MACHINE LEARNING PROGRAMS

Program 1:

Implement and demonstratethe FIND-Salgorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

```
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('data1.csv'))

concepts = np.array(data.iloc[:,0:-1])
print(concepts)

target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()

    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] = "?"
    return specific_h

specific_h = learn(concepts, target)
print(specific_h)
```

DATASET:

Sky	Airtemp	Humidity	Wind	Water	Forecast	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

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']
['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

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.DataFrame(data=pd.read_csv('data2.csv'))
print('The Dataset is: \n')
print(data)

concepts = np.array(data.iloc[:,0:-1])

print('\n The Concepts are: \n',concepts)
target = np.array(data.iloc[:,-1])
print('\nThe target is: \n',target)

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] = '?'


        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, g_final = learn(concepts, target)
print("\n\nFinal S:", s_final)
print("\n\nFinal G:", g_final)
```

DATASET and OUTPUT:

```
The Dataset is:

     Sky Airtemp Humidity    Wind Water Forecast 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 Concepts 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']


Final S: ['Sunny' 'Warm' '?' 'Strong' '?' '?']


Final G: [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

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 numpy as np
from pprint import pprint

dataset = pd.read_csv('data3.csv')
features=['Outlook','Temperature','Humidity','Wind']
def entropy(target_col):
   elements,counts = np.unique(target_col,return_counts = True)
   entropy = np.sum([(-counts[i]/np.sum(counts))*np.log2(counts[i]/np.sum(counts)) for i in
range(len(elements))])
   return entropy

def InfoGain(data,split_attribute_name,target_name="EnjoySport"):
    total_entropy = entropy(data[target_name])
    vals,counts= np.unique(data[split_attribute_name],return_counts=True)
    Weighted_Entropy =
np.sum([(counts[i]/np.sum(counts))*entropy(data.where(data[split_attribute_name]==vals[i]).dropna()[tar
get_name]) for i in range(len(vals))])
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain

def ID3(data,originaldata,features,target_attribute_name="class",parent_node_class = None):

   if len(np.unique(data[target_attribute_name])) <= 1:
      return np.unique(data[target_attribute_name])[0]

   elif len(data)==0:
      return
np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name
],return_counts=True)[1])]

   elif len(features) ==0:
      return parent_node_class

   else:
      parent_node_class =
np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name],return_count
s=True)[1])]
      item_values = [InfoGain(data,feature,target_attribute_name) for feature in features]
      best_feature_index = np.argmax(item_values)
      best_feature = features[best_feature_index]
      tree = {best_feature:{}}
      features = [i for i in features if i != best_feature]
      for value in np.unique(data[best_feature]):
```

```
        value = value
        sub_data = data.where(data[best_feature] == value).dropna()
        subtree = ID3(sub_data,dataset,features,target_attribute_name,parent_node_class)
        tree[best_feature][value] = subtree

    return(tree)

def predict(query,tree,default = 1):
  for key in list(query.keys()):
    if key in list(tree.keys()):
      try:
        result = tree[key][query[key]]
      except:
        return default
      result = tree[key][query[key]]
      if isinstance(result,dict):
        return predict(query,result)
      else:
        return result

training_data = dataset.iloc[:13]
print(training_data)

features=['Outlook','Temperature','Humidity','Wind']
target_attribute_name="class"
parent_node_class=None
tree=ID3(training_data,training_data,features,target_attribute_name,parent_node_class)
pprint(tree)

query=dataset.iloc[:,:-1].to_dict(orient="records")
result=predict(query[10],tree,1.0)
print(result)
```

DATASET and OUTPUT:

```
     Outlook Temperature Humidity    Wind class
0      Sunny         Hot     High    Weak    No
1      Sunny         Hot     High  Strong    No
2   Overcast         Hot     High    Weak   Yes
3       Rain        Mild     High    Weak   Yes
4       Rain        Cool   Normal    Weak   Yes
5       Rain        Cool   Normal  Strong    No
6   Overcast        Cool   Normal  Strong   Yes
7      Sunny        Mild     High    Weak    No
8      Sunny        Cool   Normal    Weak   Yes
9       Rain        Mild   Normal    Weak   Yes
10     Sunny        Mild   Normal  Strong   Yes
11  Overcast        Mild     High  Strong   Yes
12  Overcast         Hot   Normal    Weak   Yes
{'Outlook': {'Overcast': 'Yes',
             'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
Yes
```

Program 4:
Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0)
y = y/100
def sigmoid (x):
    return 1/(1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)
epoch=7000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    h_ip=np.dot(X,wh) + bh
    h_act = sigmoid(h_ip)
    o_ip=np.dot(h_act,wout) + bout
    output = sigmoid(o_ip)
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    Eh = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(h_act)
    d_hidden = Eh * hiddengrad
    wout += h_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hidden) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Learned Output: \n" ,output)
```

