Best Youtube video link

1. https://youtu.be/_sNNxXSfyY0

2. https://youtu.be/rKxRUnT1f8Y

3. https://youtu.be/wqMFutUxBMM, https://youtu.be/9PcbUh_4PJE

4. https://youtu.be/tY4qKXwmTe4, https://youtu.be/B4Cblyc_7W4

5. https://youtu.be/F7u9e9JtuTc

6. https://youtu.be/F7u9e9JtuTc , https://youtu.be/uSHqXNccpKA

7. https://www.youtube.com/watch?v=Jk4-POycbR8

8. https://youtu.be/RPErP1lDXDk

9. https://youtu.be/JtTohICZNCw

10. https://youtu.be/HYKILaI1J5E

DATASET

https://drive.google.com/drive/folders/1TZAoKYN5LML-cnUt2tXYAZizksUuiUtp?usp=share_link

Program 1 . find S

```
import random

import csv

attributes = [['Sunny','Rainy'],
              ['Warm','Cold'],
              ['Normal','High'],
              ['Strong','Weak'],
              ['Warm','Cool'],
              ['Same','Change']]

num_attributes = len(attributes)

print (" \n The most general hypothesis : ['?','?','?','?','?','?']\n")
print ("\n The most specific hypothesis : ['0','0','0','0','0','0']\n")

a = []
print("\n The Given Training Data Set \n")
with open("Program1dataset.csv", 'r') as csvFile:
   reader = csv.reader(csvFile)
   for row in reader:
      a.append (row)
      print(row)

print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

# Comparing with First Training Example
for j in range(0,num_attributes):
      hypothesis[j] = a[0][j];
```

```python
# Comparing with Remaining Training Examples of Given Data Set
print("\n Find S: Finding a Maximally Specific Hypothesis\n")


for i in range(0,len(a)):
   if a[i][num_attributes]=='Yes':
         for j in range(0,num_attributes):
            if a[i][j]!=hypothesis[j]:
               hypothesis[j]='?'
            else :
               hypothesis[j]= a[i][j]
   print(" For Training Example No :{0} the hypothesis is ".format(i),hypothesis)
print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

OUTPUT:

```
The most general hypothesis : ['?','?','?','?','?','?']
The most specific hypothesis : ['0','0','0','0','0','0']


 The Given Training Data Set

['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 initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

 Find S: Finding a Maximally Specific Hypothesis

 For Training Example No :0 the hypothesis is  ['sunny', 'warm', 'norma
l', 'strong', 'warm', 'same']
 For Training Example No :1 the hypothesis is  ['sunny', 'warm', '?', '
strong', 'warm', 'same']
 For Training Example No :2 the hypothesis is  ['sunny', 'warm', '?', '
strong', 'warm', 'same']
 For Training Example No :3 the hypothesis is  ['sunny', 'warm', '?', '
strong', '?', '?']

 The Maximally Specific Hypothesis for a given Training Examples :

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

PROGRAM 2 : Candidate Elimination

```
import csv


with open("Program1dataset.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
```

```
            if j != '?':
                gh.append(i)
                break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

```
OUTPUT:
Step 1 of Candidate Elimination Algorithm
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '
?', '?'], ['?', '?', '?', '?', '?', '?']]

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

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

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

Final Specific hypothesis:
 ['sunny', 'warm', '?', 'strong', '?', '?']

Final General hypothesis:
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

PROGRAM 3: ID3

```
import pandas as pd
from pprint import pprint
from sklearn.feature_selection import mutual_info_classif
from collections import Counter

def id3(df, target_attribute, attribute_names, default_class=None):
    cnt=Counter(x for x in df[target_attribute])
    if len(cnt)==1:
        return next(iter(cnt))

    elif df.empty or (not attribute_names):
         return default_class

    else:
        gainz = mutual_info_classif(df[attribute_names],df[target_attribute],discrete_features=True)
        index_of_max=gainz.tolist().index(max(gainz))
        best_attr=attribute_names[index_of_max]
        tree={best_attr:{}}
        remaining_attribute_names=[i for i in attribute_names if i!=best_attr]

        for attr_val, data_subset in df.groupby(best_attr):
            subtree=id3(data_subset, target_attribute, remaining_attribute_names,default_class)
            tree[best_attr][attr_val]=subtree

        return tree


df=pd.read_csv("Playtennis.csv")

attribute_names=df.columns.tolist()
print("List of attribut name")

attribute_names.remove("Target")

for colname in df.select_dtypes("object"):
    df[colname], _ = df[colname].factorize()

print(df)

tree= id3(df,"Target", attribute_names)
print("The tree structure")
pprint(tree)
```

