 5
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed,shuffle = True)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold,scoring='accuracy')
print(("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0))
```

```
Accuracy: 64.250% (5.099%)
```

B, Confusion Matrix

Ma trận nhầm lẫn cung cấp một bản tóm tắt của tất cả các dự đoán được thực hiện so với các giá trị thực tế dự kiến. Kết quả được trình bày dưới dạng ma trận với số đếm trong mỗi ô. Số lượng giá trị dự đoán được tóm tắt theo chiều ngang (hàng), trong khi số lượng giá trị đúng cho mỗi giá trị được trình bày theo chiều dọc (cột). Một tập hợp các dự đoán chính xác được hiển thị dưới dạng một đường chéo từ trên cùng bên trái đến dưới cùng bên phải của ma trận.

```
# calculate a confusion matrix
def confusion_matrix(actual, predicted):
    unique = set(actual)
    matrix = [list() for x in range(len(unique))]
    for i in range(len(unique)):
        matrix[i] = [0 for x in range(len(unique))]
    lookup = dict()
    for i, value in enumerate(unique):
        lookup[value] = i
    for i in range(len(actual)):
        x = lookup[actual[i]]
        y = lookup[predicted[i]]
        matrix[y][x] += 1
    return unique, matrix
```

```
# Cross Validation Classification Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
filename = 'Social_Network_Ads.csv'
names = ['Age','EstimatedSalary','Purchased']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:2]
Y = array[:,2]
test_size = 0.3
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                test_size=test_size,random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

```
[[72 11]
 [ 4 33]]
```

Ví dụ,

C, Mean Absolute Error

Trong lớp bài toán regression (đầu ra là 1 số thực), số liệu được xem xét là sai số trong các giá trị được dự đoán so với các giá trị mong đợi. Sai số tuyệt đối trung bình hoặc MAE , nó được tính là giá trị trung bình của các giá trị sai số tuyệt đối, trong đó giá trị tuyệt đối có giá trị dương để chúng có thể được cộng lại với nhau.

$$\mathbf{MAE = (P_{i=1}\, abs(predicted_i - actual_i)) / total\ predictions}$$

```
# Calculate mean absolute error
def mae_metric(actual, predicted):
    sum_error = 0.0
    for i in range(len(actual)):
        sum_error += abs(predicted[i] - actual[i])
    return sum_error / float(len(actual))
```

D, Root Mean Squared Error

RMSE được tính bằng căn bậc hai của trung bình sai số bình phương giữa kết quả thực tế và dự đoán. Bình phương mỗi sai số buộc các giá trị phải dương.

$$\mathbf{RMSE = (P_{i=1}(predicted_i - actual_i)^2 / total\ predictions)^{1/2}}$$

```python
# Calculate root mean squared error
def rmse_metric(actual, predicted):
    sum_error = 0.0
    for i in range(len(actual)):
        prediction_error = predicted[i] - actual[i]
        sum_error += (prediction_error ** 2)
    mean_error = sum_error / float(len(actual))
    return sqrt(mean_error)
```

2.3. Linear Regression

Hồi quy tuyến tính giả định mối quan hệ tuyến tính hoặc đường thẳng giữa các biến đầu vào (X) và biến đầu ra duy nhất (y). Cụ thể hơn, đầu ra (y) đó có thể được tính toán từ sự kết hợp tuyến tính của các biến đầu vào (X). Khi có một biến đầu vào duy nhất, phương pháp này được gọi là hồi quy tuyến tính đơn giản. Trong hồi quy tuyến tính đơn giản, chúng ta có thể sử dụng thống kê trên dữ liệu huấn luyện để ước tính các hệ số theo yêu cầu của mô hình để đưa ra dự đoán trên dữ liệu mới. Mô hình hồi quy tuyến tính đơn giản có thể được viết là: **y = b0 + b1 × x,** với:

$$B1 = \frac{\sum_{i=1}^{n}((x_i - mean(x)) \times (y_i - mean(y)))}{\sum_{i=1}^{n}(x_i - mean(x))^2}$$
$$B0 = mean(y) - B1 \times mean(x)$$

Bên cạnh đó chúng ta có công thức tính giá trị trung bình là

$$mean(x) = \frac{\sum_{i=1} x_i}{count(x)}$$

Và Phương sai là tổng bình phương của mỗi giá trị từ giá trị trung bình, nên:

$$variance = \sum_{i=1}^{n}(x_i - mean(x))^2$$

Hiệp phương sai của hai nhóm số mô tả cách các số đó thay đổi cùng nhau. Hiệp phương sai là một tổng quát của mối tương quan mô tả mối quan hệ giữa hai nhóm số hoặc là nhiều nhóm, và:

$$covariance = \sum_{i=1}^{n}((x_i - mean(x)) \times (y_i - mean(y)))$$

Từ đây ta có được:

b0 = mean(y) – b1 * mean(x)

b1 = covariance(x,y) / variance(x)

Lúc này, giá trị đầu ra dự đoán chính là **yhat = b0 + b1 * x_test**

Với x_test là giá trị đầu vào dùng để kiểm tra sau khi tập dữ liệu ban đầu được chia thành tập train và tập test.