OUTPUT->

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Learned Output:
 [[0.89539873]
 [0.87714782]
 [0.89670196]]
```

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

```
import pandas as pd
import numpy as np
import csv

#reading the CSV file
data1 = pd.read_csv('data5.csv')

#Separating all Yes in one Dataframe
df1 = data1[data1['class'] == 'Yes']

#Separating all No in one Dataframe
df2 = data1[data1['class'] == 'No']

#Declaring list variable for storing the input from the user
inputlist1=[]

#Initializing the Header value in head list
head=['Outlook','Temperature','Humidity','Wind','class']
inputlist1.append(head)
count=0

def counting(inputlist1,j,str1,count):
   if(inputlist1[j][4]==str1):
      count=count+1
   return count

print("\n\t\t Naive Bayesian Classifier")

with open('data5.csv','r') as csv_file1:
          csv_reader1=csv.reader(csv_file1)
          for i in range(11):
             next(csv_reader1)
          for line1 in csv_reader1:
             inputlist1.append(line1)
          for j in range(1,len(inputlist1)):
             print("\nThe ",j,"Test data is:\n",head[0]," = ",inputlist1[j][0],", ",head[1]," = ",inputlist1[j][1],",
",head[2]," = ",inputlist1[j][2],", ",head[3]," = ",inputlist1[j][3])

             #Declaring list variable for storing the result
             listyes=list()
             listno=list()
             resultyes=0.0
             resultno=0.0

             #Evaluating the Probability
```

```
        for d in range(4):
            listyes.append(df1.loc[df1[head[d]]==inputlist1[j][d],head[d]].count()/len(df1))
            listno.append(df2.loc[df2[head[d]]==inputlist1[j][d],head[d]].count()/len(df2))
        resultyes = np.prod(np.array(listyes))*(len(df1)/len(data1))
        resultno = np.prod(np.array(listno))*(len(df2)/len(data1))
        print("Probability of Yes: ",resultyes,"\nProbability of No: ",resultno)
        if resultyes>resultno:
          print("Classified as YES\n")
          count=counting(inputlist1,j,'Yes',count)
        else:
          print("Classified as NO\n")
          count=counting(inputlist1,j,'No',count)

print("\nAccuracy of the Classifier is: ",count/(len(inputlist1)-1) )
```

DATASET:

Outlook	Temperature	Humidity	Wind	class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

OUTPUT:

```
                Naive Bayesian Classifier

The  1 Test data is:
 Outlook  =  Sunny ,  Temperature  =  Mild ,  Humidity  =  Normal ,  Wind  =  Strong
Probability of Yes:  0.014109347442680773
Probability of No:  0.010285714285714285
Classified as YES


The  2 Test data is:
 Outlook  =  Overcast ,  Temperature  =  Mild ,  Humidity  =  High ,  Wind  =  Strong
Probability of Yes:  0.014109347442680773
Probability of No:  0.0
Classified as YES


The  3 Test data is:
 Outlook  =  Overcast ,  Temperature  =  Hot ,  Humidity  =  Normal ,  Wind  =  Weak
Probability of Yes:  0.028218694885361547
Probability of No:  0.0
Classified as YES


The  4 Test data is:
 Outlook  =  Rain ,  Temperature  =  Mild ,  Humidity  =  High ,  Wind  =  Strong
Probability of Yes:  0.010582010582010581
Probability of No:  0.027428571428571438
Classified as NO


Accuracy of the Classifier is:  1.0
```

Program 6:
Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set

```
import pandas as pd
msg=pd.read_csv('data6.csv',names=['message','label'])

print('\n Total instances in the dataset: ',msg.shape[0])

msg['labelnum']=msg.label.map({'pos':1,'neg':0})
x=msg.message
y=msg.labelnum

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y)

print('\n Dataset is Split into Training and Testing Samples')
print('\n Training Instances: ',xtrain.shape[0])
print(xtrain)
print('\n Testing Instances :',xtest.shape[0])
print(xtest)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm = count_vect.transform(xtest)
print('\n Total features extracted using CountVectorizer: ',xtrain_dtm.shape[1])

print('\n Features for first 5 training instances are listed below')
df = pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df[0:5])

from sklearn.naive_bayes import MultinomialNB
clf=MultinomialNB().fit(xtrain_dtm,ytrain)
predicted=clf.predict(xtest_dtm)

print('\nClassification Results of Test Dataset are:\n')
for doc, p in zip(xtest,predicted):
    pred = 'pos' if p==1 else 'neg'
    print('%s -->  %s '%(doc,pred))

from sklearn import metrics
print('\nAccuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\nConfusion Matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\nRecall and Precision')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