OUTPUT:

```
List of attribut name
    Outlook  Temperature  Humidity  Wind  Target
```

0	0	0	0	0	0
1	0	0	0	1	0
2	1	0	0	0	1
3	2	1	0	0	1
4	2	2	1	0	1
5	2	2	1	1	0
6	1	2	1	1	1
7	0	1	0	0	0
8	0	2	1	0	1
9	2	1	1	0	1
10	0	1	1	1	1
11	1	1	0	1	1
12	1	0	1	0	1
13	2	1	0	1	0

```
The tree structure
{'Outlook': {0: {'Humidity': {0: 0, 1: 1}}, 1: 1, 2: {'Wind': {0: 1, 1: 0}}}}
```

PROGRAM 4: Backpropagation

```
import numpy as np # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)
y = np.array(([92], [86], [89]), dtype=float)# Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize
y = y/100

def sigmoid(x):
  return 1/(1 + np.exp(-x))
def sigmoid_grad(x):
  return x * (1 - x)

# Variable initialization
epoch=1000              #Setting training iterations
eta =0.2                #Setting learning rate (eta)
input_neurons = 2       #number of features in data set
hidden_neurons = 3      #number of hidden layers neurons
output_neurons = 1      #number of neurons at output layer

# Weight and bias - Random initialization
wh=np.random.uniform(size=(input_neurons,hidden_neurons)) # 2x3
bh=np.random.uniform(size=(1,hidden_neurons))      # 1x3
wout=np.random.uniform(size=(hidden_neurons,output_neurons)) # 1x1
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):
#Forward Propogation
  h_ip=np.dot(X,wh) + bh          # Dot product + bias
  h_act = sigmoid(h_ip)           # Activation function
  o_ip=np.dot(h_act,wout) + bout
  output = sigmoid(o_ip)

#Backpropagation
  # Error at Output layer
  Eo = y-output        # Error at o/p
  outgrad = sigmoid_grad(output)
  d_output = Eo* outgrad          # Errj=Oj(1-Oj)(Tj-Oj)

  # Error at Hidden later
  Eh = d_output.dot(wout.T)       # .T means transpose
  hiddengrad = sigmoid_grad(h_act)          # How much hidden layer wts contributed to error
  d_hidden = Eh * hiddengrad
  wout += h_act.T.dot(d_output) *eta        # Dotproduct of nextlayererror and currentlayerop
  wh += X.T.dot(d_hidden) *eta
```

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

OUTPUT:

```
Normalized Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89541292]
 [0.8790127 ]
 [0.89494954]]
```

PROGRAM 5: Naïve Bayesian Classifier

```
# import necessary libraries
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Load Data from CSV
data = pd.read_csv('Playtennis.csv')
print("The first 5 Values of data is :\n", data.head())

# obtain train data and train output
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train data is\n", X.head())

y = data.iloc[:, -1]
print("\nThe First 5 values of train output is\n", y.head())

# convert them in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Wind = LabelEncoder()
X.Wind = le_Wind.fit_transform(X.Wind)
```

```
print("\nNow the Train output is\n", X.head())

le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20)

classifier = GaussianNB()
classifier.fit(X_train, y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

OUTPUT:

```
The first 5 Values of data is :
     Outlook Temperature Humidity    Wind Target
0      Sunny         Hot     High    Weak     No
1      Sunny         Hot     High  Strong     No
2   Overcast         Hot     High    Weak    Yes
3      Rainy        Mild     High    Weak    Yes
4      Rainy        Cool   Normal    Weak    Yes

The First 5 values of the train data is
     Outlook Temperature Humidity    Wind
0      Sunny         Hot     High    Weak
1      Sunny         Hot     High  Strong
2   Overcast         Hot     High    Weak
3      Rainy        Mild     High    Weak
4      Rainy        Cool   Normal    Weak