⇨ Với bài toán này, chúng ta sử dụng tập dữ liệu có sẵn: "Salary_Data.csv" và đào tạo dữ liệu trên tập dữ liệu đào tạo sau khi chia tập dữ liệu thành hai phần là train set và test set để xác định các hệ số cho bài toán tuyến tính đang xét và xác định giá trị đầu ra của tập test set thông qua các hệ số này. Tiếp theo chúng ta đào tạo dữ liệu và đưa ra hệ số của mô hình. Cuối cùng, chúng ta tính toán chỉ số RMSE cho mô hình đào tạo simple_linear_regression được đào tạo trên tập train set.

```
from random import randrange
from csv import reader
from math import sqrt
# load and prepare data
filename = 'Salary_Data.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])):
    str_column_to_float(dataset, i)
# evaluate algorithm
split = 1/3
rmse = evaluate_algorithm(dataset,
 simple_linear_regression, split)
print('RMSE: %.7f' % (rmse))
```

```
RMSE: 7065.8698595
```

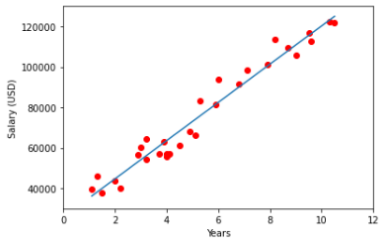
Kết quả minh họa:

Chúng ta có sai số cho các dự đoán kết quả của tập dữ liệu kiểm tra hay điểm số RMSE là khoảng 5300 ($ USD) chấp nhận được. Và chúng ta thu được phương trình của mô hình hồi quy tuyến tính đơn giản là:

Salary = 9449.962321455077 * YearsExperience + 25792.20019866869

Tiếp theo chúng ta vẽ đồ thị biểu diễn phương trình trên và tập dữ liệu:

```
x0 = np.array(x0)
y0 = np.array(y0)

# Drawing the fitting line
plt.plot(X, y, 'ro')     # data
plt.plot(x0, y0)               # the fitting line
plt.axis([0, 12, 30000, 130000])
plt.xlabel('Years' )
plt.ylabel('Salary (USD)')
plt.show()
```



Kết quả minh họa,

Từ đồ thị bên trên ta thấy rằng các điểm dữ liệu màu đỏ nằm khá gần đường thẳng dự đoán màu xanh. Vậy mô hình Linear Regression hoạt động tốt với tập dữ liệu cho trước.

2.4. Logistic

Hồi quy logistic sử dụng một phương trình làm đại diện, rất giống hồi quy tuyến tính. Giá trị đầu vào (X) được kết hợp tuyến tính bằng cách sử dụng trọng số hoặc giá trị hệ số để dự đoán giá trị đầu ra (y). Một điểm khác biệt chính so với hồi quy tuyến tính là giá trị đầu ra đang được mô hình hóa là giá trị nhị phân (0 hoặc 1) chứ không phải là giá trị số:

Ví dụ, $\mathbf{y = 1 / (1 + exp(- (b0 + b1 * x + b2 * x^2))}$

Để ước tính các giá trị hệ số cho dữ liệu đào tạo, chúng ta sử dụng phương pháp tối ưu Stochastic gradient descent, với hai tham số đi theo gồm:

- Learning rate: Được sử dụng để giới hạn số lượng mà mỗi hệ số được điều chỉnh cho mỗi lần nó được cập nhật.
- Epoch: Số lần chạy dữ liệu đào tạo trong khi cập nhật các hệ số.

Để tính các hệ số, chúng ta xây dựng 1 hàm gồm ba vòng lặp:

- Lặp cho mỗi epoch
- Lặp từng hàng trong dữ liệu đào tạo trong một epoch
- Lặp lại từng hệ số và cập nhật nó cho mỗi một hàng trong một epoch

Qua mỗi lần lặp, các hệ số (b) trong ngôn ngữ học máy được cập nhật bằng cách sử dụng phương trình: **b = b − learning rate × error × x**

Với x là giá trị đầu vào, error = prediction – expected : sai khác giữa giá trị dự đoán được thực hiện với các hệ số ứng viên và giá trị đầu ra dự kiến.

Hay tương đương với : **b = b + learning rate × (y − yhat) × yhat × (1 − yhat) × x**

```
# Estimate logistic regression coefficients using stochastic gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            yhat = predict(row, coef)
            error = row[-1] - yhat
            sum_error += error**2
            coef[0] = coef[0] + l_rate * error * yhat * (1.0 - yhat)
            for i in range(len(row)-1):
                coef[i + 1] = coef[i + 1] + l_rate * error * yhat * (1.0 - yhat) * row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return coef
```

⇨ Với bài toán này chúng ta sử dụng tập dữ liệu: Social_Network_Ads.csv và chuyển dữ liệu mỗi cột về phạm vi 0-1 do có sự chênh lệch lớn giữa những dữ liệu của mỗi cột. Chúng ta chia tập dữ liệu thành tập train và tập test bằng cách sử dụng xác thực chéo 5-fold và sử dụng phương pháp tối ưu Stochastic gradient descent ở trên để tối ưu việc tính toán hệ số cho mô hình logistic này. Tiếp theo chúng ta đào tạo dữ liệu và tính toán các hệ số của mô hình. Cuối cùng, chúng ta tính chỉ số Classification accuracy cho mỗi fold rồi tính độ chính xác trung bình cho cả bài toán.

```
from random import seed
seed(1)
# load and prepare data
filename='Social_Network_Ads.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])):
    str_column_to_float(dataset[1:], i)
# normalize
minmax = dataset_minmax(dataset[1:])
normalize_dataset(dataset[1:], minmax)c
# evaluate algorithm
n_folds = 5
l_rate = 0.1
n_epoch = 100
scores,coef = evaluate_algorithm(dataset[1:],
        logistic_regression, n_folds, l_rate, n_epoch)
print(coef)
print('Scores: %s' % scores)
print(f'Mean Accuracy:{sum(scores)/float(len(scores))} %')
```

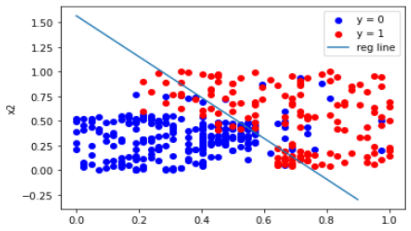
```
[-5.626183268207343, 7.206907836854775, 3.2802898478090725]
Scores: [80.0, 85.0, 86.25, 80.0, 82.5]
Mean Accuracy:82.75 %
```

Kết quả minh họa,

Chúng ta có độ chính xác cho các dự đoán trên các tập dữ liệu kiểm tra trong k-folds và giá trị trung bình của nó là 82.75% là khá cao. Vì vậy mô hình logistic regression hoạt động tốt trên tập dữ liệu cho trên.

Tiếp theo chúng ta vẽ đồ thị biểu diễn tập dữ liệu và minh họa hàm sigmoid chính là đường thẳng màu xanh lục như là ranh giới phân chia 2 class y=0 và y=1

```
# labelled observations
x_0 = X[np.where(y == 0)]
x_1 = X[np.where(y == 1)]
    # plotting points with diff color for diff label
plt.scatter([x_0[:, 0]], [x_0[:, 1]], c='b', label='y = 0')
plt.scatter([x_1[:, 0]], [x_1[:, 1]], c='r', label='y = 1')
    # plotting decision boundary
x1 = np.arange(0, 1, 0.1)
x2 = -(coef[0] + coef[1]*x1)/coef[2]
plt.plot(x1, x2, label='reg line')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()
```



Kết quả minh họa,

2.5. K nearest neighbors

Với thuật toán k-Nearest Neighbors hay viết tắt là KNN, toàn bộ tập dữ liệu đào tạo được lưu trữ. Khi một dự đoán được yêu cầu, các bản ghi k-giống nhất với một bản ghi mới từ tập dữ liệu huấn luyện sẽ được lựa chọn. Từ những người hàng xóm này, một dự đoán tóm tắt được đưa ra. Một khi các hàng xóm được phát hiện, dự đoán tóm tắt có thể được thực hiện bằng cách trả về kết quả phổ biến nhất hoặc lấy giá trị trung bình.

Để xác định k phiên bản nào trong tập dữ liệu huấn luyện gần giống nhất với đầu vào mới, một thước đo khoảng cách được sử dụng. Đối với các biến đầu vào có giá trị thực, thước đo khoảng cách phổ biến nhất là khoảng cách Euclide. Khoảng cách Euclide được tính bằng căn bậc hai của tổng bình phương chênh lệch giữa điểm a và điểm b trên tất cả các thuộc tính đầu vào i.

$$EuclideanDistance(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

```
import math
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1)-1):
        distance += (row1[i] - row2[i])**2
    return math.sqrt(distance)
```

Tiếp theo, các vùng lân cận của một phần dữ liệu mới trong tập dữ liệu là k-trường hợp gần nhất, được xác định bởi thước đo khoảng cách. Để xác định vị trí láng giềng của một phần dữ liệu mới trong tập dữ liệu, trước tiên chúng ta phải tính toán khoảng cách giữa mỗi bản ghi trong tập dữ liệu đến phần dữ liệu mới. Chúng ta có thể làm điều này bằng cách sử dụng hàm khoảng cách ở trên. Khi khoảng cách được tính toán, chúng ta phải sắp xếp tất cả các bản ghi trong tập dữ liệu huấn luyện theo khoảng cách của chúng với dữ liệu mới. Sau đó, chúng ta có thể chọn k hàng đầu để trả về là các hàng xóm gần giống nhất. Chúng ta có thể làm điều này bằng cách theo dõi khoảng cách cho mỗi bản ghi trong tập dữ liệu dưới dạng một bộ dữ liệu, sắp xếp danh sách các bộ dữ liệu theo khoảng cách (theo thứ tự giảm dần) và sau đó truy xuất các vùng lân cận. Dưới đây là một hàm có tên là get Neighbor () thực hiện điều này:

```
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors
```

Sau đó, những người hàng xóm giống nhau nhất được thu thập từ tập dữ liệu huấn luyện có thể được sử dụng để đưa ra dự đoán. Trong trường hợp phân loại, chúng ta có thể trả về class đại diện nhất trong số các lớp lân cận. Chúng ta có thể đạt được điều này bằng cách thực hiện hàm max () trên danh sách các giá trị đầu ra từ các vùng lân cận. Đưa ra một danh sách các giá trị class được quan sát trong các lớp lân cận, hàm max () nhận một tập hợp các giá trị class duy nhất và gọi tổng số trên danh sách các giá trị class này cho mỗi giá trị class trong tập hợp. Dưới đây là hàm có tên là predict classification () thực hiện điều này

```
# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
    neighbors = get_neighbors(train, test_row, num_neighbors)
    output_values = [row[-1] for row in neighbors]
    prediction = max(set(output_values), key=output_values.count)
    return prediction
```

Cuối cùng, chúng ta có hàm dự đoán đầu ra của thuật toán KNN là:

```
# kNN Algorithm
def k_nearest_neighbors(train, test, num_neighbors):
    predictions = list()
    for row in test:
        output = predict_classification(train, row, num_neighbors)
        predictions.append(output)
    return(predictions)
```

⇨ Chúng ta tiếp tục sử dụng tập dữ liệu Social_Network_Ads.csv cho bài toán này. Tương tự giải thuật Logistic ở trên, trước khi đào tạo dữ liệu, chúng ta đưa dữ liệu về dạng Normalize trong phạm vi 0-1 do có sự chênh lệch lớn giữa những dữ liệu của mỗi cột và chia tập dữ liệu thành tập train và tập test bằng cách sử dụng xác thực chéo k-fold. Chúng ta đào tạo dữ liệu và đưa ra những dự đoán đầu ra trên tập dữ liệu kiểm tra. Cuối cùng, chúng ta tính chỉ số Classification accuracy cho mỗi fold rồi tính độ chính xác trung bình cho cả bài toán.

```
n_folds = 5
num_neighbors = 5
scores = evaluate_algorithm(dataset[1:], k_nearest_neighbors,
                            n_folds, num_neighbors)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [76.25, 82.5, 78.75, 81.25, 77.5]
Mean Accuracy: 79.250%
```

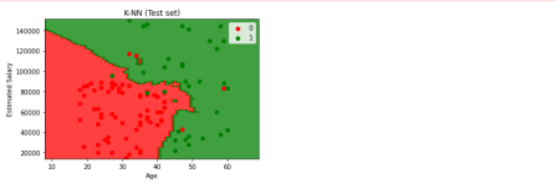
Kết quả minh họa,

Chúng ta có độ chính xác cho các dự đoán trên các tập dữ liệu kiểm tra trong k-folds và giá trị trung bình của nó gần 80%. Vì vậy mô hình K nearest neighbor hoạt động khá tốt trên tập dữ liệu trên.

Tiếp theo chúng ta vẽ đồ thị biểu diễn tập dữ liệu kiểm tra và phân vùng của 2 lớp class y= 0 và y=1

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 1),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 1))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a sin
intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a sin
intend to specify the same RGB or RGBA value for all points.
```



Kết quả minh họa,

2.6. Nayes Bayes

Định lý Bayes cung cấp một cách mà chúng ta có thể tính toán xác suất của một phần dữ liệu thuộc một lớp nhất định. Định lý Bayes được phát biểu là:

$$P(class|data) = \frac{P(data|class) \times P(class)}{P(data)}$$

Trong đó P(class|data) là xác suất của lớp cho dữ liệu được cung cấp. Naive Bayes là một thuật toán phân loại cho các bài toán phân loại nhị phân (hai lớp) và đa lớp. Nó được gọi là Naive Bayes vì các phép tính xác suất cho mỗi lớp được đơn giản hóa để làm cho các phép tính của chúng có thể kiểm soát được.

Để tính xác suất dữ liệu theo lớp mà chúng thuộc về, trước tiên chúng ta sẽ cần tách dữ liệu đào tạo của mình theo lớp. Chúng ta có thể tạo một đối tượng từ điển trong đó mỗi khóa là giá trị lớp và sau đó thêm danh sách tất cả các bản ghi làm giá trị trong từ điển. Dưới đây là một hàm được đặt tên riêng biệt theo lớp () thực hiện phương pháp này. Nó giả định rằng cột cuối cùng trong mỗi hàng trong tập dữ liệu là giá trị lớp.

```
# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
    separated = dict()
    for i in range(len(dataset)):
        vector = dataset[i]
        class_value = vector[-1]
        if (class_value not in separated):
            separated[class_value] = list()
        separated[class_value].append(vector)
    return separated
```

Tiếp theo, chúng ta cần hai thống kê từ một tập hợp dữ liệu nhất định, đó là giá trị trung bình và độ lệch chuẩn (độ lệch trung bình so với giá trị trung bình), với độ lệch chuẩn mẫu là chênh lệch trung bình so với giá trị trung bình.

$$mean = \frac{\sum_{i=1}^{n} x_i}{count(x)} \quad , \quad \text{standard deviation} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - mean(x))^2}{count(x) - 1}}$$

Sau khi tính toán,chúng ta có thể tập hợp các số liệu thống kê lại với nhau thành một danh sách hoặc nhiều số liệu thống kê. Sau đó, lặp lại thao tác này cho

mỗi cột trong tập dữ liệu và trả về một danh sách các bộ thống kê, hàm sử dụng Summary dataset

```
# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
    summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
    del(summaries[-1])
    return summaries
```

Tiếp theo, tóm tắt các cột trong tập dữ liệu được tổ chức theo giá trị lớp. Dưới đây là một hàm có tên là Summary by class () để thực hiện thao tác này. Đầu tiên, tập dữ liệu được chia theo lớp, sau đó thống kê được tính toán trên từng tập con. Các kết quả dưới dạng một danh sách các bộ thống kê sau đó được lưu trữ trong từ điển bởi giá trị lớp của chúng.

```
# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = dict()
    for class_value, rows in separated.items():
        summaries[class_value] = summarize_dataset(rows)
    return summaries
```

Bên cạnh đó, việc tính toán xác suất hoặc khả năng quan sát được một giá trị thực nhất định là khó nhưng với Naive Bayes có thể được mở rộng cho các thuộc tính có giá trị thực, nhờ việc sử dụng phân phối Gauss bằng việc ước tính giá trị trung bình và độ lệch chuẩn từ dữ liệu huấn luyện. Do đó, chúng ta có thể ước tính xác suất của một giá trị bởi hàm phân phối xác suất Gauss:

$$probability(x) = \frac{1}{\sqrt{2 \times PI} \times \text{standard\_deviation}} \times e^{-(\frac{(x-mean(x))^2}{2 \times \text{standard\_deviation}^2})}$$

```
# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
    exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
    return (1 / (sqrt(2 * pi) * stdev)) * exponent
```

Bây giờ chúng ta sử dụng thống kê được tính toán từ dữ liệu đào tạo để tính toán xác suất cho dữ liệu mới. Xác suất được tính riêng cho từng lớp. Điều này có nghĩa là đầu tiên tính xác suất để một phần dữ liệu mới thuộc về lớp đầu tiên, sau đó tính xác suất mà nó thuộc về lớp thứ hai, và như vậy đối với tất cả các lớp. Xác suất

rằng một phần dữ liệu thuộc về một lớp được tính như sau:

$$P(class|data) = P(X|class) \times P(class)$$

Ở đây, phép chia đã được loại bỏ để đơn giản hóa việc tính toán, khác với định lý Bayes. Do đó kết quả không còn là xác suất của dữ liệu thuộc một lớp. Giá trị ở đây vẫn là cực đại, hay phép tính cho lớp dẫn đến giá trị lớn nhất được lấy làm dự đoán. Đây là cách triển khai đơn giản hóa phổ biến vì chúng ta thường quan tâm đến dự đoán lớp hơn là xác suất. Và đây là nguyên nhân chúng ta cần tách dữ liệu theo giá trị lớp.

Dưới đây là một hàm calculate class probabilities() liên kết xác suất của các giá trị thực và các số liệu thống kê từ hàm calculate_probability(), nó có một tập hợp các bản summaries đã chuẩn bị và một hàng mới là hai đối số đầu vào:

```
# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
    total_rows = sum([summaries[label][0][2] for label in summaries])
    probabilities = dict()
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
        for i in range(len(class_summaries)):
            mean, stdev, _ = class_summaries[i]
            probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
    return probabilities
```

Đầu tiên, tổng số dữ liệu đào tạo được tính từ số lượng dữ liệu được lưu trữ trong các bản summaries . Điều này được sử dụng để tính toán xác suất của một lớp nhất định hoặc P (lớp) dưới dạng tỷ lệ của các hàng với một lớp nhất định của tất cả các hàng trong dữ liệu huấn luyện. Tiếp theo, xác suất được tính cho mỗi giá trị đầu vào trong hàng bằng cách sử dụng hàm mật độ xác suất và thống kê Gaussian cho cột đó và của lớp đó. Xác suất được nhân tích lũy với nhau . Quá trình này được lặp lại cho mỗi lớp trong tập dữ liệu. Cuối cùng, một từ điển xác suất được trả về với một mục nhập cho mỗi lớp.

Cuối cùng, sử dụng hàm predict() để tính toán xác suất của một hàng mới thuộc mỗi lớp và chọn lớp có giá trị xác suất lớn nhất. Và hàm Naive_bayes() để học thống kê từ tập dữ liệu huấn luyện và sử dụng chúng để đưa ra dự đoán cho tập dữ liệu thử nghiệm thông qua predict().

```
# Predict the class for a given row
def predict(summaries, row):
    probabilities = calculate_class_probabilities(summaries, row)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label
```

```
# Naive Bayes Algorithm
def naive_bayes(train, test):
    summarize = summarize_by_class(train)
    predictions = list()
    for row in test:
        output = predict(summarize, row)
        predictions.append(output)
    return(predictions)
```

⇨ Với bài toán này, chúng ta sử dụng tập dữ liệu tương tự như bào toán Logistic. Vì vậy trước khi đào tạo mô hình, chúng ta chia tập dữ liệu thành tập train và tập test bằng cách sử dụng xác thực chéo k-fold. Sau đó đào tạo mô hình dựa trên tập dữ liệu đào tạo và đưa ra những dự đoán đầu ra trên tập dữ liệu kiểm tra. Cuối cùng, chúng ta tính chỉ số Classification accuracy cho mỗi fold rồi tính độ chính xác trung bình cho cả bài toán.

```
n_folds = 5
scores = evaluate_algorithm(dataset[1:], naive_bayes, n_folds)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [87.5, 86.25, 93.75, 86.25, 91.25]
Mean Accuracy: 89.000%
```

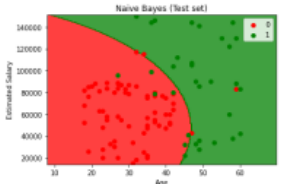
Kết quả minh họa,

Chúng ta có độ chính xác cho các dự đoán trên các tập dữ liệu kiểm tra trong k-folds và giá trị trung bình của nó gần 90%. Vì vậy mô hình Naive Bays hoạt động rất tốt trên tập dữ liệu trên.

Tiếp theo chúng ta vẽ đồ thị biểu diễn tập dữ liệu kiểm tra và phân vùng của 2 lớp class y= 0 và y=1:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()])).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Using Google Colab:



Kết quả minh họa: \

2.7. Decision tree

Cây phân loại và cây hồi quy hay viết tắt là CART, dùng để chỉ các thuật toán Cây quyết định có thể được sử dụng cho các bài toán mô hình dự báo phân loại hoặc hồi quy. Biểu diễn của mô hình CART là một cây nhị phân. Đây là cây nhị phân giống nhau từ thuật toán và cấu trúc dữ liệu (mỗi nút có thể có 0, một hoặc hai nút con). Một nút đại diện cho một biến đầu vào duy nhất (X) và một điểm phân tách trên biến đó,, các nút lá của cây chứa một biến đầu ra (y) được sử dụng để đưa ra dự đoán. Sau khi được tạo, một cây có thể được điều hướng với một hàng dữ liệu mới đi theo mỗi nhánh với các phần tách cho đến khi đưa ra dự đoán cuối cùng.

Tạo cây quyết định nhị phân thực chất là một quá trình phân chia không gian đầu vào. Một cách tiếp cận tham lam được sử dụng được gọi là tách nhị phân đệ quy. Đây là một thủ tục số trong đó tất cả các giá trị được sắp hàng và các điểm phân tách khác nhau được thử và kiểm tra bằng cách sử dụng một hàm chi phí. Việc phân chia với chi phí tốt nhất sẽ được chọn. Tất cả các biến đầu vào và tất cả các điểm phân tách có thể được đánh giá và lựa chọn một cách tham lam dựa trên hàm chi phí.

- Hồi quy: Hàm chi phí được tối thiểu hóa để chọn điểm phân tách là sai số tổng bình phương trên tất cả các mẫu đào tạo nằm trong hình chữ nhật.
- Phân loại: Hàm chi phí Gini được sử dụng để cung cấp chỉ báo về mức độ tinh khiết(purity) của các nút, trong đó độ tinh khiết(purity) của nút đề cập đến việc kết hợp dữ liệu đào tạo được chỉ định cho mỗi nút như thế nào.

Việc phân tách tiếp tục cho đến khi các nút chứa một số lượng mẫu đào tạo tối thiểu hoặc đạt đến độ sâu cây tối đa.

Chỉ số Gini là tên của hàm chi phí được sử dụng để đánh giá sự phân tách trong tập dữ liệu. Điểm Gini cho biết mức độ phân chia tốt như thế nào bằng cách trộn lẫn các lớp trong hai nhóm được tạo ra bởi sự phân chia. Một sự phân tách

hoàn hảo dẫn đến điểm Gini là 0, trong khi sự phân tách trong trường hợp xấu nhất dẫn đến 50/50 lớp trong mỗi nhóm dẫn đến điểm Gini là 0,5 (đối với bài toán 2 lớp).

Để tính toán Gini, đầu tiên chúng ta cần tính tỷ lệ các lớp trong mỗi nhóm.

$$proportion = \frac{count(class\_value)}{count(rows)}$$

Sau đó, Gini được tính cho mỗi nút con như sau:

$$\begin{aligned}\text{gini\_index} &= \sum_{i=1}(proportion_i \times (1.0 - proportion_i)) \\ &= 1 - \sum_{i=1}^{n} proportion_i^2\end{aligned}$$

Chỉ số Gini cho mỗi nhóm sau đó phải được tính theo quy mô của nhóm, liên quan đến tất cả các mẫu trong nhóm gốc, ví dụ: tất cả các mẫu hiện đang được nhóm. Chúng ta có thể thêm trọng số này vào phép tính Gini cho một nhóm như sau:

$$\text{gini\_index} = (1 - \sum_{i=1}^{n} proportion_i^2) \times \frac{group\_size}{total\_samples}$$

Điểm số sau đó được cộng trên mỗi nút con tại điểm phân tách để đưa ra điểm Gini cuối cùng cho điểm phân tách có thể được so sánh với các điểm phân tách ứng viên khác.

Dưới đây là hàm gini index() để tính toán chỉ số Gini cho một danh sách các nhóm và một danh sách các giá trị lớp đã biết. Bạn có thể thấy rằng có một số kiểm tra an toàn trong đó để tránh chia cho không cho một nhóm trống:

```
# Calculate the Gini index for a split dataset
def gini_index(groups, classes):
# count all samples at split point
    n_instances = float(sum([len(group) for group in groups]))
# sum weighted Gini index for each group
    gini = 0.0
    for group in groups:
        size = float(len(group))
        # avoid divide by zero
        if size == 0:
            continue
        score = 0.0
        # score the group based on the score for each class
        for class_val in classes:
            p = [row[-1] for row in group].count(class_val) / size
            score += p * p
        # weight the group score by its relative size
        gini += (1.0 - score) * (size / n_instances)
    return gini
```

Phần tách bao gồm một thuộc tính trong tập dữ liệu và một giá trị. Chúng ta có thể tóm tắt đây là chỉ số của một thuộc tính cần tách và giá trị để tách các hàng trên thuộc tính đó. Tạo một sự phân chia bao gồm ba phần, phần đầu tiên chúng ta đã xem xét đó là tính điểm Gini, tách tập dữ liệu và đánh giá tất cả các vết tách.

Với tách tập dữ liệu, tập dữ liệu tách thành hai danh sách các hàng được cung cấp chỉ mục của thuộc tính và giá trị tách cho thuộc tính đó. Sau đó chúng ta có thể sử dụng điểm Gini ở trên để đánh giá chi phí của việc phân tách. Tách tập dữ liệu bao gồm việc lặp lại từng hàng, kiểm tra xem giá trị thuộc tính có thấp hơn hay cao hơn giá trị phân tách hay không và gán nó cho nhóm bên trái hoặc bên phải tương ứng. Dưới đây là hàm test split () thực hiện thủ tục này:

```
# Split a dataset based on an attribute and an attribute value
def test_split(index, value, dataset):
    left, right = list(), list()
    for row in dataset:
        if row[index] < value:
            left.append(row)
        else:
            right.append(row)
    return left, right
```

Tiếp theo về đánh giá các vết tách, với một tập dữ liệu, chúng ta kiểm tra mọi giá trị trên mỗi thuộc tính dưới dạng một phần tách ứng viên, đánh giá chi phí của phần tách và tìm ra phần tách tốt nhất có thể. Sau khi tìm thấy phần tách tốt nhất, chúng ta có thể sử dụng nó như một nút trong cây quyết định. Sau đó sẽ lưu trữ chỉ mục của thuộc tính đã chọn, giá trị của thuộc tính đó để tách và hai nhóm dữ liệu được phân tách theo điểm phân tách đã chọn. Dưới đây là hàm get split () thực

hiện thủ tục này. Nó lặp lại từng thuộc tính (ngoại trừ giá trị lớp) và sau đó là từng giá trị cho thuộc tính đó, phân tách và đánh giá các phân tách. Sự phân chia tốt nhất được ghi lại và được trả lại sau khi tất cả các kiểm tra hoàn tất:

```
# Select the best split point for a dataset
def get_split(dataset):
    class_values = list(set(row[-1] for row in dataset))
    b_index, b_value, b_score, b_groups = 999, 999, 999, None
    for index in range(len(dataset[0])-1):
        for row in dataset:
            groups = test_split(index, row[index], dataset)
            gini = gini_index(groups, class_values)
            if gini < b_score:
                b_index, b_value, b_score, b_groups = index, row[index], gini, groups
    return {'index':b_index, 'value':b_value, 'groups':b_groups}
```

Việc xây dựng 1 cây được chia thành 3 phần:

- Các nút đầu cuối: Xác định thời điểm ngừng trồng cây bằng cách sử dụng

độ sâu và số lượng hàng tối thiểu mà nút chịu trách nhiệm. Khi chúng ta ngừng phát triển tại một điểm nhất định, nút đó được gọi là nút đầu cuối và được sử dụng để đưa ra dự đoán cuối cùng. Điều này được thực hiện bằng cách lấy nhóm hàng được gán cho nút đó và chọn giá trị lớp phổ biến nhất trong nhóm. Từ đó được sử dụng để đưa ra dự đoán. Dưới đây là hàm terminal () sẽ chọn một giá trị lớp cho một nhóm hàng và nó trả về giá trị đầu ra phổ biến nhất trong danh sách các hàng.

```
# Create a terminal node value
def to_terminal(group):
    outcomes = [row[-1] for row in group]
    return max(set(outcomes), key=outcomes.count)
```

- Tách đệ quy: Các nút mới được thêm vào một nút hiện có được gọi là các nút

con bằng việc gọi hàm get split () lặp đi lặp lại trên mỗi nhóm. Khi một nút được tạo, chúng ta có thể tạo các nút con một cách đệ quy trên từng nhóm dữ liệu từ phần tách bằng cách gọi lại cùng một hàm. Dưới đây là một hàm thực hiện thủ tục đệ quy này. Nó lấy một nút làm đối số cũng như độ sâu tối đa, số lượng mẫu tối thiểu trong một nút và độ sâu hiện tại của một nút.

```
# Create child splits for a node or make terminal
def split(node, max_depth, min_size, depth):
    left, right = node['groups']
    del(node['groups'])
    # check for a no split
    if not left or not right:
        node['left'] = node['right'] = to_terminal(left + right)
        return
    # check for max depth
    if depth >= max_depth:
        node['left'], node['right'] = to_terminal(left), to_terminal(right)
        return
    # process left child
    if len(left) <= min_size:
        node['left'] = to_terminal(left)
    else:
        node['left'] = get_split(left)
        split(node['left'], max_depth, min_size, depth+1)
    # process right child
    if len(right) <= min_size:
        node['right'] = to_terminal(right)
    else:
        node['right'] = get_split(right)
        split(node['right'], max_depth, min_size, depth+1)
```

- Xây dựng cây: Xây dựng cây liên quan đến việc tạo nút gốc và gọi hàm split

() và sau đó gọi lại chính nó một cách đệ quy để xây dựng toàn bộ cây. Dưới đây là hàm build tree () nhỏ thực hiện thủ tục này.

```
# Build a decision tree
def build_tree(train, max_depth, min_size):
    root = get_split(train)
    split(root, max_depth, min_size, 1)
    return root
```

Tiếp theo là dự đoán kết quả, việc đưa ra dự đoán với cây quyết định bao gồm việc điều hướng cây với hàng dữ liệu được cung cấp cụ thể và bằng cách sử dụng một hàm đệ quy, trong đó quy trình dự đoán tương tự được gọi lại với các nút con bên trái hoặc bên phải. Chúng ta phải kiểm tra xem một nút con có phải là một giá trị đầu cuối được trả về dưới dạng dự đoán hay không, hoặc nếu nó là một nút từ điển có chứa một cấp độ khác của cây cần được xem xét. Dưới đây là hàm dự đoán () thực hiện thủ tục này.

```
# Make a prediction with a decision tree
def predict(node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return predict(node['right'], row)
        else:
            return node['right']
```

⇨ Với bài toán này, chúng ta sử dụng tập dữ liệu tương tự như những bài toán Classification ở trên. Vì vậy trước khi đào tạo mô hình, chúng ta chia tập dữ liệu thành tập train và tập test bằng cách sử dụng xác thực chéo k-fold. Sau đó đào tạo mô hình dựa trên tập dữ liệu đào tạo và đưa ra những dự đoán đầu ra trên tập dữ liệu kiểm tra. Cuối cùng, chúng ta tính chỉ số Classification accuracy cho mỗi fold rồi tính độ chính xác trung bình cho cả bài toán.

```
# evaluate algorithm
n_folds = 5
max_depth = 5
min_size = 10
scores = evaluate_algorithm(dataset[1:], decision_tree, n_folds, max_depth, min_size)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

```
Scores: [90.0, 93.75, 88.75, 82.5, 91.25]
Mean Accuracy: 89.250%
```

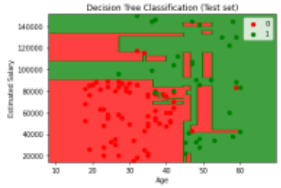
Kết quả minh họa,

Chúng ta có độ chính xác cho các dự đoán trên các tập dữ liệu kiểm tra trong k-folds và giá trị trung bình của nó gần 90%. Vì vậy mô hình Decision Tree hoạt động rất tốt trên tập dữ liệu trên.

Tiếp theo chúng ta vẽ đồ thị biểu diễn tập dữ liệu kiểm tra và phân vùng của 2 lớp class y= 0 và y=1:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will hav
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify
RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will hav
case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify
RGBA value for all points.
```



Kết quả minh họa,

KẾT LUẬN

Trong báo cáo này, chúng ta đã tìm hiểu sơ lược về những khía cạnh trong ngôn ngữ lập trình Python bao gồm một số cấu trúc dữ liệu, các thư viện quan trọng trong việc tính toán và biểu diễn hình ảnh trực quan và một số thuật toán học máy nổi bật về hồi quy (regression) gồm Linear regression, Logistic regression và phân loại (classification) bao gồm K nearest neughbor, Naive Bays, Decision Tree cùng với việc xử lý dữ liệu trước khi đào tạo mô hình học máy trên tập dữ liệu. Báo cáo đã thực hiện được yêu cầu của đề tài là minh họa lại một số thuật toán học máy bằng Python, nhưng bên cạnh đó trong báo cáo mới chỉ áp dụng thuật toán vào việc đào tạo mô hình trên tập dữ liệu đơn giản để đưa ra những dự đoán đầu ra đơn giản mà chưa có so sánh đánh giá hiệu suất của các thuật toán học máy với nhau, cũng như chưa đưa ra kết luận về những điều kiện trong việc áp dụng các thuật toán vào những bài toán cụ thể nào.

TÀI LIỆU THAM KHẢO

[1] " Python Programming Language: Step by Step Guide 2022 " [Trực tuyến]. Available: [Python Programming Language - A Step by Step Guide (hackr.io)]

[2] " Jupyter Notebook: An Introduction" [Trực tuyến]. Available : [Sổ ghi chép Jupyter: Giới thiệu - Python thực (realpython.com)]

[3] " Giới thiệu về Machine Learning" [Trực tuyến]. Available : [machinelearningcoban.com]

[4] "Các thuật toán Machine Learning cơ bản "[Trực tuyến]. Available : [machinelearningmastery.com]