DATASET:

I love this sandwich	pos
This is an amazing place	pos
I feel very good about these places	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg
My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemey's house today	neg

OUTPUT:

```
 Total instances in the dataset:  18

 Dataset is Split into Training and Testing Samples

 Training Instances:  13
17      I went to my enemey's house today
14                  What a great holiday
2     I feel very good about these places
5           I do not like this restaurant
0                   I love this sandwich
4                   What an awesome view
10              This is an awesome place
15         That is a bad locality to stay
7                  I can't deal with this
3                    This is my best work
9                     My boss is horrible
13      I am sick and tired of this place
8                   He is my sworn enemy
Name: message, dtype: object

 Testing Instances : 5
6                  I am tired of this stuff
16           We will have good fun tomorrow
1                 This is an amazing place
11    I do not like the taste of this juice
12                          I love to dance
Name: message, dtype: object

 Total features extracted using CountVectorizer:  46

 Features for first 5 training instances are listed below
   about  am  an  and  awesome  bad  ...   very  view  went  what  with  work
0      0   0   0    0        0    0  ...      0     0     1     0     0     0
1      0   0   0    0        0    0  ...      0     0     0     1     0     0
2      1   0   0    0        0    0  ...      1     0     0     0     0     0
3      0   0   0    0        0    0  ...      0     0     0     0     0     0
4      0   0   0    0        0    0  ...      0     0     0     0     0     0

[5 rows x 46 columns]
Classification Results of Test Dataset are:

I am tired of this stuff -->  neg
We will have good fun tomorrow -->  pos
This is an amazing place -->  pos
I do not like the taste of this juice -->  neg
I love to dance -->  neg

Accuracy of the classifier is 0.8

Confusion Matrix
[[2 0]
 [1 2]]

Recall and Precision
0.6666666666666666
1.0
```

Program 7:
Write a program to construct aBayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

```
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

#Read the attributes
lines = list(csv.reader(open('data7_names.csv', 'r')));
attributes = lines[0]

#Read Cleveland Heart dicease data
heartDisease = pd.read_csv('data7_heart.csv', names = attributes)
heartDisease = heartDisease.replace('?', np.nan)

# Model Baysian Network
model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'), ('sex', 'trestbps'),
('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),
('heartdisease','restecg'),('heartdisease','thalach'),('heartdisease','chol')])


print('\nBayesian Network Nodes are: ')
print('\t',model.nodes())
print('\nBayesian Network Edges are:')
print('\t',model.edges())


# Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPDs using Maximum Likelihood Estimators...');
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

# Inferencing with Bayesian Network
print('\nInferencing with Bayesian Network:')
HeartDisease_infer = VariableElimination(model)

# Computing the probability of bronc given smoke.
print('\n1.Probability of HeartDisease given Age=28')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'age': 28})
print(q['heartdisease'])
print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')
q = HeartDisease_infer.query(variables=['heartdisease'], evidence={'chol': 100})
print(q['heartdisease'])
```

DATASET: data7_name.csv

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease

Data7_heart.csv

63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
53	1	4	140	203	1	2	155	1	3.1	3	0	7	1
57	1	4	140	192	0	0	148	0	0.4	2	0	6	0
56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
56	1	3	130	256	1	2	142	1	0.6	2	1	6	2
44	1	2	120	263	0	0	173	0	0	1	0	7	0
52	1	3	172	199	1	0	162	0	0.5	1	0	7	0
57	1	3	150	168	0	0	174	0	1.6	1	0	3	0
48	1	2	110	229	0	0	168	0	1	3	0	7	1
54	1	4	140	239	0	0	160	0	1.2	1	0	3	0
48	0	3	130	275	0	0	139	0	0.2	1	0	3	0
49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
64	1	1	110	211	0	2	144	1	1.8	2	0	3	0
58	0	1	150	283	1	2	162	0	1	1	0	3	0

OUTPUT:

```
Bayesian Network Nodes are:
        ['heartdisease', 'sex', 'fbs', 'exang', 'restecg', 'chol', 'thalach', 'trestbps', 'age']

Bayesian Network Edges are:
        [('heartdisease', 'restecg'), ('heartdisease', 'chol'), ('heartdisease', 'thalach'), ('sex',
'trestbps'), ('fbs', 'heartdisease'), ('exang', 'trestbps'), ('trestbps', 'heartdisease'), ('age',
'trestbps'), ('age', 'fbs')]

Learning CPDs using Maximum Likelihood Estimators...

Inferencing with Bayesian Network:

1.Probability of HeartDisease given Age=28
```

heartdisease	phi(heartdisease)
heartdisease_0	0.6791
heartdisease_1	0.1212
heartdisease_2	0.0810
heartdisease_3	0.0939
heartdisease_4	0.0247

2. Probability of HeartDisease given chol (Cholestoral) =100

heartdisease	phi(heartdisease)
heartdisease_0	0.5400
heartdisease_1	0.1533
heartdisease_2	0.1303
heartdisease_3	0.1259
heartdisease_4	0.0506

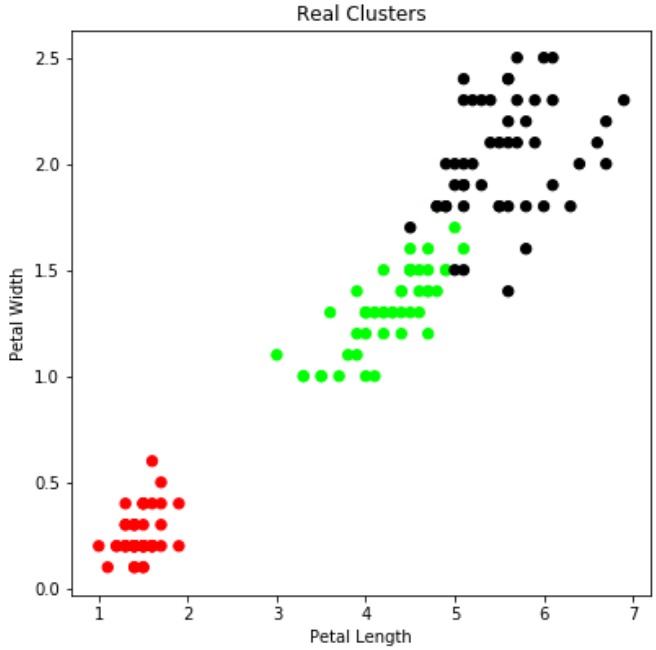
Program 8:
Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

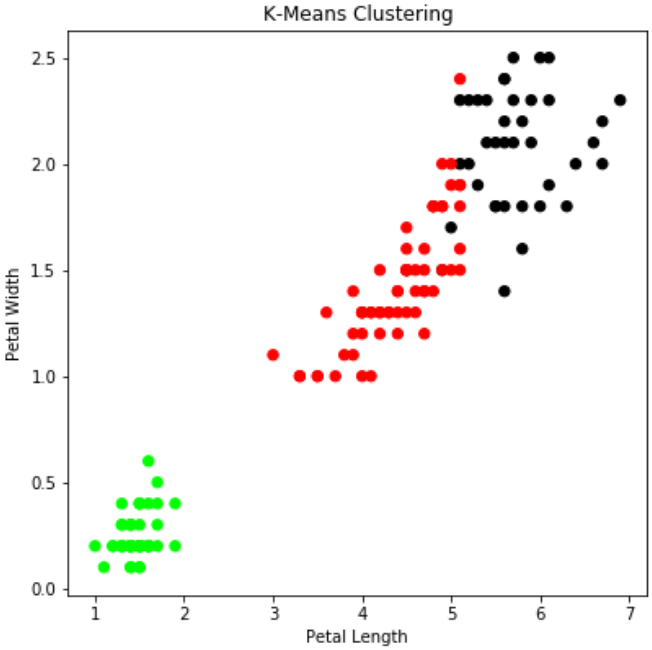
```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

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=(14,14))
colormap = np.array(['red', 'lime', 'black'])

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

plt.subplot(2, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K-Means Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
gmm_y = gmm.predict(xs)
plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[gmm_y], s=40)
plt.title('GMM Clustering')
plt.xlabel('Petal Length')
```

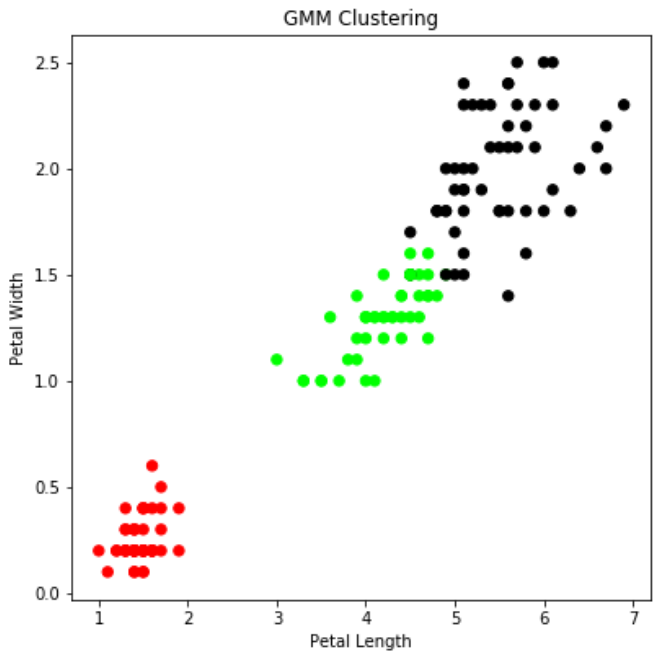
```
plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the')
print(' true labels more closely than the Kmeans.')
```

OUTPUT:

```
Observation: The GMM using EM algorithm based clustering matched the
 true labels more closely than the Kmeans.
```







Program 9:

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
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()
iris_data=iris.data
iris_labels=iris.target

x_train,x_test,y_train,y_test=train_test_split(iris_data,iris_labels,test_size=0.20)

classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train,y_train)

y_pred=classifier.predict(x_test)

print('Confusion Matrix is as follows')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

OUTPUT:

```
Confusion Matrix is as follows
[[ 8  0  0]
 [ 0 10  1]
 [ 0  1 10]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       0.91      0.91      0.91        11
           2       0.91      0.91      0.91        11

   micro avg       0.93      0.93      0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

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 *
import operator
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy.linalg
from scipy.stats.stats import pearsonr


def kernel(point,xmat,k):
    m,n= shape(xmat)
    weights=mat(eye((m)))
    for j in range(m):
     diff = point - X[j]
     weights[j,j]= exp(diff*diff.T/(-2*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=shape(xmat)
    ypred=zeros(m)
    for i in range(m):
     ypred[i]=xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

#load data points
data=pd.read_csv('data10.csv')
bill=array(data.totbill)
tip=array(data.tip)

#Preparing and add 1 in bill
mbill=mat(bill)
mtip=mat(tip)
m=shape(mbill)[1]
one=mat(ones(m))
X=hstack((one.T,mbill.T))

#set k here
ypred=localWeightRegression(X,mtip,0.5)
SortIndex=X[:,1].argsort(0)
```

```
xsort=X[SortIndex][:,0]
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.scatter(bill,tip,color='green')
ax.plot(xsort[:,1],ypred[SortIndex],color='red',linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

DATASET:

obs	totbill	tip	sex	smoker	day	time	size
1	16.99	1.01	F	No	Sun	Night	2
2	10.34	1.66	M	No	Sun	Night	3
3	21.01	3.5	M	No	Sun	Night	3
4	23.68	3.31	M	No	Sun	Night	2
5	24.59	3.61	F	No	Sun	Night	4
6	25.29	4.71	M	No	Sun	Night	4
7	8.77	2	M	No	Sun	Night	2
8	26.88	3.12	M	No	Sun	Night	4
9	15.04	1.96	M	No	Sun	Night	2
10	14.78	3.23	M	No	Sun	Night	2
11	10.27	1.71	M	No	Sun	Night	2
12	35.26	5	F	No	Sun	Night	4
13	15.42	1.57	M	No	Sun	Night	2
14	18.43	3	M	No	Sun	Night	4
15	14.83	3.02	F	No	Sun	Night	2
16	21.58	3.92	M	No	Sun	Night	2
17	10.33	1.67	F	No	Sun	Night	3
18	16.29	3.71	M	No	Sun	Night	3
19	16.97	3.5	F	No	Sun	Night	3
20	20.65	3.35	M	No	Sat	Night	3
21	17.92	4.08	M	No	Sat	Night	2
22	20.29	2.75	F	No	Sat	Night	2
23	15.77	2.23	F	No	Sat	Night	2

OUTPUT: