VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

MACHINE LEARNING
(20CS6PCMAL)

Submitted by

Sushmitha Y V(1BM19CS165)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
2022-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**MACHINE LEARNING**" carried out by **SUSHMITHA Y V (1BM19CS165),** 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 2023. 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.

Name of the Lab-In charge	Prameetha Pai
Designation	Assistant Professor
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

LAB PROGRAM 1:

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

```
import pandas as pd
import numpy as np
d=pd.read_csv("data.csv")
print(d)
att=np.array(d)[:,:-1]
print(att)
tar=np.array(d)[:,-1]
print(tar)
def finds(att, tar):
    for i, val in enumerate(tar):
        if val == "yes":
            res=att[i].copy()
            break
    for i, val in enumerate(att):
        if tar[i] == "yes":
            for x in range (len(res)):
                if val[x] != res[x]:
                    res[x] = "?"
                else:
                    pass
    return res
  print(finds(att,tar))
```

```
    return res
```

```
In [10]: print(finds(att,tar))
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
```

Output:

```
[['sky', 'airtemp', 'humidity', 'wind', 'water', 'forcast', 'enjoysport'], ['
sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes'], ['sunny', 'warm',
 'high', 'strong', 'warm', 'same', 'yes'], ['rainy', 'cold', 'high', 'strong',
 'warm', 'change', 'no'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change
', 'yes']]

 The total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
```

```
['0', '0', '0', '0', '0', '0']

The hypothesis for the training instance 2 is :
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

The hypothesis for the training instance 3 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 4 is :
['sunny', 'warm', '?', 'strong', 'warm', 'same']

The hypothesis for the training instance 5 is :
['sunny', 'warm', '?', 'strong', '?', '?']

The Maximally specific hypothesis for the training instance is
['sunny', 'warm', '?', 'strong', '?', '?']
```

LAB PROGRAM 2:

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

```
import numpy as np
import pandas as pd
data= pd.read_csv("data.csv")
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h=concepts[0].copy()
    general_h=[["?" for i in range(len(specific_h))] for i in range(len(specific_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]=specific_h[x]

        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]='?'

    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, f_final = learn(concepts,target)
s_final
f_final
```

```
In [37]: s_final

Out[37]: array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object)

In [38]: f_final

Out[38]: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Output :

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]
For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?
', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]



For Loop Starts
If instance is Negative
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
 '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?
', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



For Loop Starts
If instance is Positive
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
 '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?
', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

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

LAB PROGRAM 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.

```
import pandas as pd
import math
import numpy as np
data = pd.read_csv("3-dataset.csv")
features=[feat for feat in data]
features.remove("answer")
class Node:
    def __init__(self):
        self.children=[]
        self.value=""
        self.isLeaf=False
        self.pred=""

def entropy(examples):
    pos=0.0
    neg=0.0
    for _, row in examples.iterrows():
        if row["answer"]=="yes":
            pos+=1
        else:
            neg+=1
    if pos==0.0 or neg==0.0:
        return 0.0
    else:
        p=pos/(pos+neg)
        n=neg/(pos+neg)
        return -(p * math.log(p,2) + n * math.log(n,2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain=entropy(examples)
    for u in uniq:
        subdata=examples[examples[attr] == u]
        sub_e = entropy(subdata)
        gain -=(float(len(subdata))/float(len(examples)))*sub_e
    return gain

def ID3(examples, attrs):
    root = Node()
```

```
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata)==0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end=" ")
    print(root.value, end=" ")
    if root.isLeaf:
        print("->", root.pred)
    print()
    for child in root.children:
        printTree(child, depth+1)

root=ID3(data, features)
printTree(root)
```

In [41]:

```
root=ID3(data, features)
printTree(root)
```

```
outlook
        overcast -> ['yes']

        rain
                wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                humidity
                        high -> ['no']

                        normal -> ['yes']
```

```
The test instance: ['rain', 'cool', 'normal', 'strong']
The label for test instance:   no
The test instance: ['sunny', 'mild', 'normal', 'strong']
The label for test instance:   yes
```

LAB PROGRAM 4:

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

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/canada_per_capita_income - canada_per_capita_income.csv')
df
%matplotlib inline
plt.xlabel('year')
plt.ylabel('income')
plt.scatter(df.year,df.income,color='red',marker='+')
new_df = df.drop('income',axis='columns')
new_df
income = df.income
income
reg = linear_model.LinearRegression()
reg.fit(new_df,income)
reg.predict([[2021]])
reg.coef_
reg.intercept_
reg.predict([[2020]])
plt.xlabel('year',fontsize=20)
plt.ylabel('income',fontsize=20)
plt.scatter(df.year,df.income,color='red',marker='+')
plt.plot(df.year,reg.predict(df[['year']]),color='blue')
```

In [ ]:

```
reg.predict([[2021]])
reg.coef_
reg.intercept_
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have vali
d feature names, but LinearRegression was fitted with feature names
  "X does not have valid feature names, but"
```

Out[ ]:

```
-1632210.7578554575
```

In [ ]:

```
reg.predict([[2020]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have vali
d feature names, but LinearRegression was fitted with feature names
  "X does not have valid feature names, but"
```

Out[ ]:

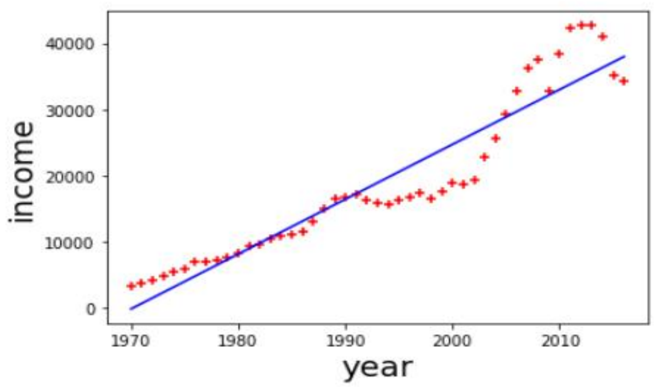
```
array([41288.69409442])
```

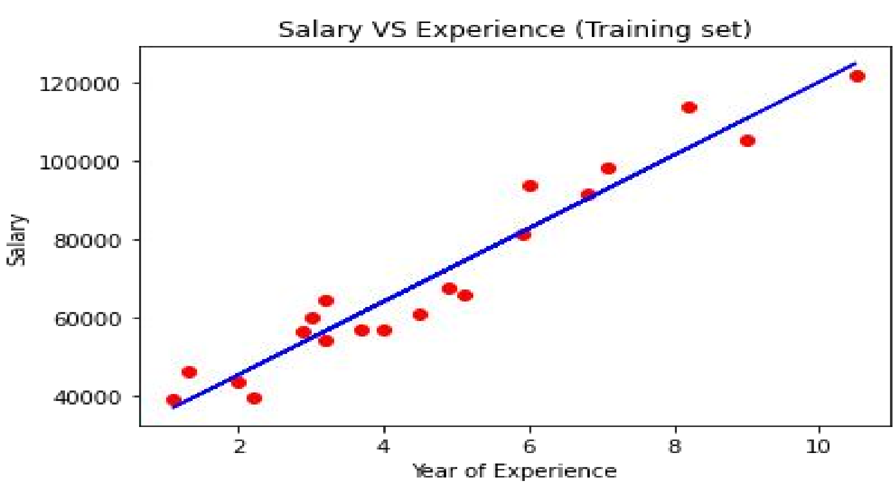
In [ ]:

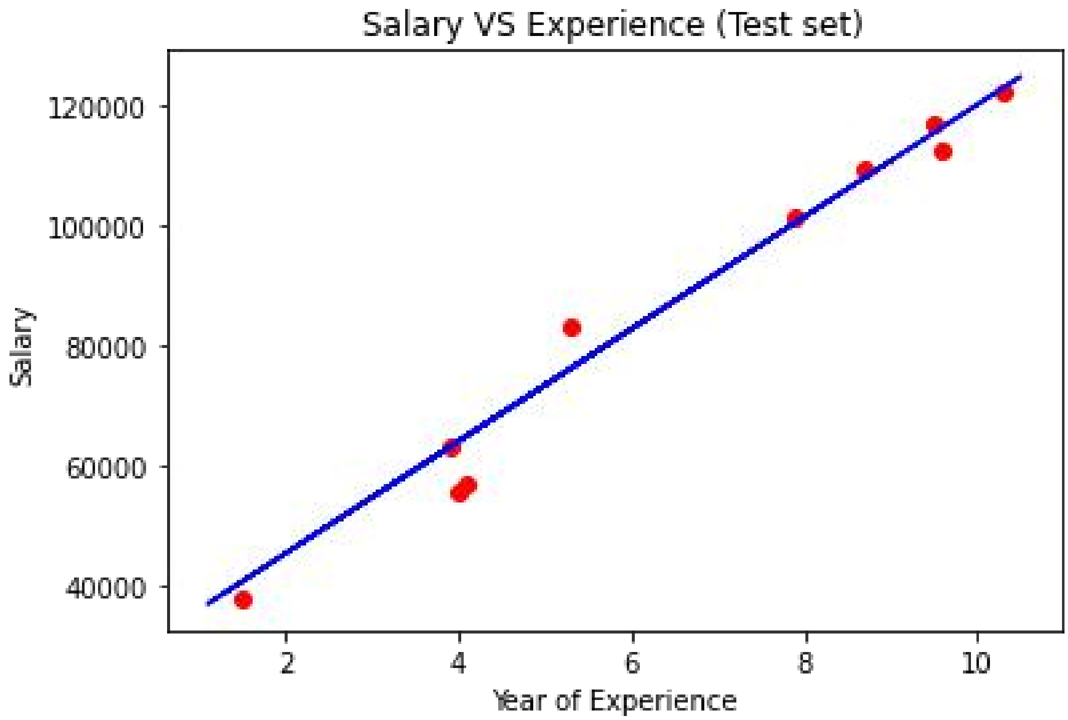
```
plt.xlabel('year',fontsize=20)
plt.ylabel('income',fontsize=20)
plt.scatter(df.year,df.income,color='red',marker='+')
plt.plot(df.year,reg.predict(df[['year']]),color='blue')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f1d402a2590>]
```






Salary VS Experience (Test set)
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Year of Experience

LAB PROGRAM 5:

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

```
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups()
data.target_names
categories = ['talk.religion.misc', 'soc.religion.christian',
              'sci.space', 'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
print(train.data[5])
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
model.fit(train.data, train.target)
labels = model.predict(test.data)
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
mat = confusion_matrix(test.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=train.target_names, yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
predict_category('Rocket launch in 3 months')
```

```
In [22]: predict_category('Rocket launch in 3 months')

Out[22]: 'sci.space'
```

```
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

```
predict_category('determining the screen resolution')
```

```
'comp.graphics'
```

```
predict_category('what is 650 cc?')
```

```
'rec.motorcycles'
```

```
predict_category('launching payload')
```

```
'sci.space'
```

```
Split 768 rows into train=514 and test=254rows
Accuracy of the classifier is: 74.40944881889764%
```

Gaussian Naïve Bayes Algorithm Output:

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thic
kness  insulin   bmi  \
0           6           148            72          35         0  33.6
1           1            85            66          29         0  26.6
2           8           183            64           0         0  23.3
3           1            89            66          23        94  28.1
4           0           137            40          35       168  43.1
..        ...           ...           ...         ...       ...   ...
763        10           101            76          48       180  32.9
764         2           122            70          27         0  36.8
765         5           121            72          23       112  26.2
766         1           126            60           0         0  30.1
767         1            93            70          31         0  30.4

     diab_pred  age  diabetes
0        0.627   50         1
1        0.351   31         0
2        0.672   32         1
3        0.167   21         0
4        2.288   33         1
..         ...  ...       ...
763      0.171   63         0
764      0.340   27         0
765      0.245   30         0
766      0.349   47         1
767      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
[[147  26]
 [ 28  53]]

 Accuracy of the classifier is 0.7874015748031497

 The value of Precision 0.6708860759493671

 The value of Recall 0.654320987654321
Predicted Value for individual Test Data: [1]
```

LAB PROGRAM 6:

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

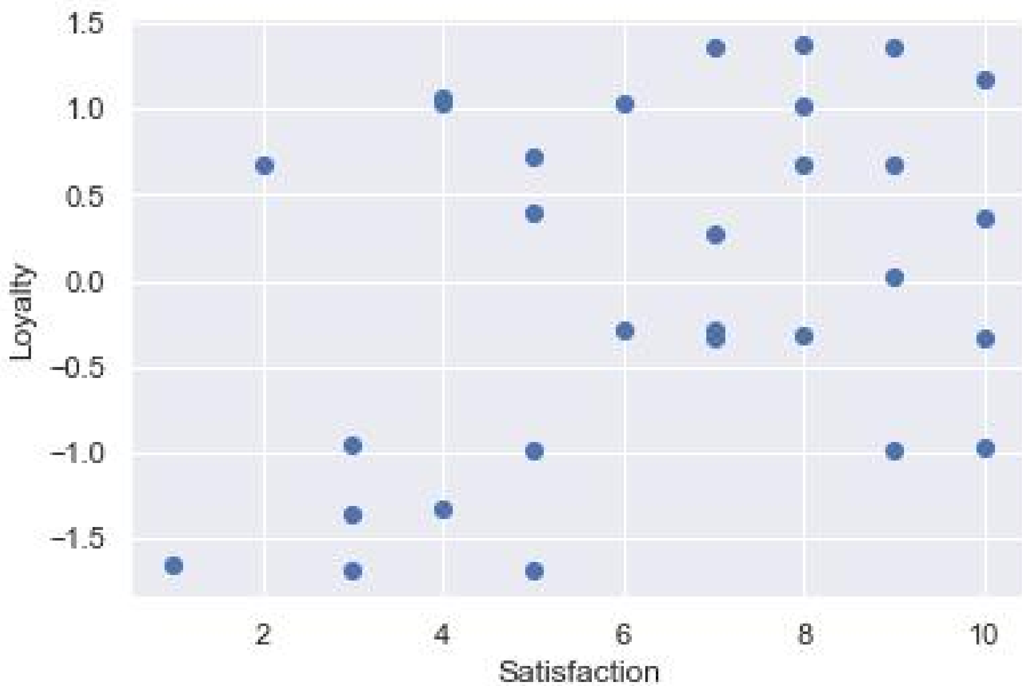
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
data = pd.read_csv('sample.csv')
data
```

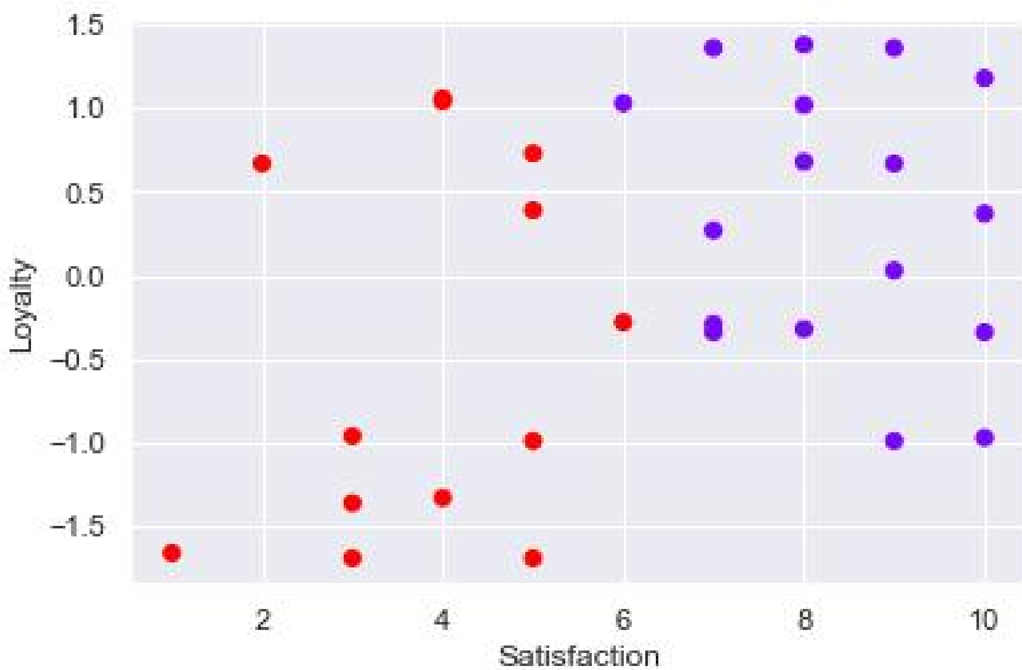
- **DATA OUTPUT**

	Satisfaction	Loyalty
0	4	-1.33
1	6	-0.28
2	5	-0.99
3	7	-0.29
4	4	1.06
5	1	-1.66
6	10	-0.97
7	8	-0.32
8	8	1.02
9	8	0.68
10	10	-0.34
11	5	0.39

	Satisfaction	Loyalty
12	5	-1.69
13	2	0.67
14	7	0.27
15	9	1.36
16	8	1.38
17	7	1.36
18	7	-0.34
19	9	0.67
20	10	1.18
21	3	-1.69
22	4	1.04
23	3	-0.96
24	6	1.03
25	9	-0.99
26	10	0.37
27	9	0.03
28	3	-1.36

	Satisfaction	Loyalty
29	5	0.73





LAB PROGRAM 7:

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

```
import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
model=
BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdiseas
e'),('heartdisease','restecg'),('heartdisease','chol')])
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
model= BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdi
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')
```

```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:
```

```
HeartDiseasetest_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

```
Finding Elimination Order: : 100%|████████████████████████████████████████| 5/5 [00:00<00:00, 2
500.78it/s]
Eliminating: chol: 100%|████████████████████████████████████████| 5/5 [00:00<00:00,
185.63it/s]
 1. Probability of HeartDisease given evidence= restecg
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1012 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2392 |
+-----------------+---------------------+
| heartdisease(3) |              0.2015 |
+-----------------+---------------------+
| heartdisease(4) |              0.4581 |
+-----------------+---------------------+
```

```
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
Finding Elimination Order: : 100%|████████████████████████████████████████| 5/5 [00:00<00:00, 2
507.06it/s]
Eliminating: restecg: 100%|████████████████████████████████████████| 5/5 [00:00<00:00,
179.06it/s]
 2. Probability of HeartDisease given evidence= cp
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3610 |
+-----------------+---------------------+
| heartdisease(1) |              0.2159 |
+-----------------+---------------------+
| heartdisease(2) |              0.1373 |
+-----------------+---------------------+
| heartdisease(3) |              0.1537 |
+-----------------+---------------------+
| heartdisease(4) |              0.1321 |
+-----------------+---------------------+
```

LAB PROGRAM 8:

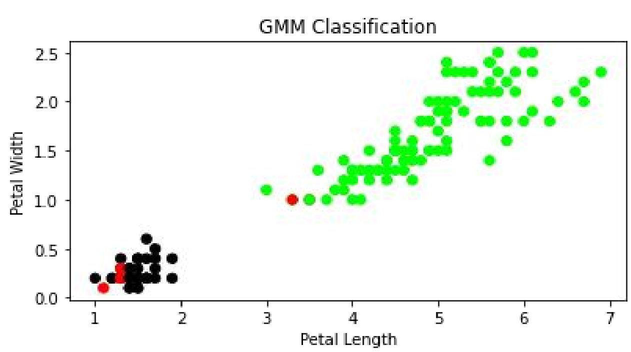
Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

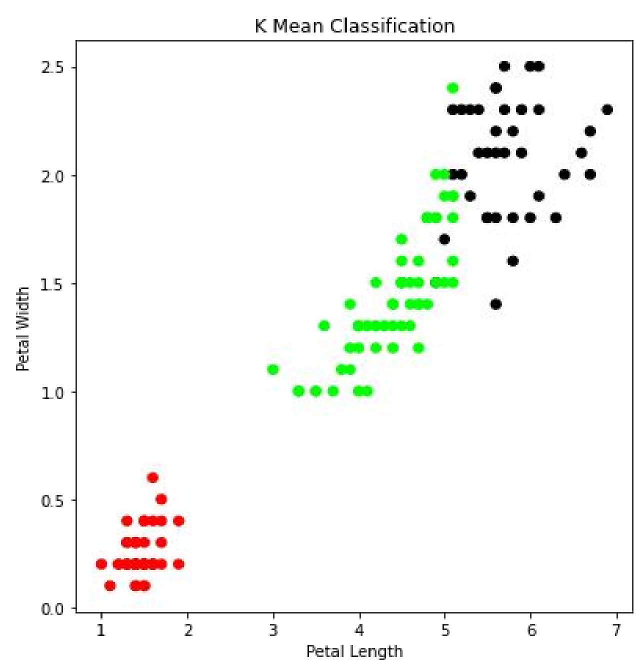
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np

iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)


plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))


from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)
```

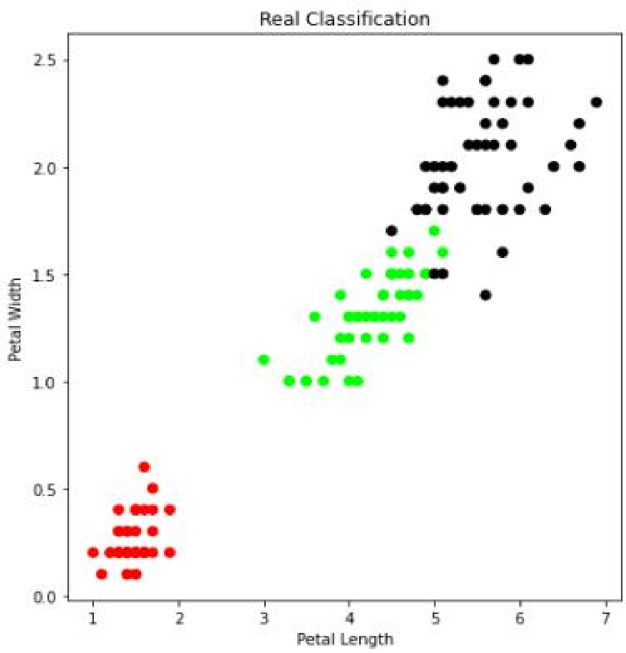
```python
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

OUTPUT:

```
The accuracy score of K-Mean:  0.8933333333333333
The Confusion matrixof K-Mean:  [[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
The accuracy score of EM:  0.35333333333333333
The Confusion matrix of EM:  [[ 5  0 45]
 [ 2 48  0]
 [ 0 50  0]]
```







Real Classification
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

LAB PROGRAM 9:

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
Y = iris.target

print('sepal-length','sepal-width','petal-length','petal-width')
print(X)
print('target')
print(Y)
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.3)
classier = KNeighborsClassifier(n_neighbors=5)
classier.fit(x_train, y_train)
y_pred=classier.predict(x_test)
print('confusion matrix')
print(confusion_matrix(y_test,y_pred))
print('accuracy')
print(classification_report(y_test,y_pred))
```

```
y_pred=classier.predict(x_test)
```

```
print('confusion matrix')
print(confusion_matrix(y_test,y_pred))
```

```
confusion matrix
[[16  0  0]
 [ 0 12  1]
 [ 0  0 16]]
```

```
print('accuracy')
print(classification_report(y_test,y_pred))
```

```
accuracy
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      0.92      0.96        13
           2       0.94      1.00      0.97        16

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.98        45
weighted avg       0.98      0.98      0.98        45
```

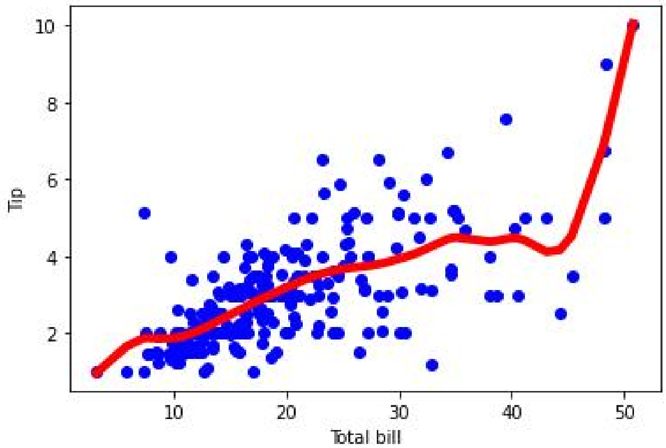
LAB PROGRAM 10:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]

one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE

ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
```

```
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):
    x0 = np.r_[1, x0]
    X = np.c_[np.ones(len(X)), X]

    xw = X.T * radial_kernel(x0, X, tau)
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    return x0 @ beta

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))


n = 1000
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot_lwr(tau):

    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot
```

show(gridplot([
[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))




```
def plot_lwr(tau):

    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot


show(gridplot([
[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))
```

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-2.98256634 -2.99368144 -3.05914505 -3.03174286 -3.07963801 -2.85954046
 -2.92988067 -2.958209   -2.96962333]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```