The First 5 values of train output is
 0      No
1      No
2     Yes
3     Yes
4     Yes
Name: Target, dtype: object

Now the Train output is
    Outlook  Temperature  Humidity  Wind
0        2            1         0     1
1        2            1         0     0
```

```
2        0            1         0     1
3        1            2         0     1
4        1            0         1     1

Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Accuracy is: 1.0
```

PROGRAM 6: Naïve Bayesian Text Classifier

```
import pandas as pd
msg=pd.read_csv("Program6dataset.csv",names=['message','label']) #Tabul
ar form data
print('Total instances in the dataset:',msg.shape[0])

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

print('\nThe message and its label of first 5 instances are listed belo
w')
X5, Y5 = X[0:5], msg.label[0:5]
for x, y in zip(X5,Y5):
    print(x,',',y)

# Splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y)
print('\nDataset is split into Training and Testing samples')
print('Total training instances :', xtrain.shape[0])
print('Total testing instances :', xtest.shape[0])

# Output of count vectoriser is a sparse matrix
# CountVectorizer - stands for 'feature extraction'
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain) #Sparse matrix
xtest_dtm = count_vect.transform(xtest)
print('\nTotal features extracted using CountVectorizer:',xtrain_dtm.sh
ape[1])

print('\nFeatures for first 5 training instances are listed below')
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_nam
es())
print(df[0:5])#tabular representation
#print(xtrain_dtm) #Same as above but sparse matrix representation

# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)

print('\nClassstification results of testing samples are given below')
for doc, p in zip(xtest, predicted):
    if p==1:
        pred = 'pos'
    else:
        'neg'
    print('%s -> %s ' % (doc, pred))

#printing accuracy metrics
from sklearn import metrics
print('\nAccuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predi
cted))
```

```
print('Recall :',metrics.recall_score(ytest,predicted), '\nPrecison :',
metrics.precision_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
```

```
OUTPUT:
Total instances in the dataset: 8

The message and its label of first 5 instances are listed below
I love this sandwich , pos
This is an amazing place , pos
I feel very good about these beers , pos
This is my best work , pos
What a great holiday , pos

Dataset is split into Training and Testing samples
Total training instances : 6
Total testing instances : 2

Total features extracted using CountVectorizer: 27

Features for first 5 training instances are listed below
   about  amazing  an  beers  enemy  feel  fun  good  great  have  ...
these  \
0      0        0   0      0      0     0    0     0      1     0  ...
0
1      0        1   1      0      0     0    0     0      0     0  ...
0
2      0        0   0      0      0     0    0     0      0     0  ...
0
3      0        0   0      0      1     0    0     0      0     0  ...
0
4      0        0   0      0      0     0    1     1      0     1  ...
0

   this  to  today  tomorrow  very  we  went  what  will
0     0   0      0         0     0   0     0     1     0
1     1   0      0         0     0   0     0     0     0
2     1   0      0         0     0   0     0     0     0
3     0   1      1         0     0   0     1     0     0
4     0   0      0         1     0   1     0     0     1

[5 rows x 27 columns]

Classstification results of testing samples are given below
This is my best work -> pos
That is a bad locality to stay -> pos

Accuracy metrics
Accuracy of the classifer is 0.5
Recall : 1.0
Precison : 0.5
Confusion matrix
[[0 1]
 [0 1]]
```

PROGRAM 7: Bayesian Network Model to demonstrate Diagnosis of Heart disease

```
pip install pgmpy

import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

heart_disease=pd.read_csv('data7.csv')
print('columns in datasets')
for col in heart_disease.columns:
    print(col)

from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator as MLE
model=BayesianModel([('age','trestbps'),('age','fbs'),('sex','trestbps'),('exang','trestbps'),
                     ('trestbps','heartdisease'),('fbs','heartdisease'),('heartdisease','restecg'),
                     ('heartdisease','thalach'),('heartdisease','chol')])
model.fit(heart_disease,estimator=MLE)

print(model.get_cpds('sex'))

from pgmpy.inference import VariableElimination
HeartDisease_infer = VariableElimination(model)
q = HeartDisease_infer.query(variables = ['heartdisease'],evidence = {'age':29,'sex' : 0,'fbs':1})
print(q)
```

OUTPUT:

```
columns in datasets
age
sex
cp
trestbps
chol
fbs
restecg
thalach
exang
oldpeak
slope
ca
thal
heartdisease

+--------+----------+
| sex(0) | 0.320132 |
+--------+----------+
| sex(1) | 0.679868 |
+--------+----------+

+-----------------+--------------------+
| heartdisease    |   phi(heartdisease) |
```

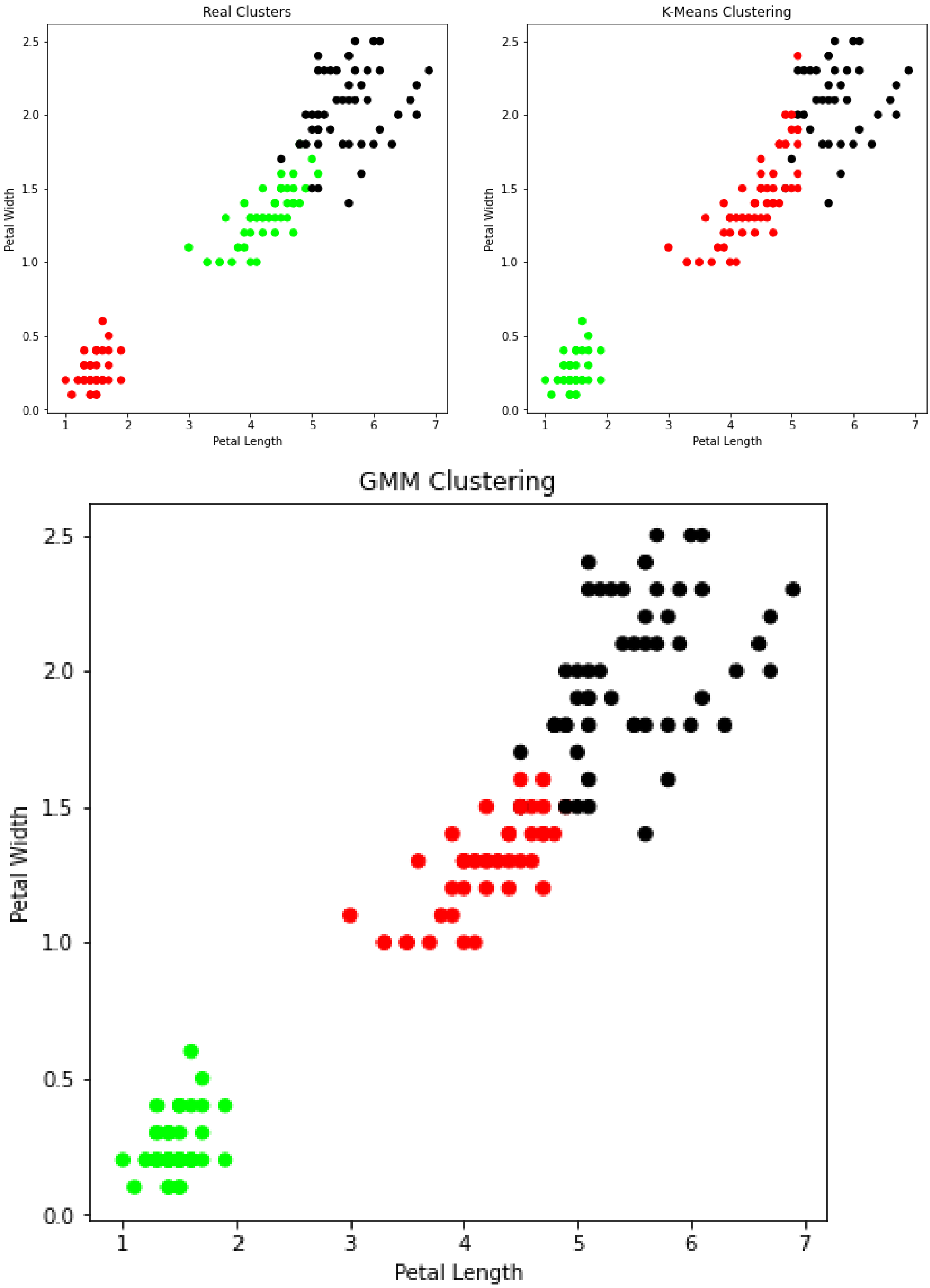
heartdisease(0)	0.3587
heartdisease(1)	0.1220
heartdisease(2)	0.2020
heartdisease(3)	0.2053
heartdisease(4)	0.1120

PROGRAM 8: K-means Clustering

```
import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np


# import some data to play with

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']


# Build the K Means Model

model = KMeans(n_clusters=3)

model.fit(X) # model.labels_ : Gives cluster no for which samples
belongs to


# # Visualise the clustering results

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

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

# Plot the Original Classifications using Petal features

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')

# Plot the Models Classifications

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')
# General EM for GMM
from sklearn import preprocessing
# transform your data such that its distribution will have a
# mean value 0 and standard deviation of 1.
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)

from sklearn.mixture import GaussianMixture
plt.figure(figsize=(14,14))
colormap = np.array(['red', 'lime', 'black'])
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 true labels more closely than the Kmeans.')
```

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



Real Clusters
K-Means Clustering
GMM Clustering
Petal Width
Petal Length

PROGRAM 9: K nearest neighbour

```
# import the required packages
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets


# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")
# Split the data into train and test samples
x_train, x_test, y_train, y_test =
train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)
# Prints Label no. and their names
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))


# Create object of KNN classifier
classifier = KNeighborsClassifier(n_neighbors=1)


# Perform Training
classifier.fit(x_train, y_train)
# Perform testing
y_pred=classifier.predict(x_test)


# Display the results
print("Results of Classification using K-nn with K=1")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), "Actual-label:", str(y_test[r]),
" Predicted-label:",str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

```
OUTPUT:
Iris Data set loaded...
Dataset is split into training and testing...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [5.9 3.2 4.8 1.8] Actual-label: 1  Predicted-label: 2
 Sample: [6.4 3.2 4.5 1.5] Actual-label: 1  Predicted-label: 1
 Sample: [5.7 2.9 4.2 1.3] Actual-label: 1  Predicted-label: 1
 Sample: [5.4 3.4 1.7 0.2] Actual-label: 0  Predicted-label: 0
 Sample: [6.3 2.5 5.  1.9] Actual-label: 2  Predicted-label: 2
 Sample: [5.1 3.5 1.4 0.3] Actual-label: 0  Predicted-label: 0
 Sample: [4.7 3.2 1.3 0.2] Actual-label: 0  Predicted-label: 0
 Sample: [6.7 3.1 4.7 1.5] Actual-label: 1  Predicted-label: 1
 Sample: [6.  2.9 4.5 1.5] Actual-label: 1  Predicted-label: 1
 Sample: [5.  2.3 3.3 1. ] Actual-label: 1  Predicted-label: 1
 Sample: [5.6 2.8 4.9 2. ] Actual-label: 2  Predicted-label: 2
 Sample: [6.2 2.8 4.8 1.8] Actual-label: 2  Predicted-label: 2
 Sample: [6.6 3.  4.4 1.4] Actual-label: 1  Predicted-label: 1
 Sample: [6.3 2.7 4.9 1.8] Actual-label: 2  Predicted-label: 2
 Sample: [6.7 3.1 4.4 1.4] Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
```

PROGRAM 10: Local Weighted Regression

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

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

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    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();


# load data points

data = pd.read_csv("C://Users//Tejasree25//Downloads//CSE SEM 6//ML LAB
DATASET//ML LAB DATASET//Program10dataset.csv")

bill = np.array(data.total_bill) # We use only Bill amount and Tips
data

tip = np.array(data.tip)


mbill = np.mat(bill) # .mat will convert nd array is converted in 2D
array

mtip = np.mat(tip)

m= np.shape(mbill)[1]

one = np.mat(np.ones(m))

X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols


# increase k to get smooth curves

ypred = localWeightRegression(X,mtip,3)

graphPlot(X,ypred)
```

OUTPUT: