VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaNangama", Belgaum -590014, Karnataka.

LAB REPORT on

Machine Learning

Submitted by

Aruna Ravi K R(1BM19CS225)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

BENGALURU-560019 May-**2022 to July-2022**

(Autonomous Institution under VTU)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning" carried out by **Aruna Ravi K R (1BM19CS225),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Asha G R
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE,Bengaluru

Index Sheet

Course Outcome

CO1	Ability to apply the different learning algorithms.
CO2	Ability to analyse the learning techniques for given dataset
CO3	Ability to design a model using machine learning to solve a problem.
CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning Techniques.

1) Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

a) Using CSV as input:

```
import csv

def updateHypothesis(x,h):
    if h==[]:
        return x

    for i in range(0,len(h)):
        if x[i].upper()!=h[i].upper():
            h[i] = '?'

    return h

if __name__ == "__main__":
    data = []
    h = []

    # reading csv file
    with open('Desktop/FindS.csv', 'r') as file:
        reader = csv.reader(file)
        print("Data: ")
        for row in reader:
            data.append(row)
            print(row)
    if data:
        for x in data:
            if x[-1].upper()=="YES":
                x.pop() # removing last field
                h = updateHypothesis(x,h)
    print("\nHypothesis: ",h)
```

Output:

```
Data:
['Time', 'Weather', 'Temperature', 'Company', 'Humidity', 'Wind', 'Goes']
['Morning', 'Sunny', 'Warm', 'Yes', 'Mild', 'Strong', 'Yes']
['Evening', 'Rainy', 'Cold', 'No', 'Mild', 'Normal', 'No']
['Morning', 'Sunny', 'Moderate', 'Yes', 'Normal', 'Normal', 'Yes']
['Evening', 'Sunny', 'Cold', 'Yes', 'High', 'Strong', 'Yes']

Hypothesis:  ['?', 'Sunny', '?', 'Yes', '?', '?']
```

B) Using user Input:

```
import numpy as np
import pandas as pd

n=int(input("Enter the number of attributes "))
l=int(input("Enter the number of rows "))

print("Enter the ",n,"ättributes")
attributes=[]
for i in range(1,n+1):
  print("Enter the name of ",i," attribute ")
  name=input()

for i in range(1,l+1):
  print("Ënter the values of ",i," row")
  print("Enter the values of attributes")
  res=[]
  for j in range(1,l+1):
    res.append(input())
  attributes.append(res)

print("Enter the target values")
target=[]
for i in range(1,l+1):
  print("Enter the value of ",i," target")
  x=input()
  target.append(x)

def findS(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis

print("\n The final hypothesis is:",findS(attributes,target))
```

Output:

```
Enter the  3 ättributes
Enter the name of  1  attribute

Enter the name of  2  attribute

Enter the name of  3  attribute

Ënter the values of  1  row
Enter the values of attributes

Ënter the values of  2  row
Enter the values of attributes

Ënter the values of  3  row
Enter the values of attributes

Enter the target values
Enter the value of  1  target

Enter the value of  2  target

Enter the value of  3  target

 The final hypothesis is: ['?', 'Rainy', 'Cold']
```

2) For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a descripon of the set of all hypotheses consistent with the training examples

```
import numpy as np
import pandas as pd

#to read the data in the csv file
data = pd.DataFrame(data=pd.read_csv('/content/drive/MyDrive/enjoysport.csv'))
print(data,"\n")

#making an array of all the attributes
concepts = np.array(data.iloc[:,0:-1])
print("The attributes are: ",concepts)

#segregating the target that has positive and negative examples
target = np.array(data.iloc[:,-1])
print("\n The target is: ",target)

#training function to implement candidate_elimination algorithm
def learn(concepts, target):
 specific_h = concepts[0].copy()
 print("\n Initialization of specific_h and general_h")
 print(specific_h)
 general_h = [["?" for i in range(len(specific_h))] for i in
range(len(specific_h))]
 print(general_h)
 for i, h in enumerate(concepts):
    if target[i] == "yes":
       for x in range(len(specific_h)):
          if h[x]!= specific_h[x]:
             specific_h[x] ='?'
             general_h[x][x] ='?'
          #  print(specific_h)
    if target[i] == "no":
       for x in range(len(specific_h)):
          if h[x]!= specific_h[x]:
             general_h[x][x] = specific_h[x]
          else:
             general_h[x][x] = '?'
    print("\n Steps of Candidate Elimination Algorithm",i+1)
    print(specific_h)
    print(general_h)
 indices = [i for i, val in enumerate(general_h) if val ==
['?', '?', '?', '?', '?', '?']]
 for i in indices:
    general_h.remove(['?', '?', '?', '?', '?', '?'])
 return specific_h, general_h
s_final, g_final = learn(concepts, target)

#obtaining the final hypothesis
print("\nFinal Specific_h:", s_final, sep="\n")
```

```
print("\nFinal General_h:", g_final, sep="\n")
```

Output:

```
     sky  temp humidity    wind water forcast enjoysport
0  sunny  warm   normal  strong  warm    same       yes
1  sunny  warm     high  strong  warm    same       yes
2  rainy  cold     high  strong  warm  change        no
3  sunny  warm     high  strong  cool  change       yes

The attributes are:  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

 The target is:  ['yes' 'yes' 'no' 'yes']

 Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?
'?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?
'?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?
'?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

 Steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3)Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

a)ID3 :

```
import math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"));
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic

def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
```

```
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
```

```
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:",end="    ")
    classify(node1,xtest,features)
```

Output:

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   rain
     Wind
       strong
         no
       weak
         yes
   overcast
     yes
   sunny
     Humidity
       normal
         yes
       high
         no
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:   no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:   no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:   yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:   yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:   no
```

b) Using SKlearn:

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_iris

data = load_iris()
```

In [2]:

```
df = pd.DataFrame(data.data, columns = data.feature_names)
```

In [3]:

```
df.head()
df['Species'] = data.target

#replace this with the actual names

target = np.unique(data.target)

target_names = np.unique(data.target_names)

targets = dict(zip(target, target_names))

df['Species'] = df['Species'].replace(targets)
```

In [5]:

```
x = df.drop(columns="Species")

y = df["Species"]
```

In [6]:

```
feature_names = x.columns
labels = y.unique()
```

In [7]:

```
from sklearn.model_selection import train_test_split

X_train, test_x, y_train, test_lab = train_test_split(x,y,test_size = 0.4,random_state = 42)
```

In [8]:

```
 from sklearn.tree import DecisionTreeClassifier
 clf = DecisionTreeClassifier(max_depth =4, random_state = 42)
```

In [9]:

```
clf.fit(X_train, y_train)
test_pred = clf.predict(test_x)
```

In [11]:

```
from sklearn import metrics

import seaborn as sns

import matplotlib.pyplot as plt
```

```
confusion_matrix = metrics.confusion_matrix(test_lab,test_pred)
```

In [12]:

```
confusion_matrix
```

```
matrix_df = pd.DataFrame(confusion_matrix)
ax = plt.axes()
sns.set(font_scale=1.3)
plt.figure(figsize=(10,7))
sns.heatmap(matrix_df, annot=True, fmt="g", ax=ax, cmap="magma")
ax.set_title('Confusion Matrix - Decision Tree')
ax.set_xlabel("Predicted label", fontsize =15)
ax.set_xticklabels(['']+labels)
ax.set_ylabel("True Label", fontsize=15)
ax.set_yticklabels(list(labels), rotation = 0)
plt.show()
```

```
clf.score(test_x,test_lab)
```

```
from sklearn import tree
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=data.feature_names,
                   class_names=data.target_names,
                   filled=True)
```

Output:

Out[3]:

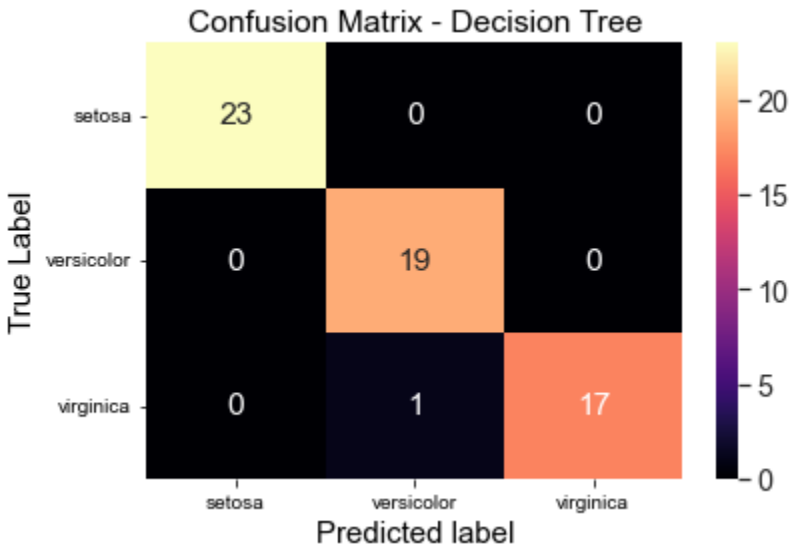
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Out[9]:

```
DecisionTreeClassifier(max_depth=4, random_state=42)
```

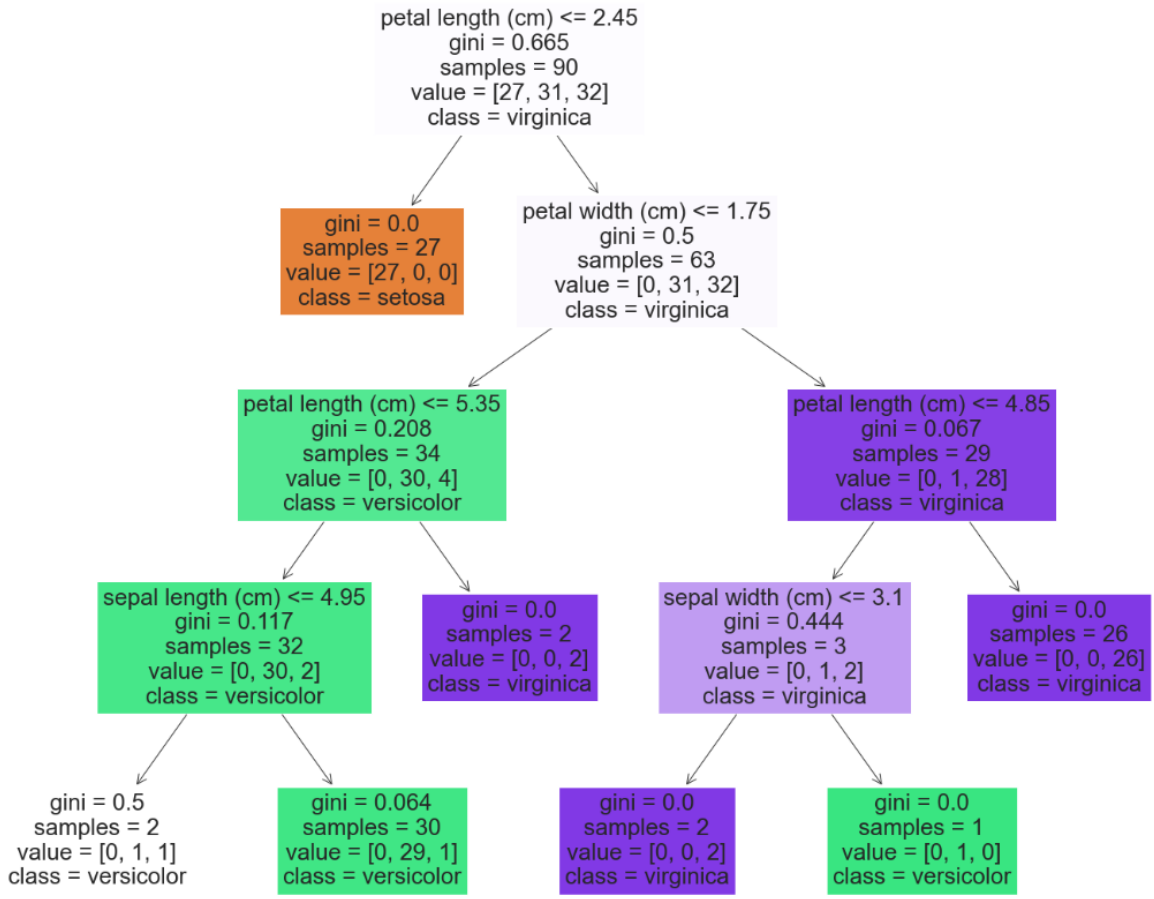
Out[12]:

```
array([[23,  0,  0],
       [ 0, 19,  0],
       [ 0,  1, 17]], dtype=int64)
```




```
In [14]: clf.score(test_x,test_lab)
```

```
Out[14]: 0.9833333333333333
```



4)Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

a) Without using SKlearn:

```
import numpy as np
import pandas as pd

data = pd.read_csv('/content/dataset.csv')
data.head()

y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values
print(f'Target Values: {y}')
print(f'Features: \n{X}')
y_train = y[:8]
y_val = y[8:]
X_train = X[:8]
X_val = X[8:]
print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
class NaiveBayesClassifier:
    def __init__(self, X, y):
        self.X, self.y = X, y
        self.N = len(self.X)
        self.dim = len(self.X[0])
        self.attrs = [[] for _ in range(self.dim)]
        self.output_dom = {}
        self.data = []
        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])
            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1
            else:
                self.output_dom[self.y[i]] += 1
            self.data.append([self.X[i], self.y[i]])
    def classify(self, entry):
        solve = None
        max_arg = -1
        for y in self.output_dom.keys():
            prob = self.output_dom[y]/self.N
            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N
            if prob > max_arg:
                max_arg = prob
                solve = y
```

```
        return solve
nbc = NaiveBayesClassifier(X_train, y_train)
total_cases = len(y_val)
good = 0
bad = 0
predictions = []
for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)
    if y_val[i] == predict:
        good += 1
    else:
        bad += 1
print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

Output:

Out[2]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

```
Number of instances in training set: 8
Number of instances in testing set: 6

Predicted values: ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
Actual values: ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 6
Number of correct predictions: 4
Number of wrong predictions: 2

Accuracy of Bayes Classifier: 0.6666666666666666
```

b)Using SKlearn:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("/content/pima_indian.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']
X = df[feature_col_names].values
y = df[predicted_class_names].values
print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.33)
print ('\nThe total number of Training Data:',ytrain.shape)
print ('The total number of Test Data:',ytest.shape)
clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])
print('\nConfusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\nAccuracy of the classifier:',metrics.accuracy_score(ytest,predicted))
print('The value of Precision:', metrics.precision_score(ytest,predicted))
print('The value of Recall:', metrics.recall_score(ytest,predicted))
print("Predicted Value for individual Test Data:", predictTestData)
```

Output:

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  ...  diab_pred  age  diabetes
0           6           148            72  ...      0.627   50         1
1           1            85            66  ...      0.351   31         0
2           8           183            64  ...      0.672   32         1
3           1            89            66  ...      0.167   21         0
4           0           137            40  ...      2.288   33         1
..        ...           ...           ...  ...        ...  ...       ...
763        10           101            76  ...      0.171   63         0
764         2           122            70  ...      0.340   27         0
765         5           121            72  ...      0.245   30         0
766         1           126            60  ...      0.349   47         1
767         1            93            70  ...      0.315   23         0

[768 rows x 9 columns]>

The total number of Training Data: (514, 1)
The total number of Test Data: (254, 1)

Confusion matrix
[[156  16]
 [ 35  47]]

Accuracy of the classifier: 0.7992125984251969
The value of Precision: 0.746031746031746
The value of Recall: 0.573170731707317
Predicted Value for individual Test Data: [1]
```

5)Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

a)Using SKlearn:

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

# Importing the dataset
dataset = pd.read_csv('salary_data.csv')
X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st:

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)

# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
# Predicting the Test set results
y_pred = regressor.predict(X_test)
print(y_pred)
```

Output:

```
Out[4]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [8]:
```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
print(y_pred)
```

```
[ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]
```

b) Without using SKlearn:

```
import pandas as pd
import numpy as np

class LR():
    def __init__(self):
        self.w = []
    def fit(self, X, y):
        self.w = np.linalg.solve(X.T@X, X.T@y)
    def predict(self, X):
        return X@self.w
    def score(self, X, y):
        SS_reg = np.sum((X@self.w - y)**2)
        SS_tot = np.sum((y - np.mean(y))**2)
        return (1 - (SS_reg/SS_tot))


from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing

fetch_california_housing

data, labels = fetch_california_housing(return_X_y = True)

data.shape, labels.shape

one = np.ones(data.shape[0])
data = np.column_stack((one, data))

X_train,X_test, y_train, y_test = train_test_split(data, labels, train_size = 0.75, random_state = 42)

lro = LR()
lro.fit(X_train, y_train)

lro.w

lro.predict(X_test)


lro.score(X_test, y_test)
```

Output:

```
data.shape, labels.shape
```

```
((20640, 9), (20640,))
```

```
lro.w
```

```
array([-3.70278276e+01,  4.47600069e-01,  9.56752596e-03, -1.24755956e-01,
        7.94471254e-01, -1.43902596e-06, -3.44307993e-03, -4.18555257e-01,
       -4.33405135e-01])
```

```
lro.predict(X_test)
```

```
array([0.72412832, 1.76677807, 2.71151581, ..., 1.72382152, 2.34689276,
       3.52917352])
```

```
lro.score(X_test, y_test)
```

```
0.5910509795491321
```