ML LAB PROGRAMS

AMULYA
[CSE]

Program: **1. FIND S**

Dataset: 1.csv

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

```
import csv

num_attributes=6
a=[]
print("\n The given training data set \n")
csvfile=open('1.csv','r')
reader=csv.reader(csvfile)
for row in reader:
   a.append(row)
   print(row)
print("The initial values of hypothesis ")
hypothesis=['0']*num_attributes
print(hypothesis)

for j in range(0,num_attributes):
   hypothesis[j]=a[0][j]

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 instance no:",i," the hypothesis is ",hypothesis)
print("The maximally specific hypothesis is ",hypothesis)
```

1output:

```
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 values of hypothesis
['0', '0', '0', '0', '0', '0']
For training instance no:0 the hypothesis is['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
For training instance no:1 the hypothesis is['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For training instance no: 2 the hypothesis is['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For training instance no: 3 the hypothesis is['Sunny', 'Warm', '?', 'Strong', '?', '?']
The maximally specific hypothesis is['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Program: 2. CANDIDATE ELIMINATION ALGORITHM

```
import csv

a=[]
csvfile=open('1.csv','r')
reader=csv.reader(csvfile)
for row in reader:
   a.append(row)
   print(row)

num_attributes=len(a[0])-1
print("Initial hypothesis is ")
S=['0']*num_attributes
G=['?']*num_attributes
print("The most specific : ",S)
print("The most general  : ",G)

for j in range(0,num_attributes):
   S[j]=a[0][j]
print("The candidate algorithm \n")
temp=[]

for i in range(0,len(a)):
   if(a[i][num_attributes]=='Yes'):
      for j in range(0,num_attributes):
         if(a[i][j]!=S[j]):
            S[j]='?'
      for j in range(0,num_attributes):
         for k in range(1,len(temp)):
            if temp[k][j]!='?' and temp[k][j]!=S[j]:
               del temp[k]
      print("For instance {0} the hypothesis is S{0}".format(i+1),S)
      if(len(temp)==0):
         print("For instance {0} the hypothesis is G{0}".format(i+1),G)
      else:
         print("For instance {0} the hypothesis is S{0}".format(i+1),temp)

   if(a[i][num_attributes]=='No'):
      for j in range(0,num_attributes):
         if(S[j]!=a[i][j] and S[j]!='?'):
            G[j]=S[j]
            temp.append(G)
            G=['?']*num_attributes
      print("For instance {0} the hypothesis is S{0}".format(i+1),S)
      print("For instance {0} the hypothesis is G{0}".format(i+1),temp)
```

2output:

```
['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']
Initial hypothesis is
The most specific :  ['0', '0', '0', '0', '0', '0']
The most general  :  ['?', '?', '?', '?', '?', '?']
The candidate algorithm

For instance 1 the hypothesis is S1 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
```

For instance 1 the hypothesis is G1 ['?', '?', '?', '?', '?', '?']
For instance 2 the hypothesis is S2 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For instance 2 the hypothesis is G2 ['?', '?', '?', '?', '?', '?']
For instance 3 the hypothesis is S3 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For instance 3 the hypothesis is G3 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
For instance 4 the hypothesis is S4 ['Sunny', 'Warm', '?', 'Strong', '?', '?']
For instance 4 the hypothesis is S4 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

3Data set: playtennis.csv

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

3output:

Given Play Tennis Data Set:

	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
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong
7	No	Sunny	Mild	High	Weak
8	Yes	Sunny	Cool	Normal	Weak
9	Yes	Rain	Mild	Normal	Weak
10	Yes	Sunny	Mild	Normal	Strong
11	Yes	Overcast	Mild	High	Strong
12	Yes	Overcast	Hot	Normal	Weak
13	No	Rain	Mild	High	Strong

List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']
Gain= [0.2467498197744391, 0.029222565658954647, 0.15183550136234136, 0.04812703040826927]
Best Attribute: Outlook
Gain= [0.01997309402197489, 0.01997309402197489, 0.9709505944546686]
Best Attribute: Wind
Gain= [0.5709505944546686, 0.9709505944546686, 0.01997309402197489]
Best Attribute: Humidity

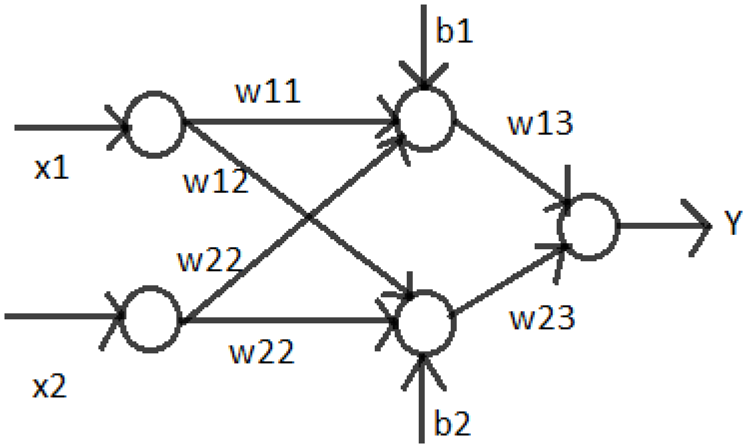
The Resultant Decision Tree is :

{'Outlook': {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}

Program: 3.ID3 ALGORITHM

```
import pandas as pd
from collections import Counter
import math

tennis = pd.read_csv('playtennis.csv')
print("\n Given Play Tennis Data Set:\n\n", tennis)

def entropy(alist):
   c = Counter(x for x in alist)
   instances = len(alist)
   prob = [x / instances for x in c.values()]
   return sum( [-p*math.log(p, 2) for p in prob] )

def information_gain(d, split, target):
   splitting = d.groupby(split)
   n = len(d.index)
   agent = splitting.agg({target : [entropy, lambda x: len(x)/n] })[target] #aggregating
   agent.columns = ['Entropy', 'observations']
   newentropy = sum( agent['Entropy'] * agent['observations'] )
   oldentropy = entropy(d[target])
   return oldentropy - newentropy

def id3(sub, target, a):
   count = Counter(x for x in sub[target])# class of YES /NO
   if len(count) == 1:
      return next(iter(count))  # next input data set, or raises StopIteration when EOF is hit

   else:
      gain = [information_gain(sub, attr, target) for attr in a]
      print("Gain=",gain)
      maximum = gain.index(max(gain))
      best = a[maximum]
      print("Best Attribute:",best)
      tree = {best:{}}
      remaining = [i for i in a if i != best]

      for val, subset in sub.groupby(best):
         subtree = id3(subset,target,remaining)
         tree[best][val] = subtree
      return tree

names = list(tennis.columns)
print("List of Attributes:", names)
names.remove('PlayTennis')
print("Predicting Attributes:", names)

tree = id3(tennis,'PlayTennis',names)
print("\n\nThe Resultant Decision Tree is :\n")
print(tree)
```

Program: 4. BACKPROPOGATION




```
import math
def sigmoid(x):
   y=1/(1+math.exp(-x))
   return y
x1=[0,0,1,1]
x2=[0,1,0,1]
t=[0,1,1,0]
b1=-0.3
w11=0.21
w21=0.15
b2=0.25
w12=-0.4
w22=0.1
b3=-0.4
w13=-0.2
w23=0.3
error=0
iteration=0
train=True
print("weigth are: ")
print("w11 :%4.2f w12:%4.2f w21:%4.2f w22:%4.2f w13:%4.2f w23:%4.2f \n"
%(w11,w12,w21,w22,w13,w23))
while(train):
      for i in range(len(x1)):
            z_in1=b1+x1[i]*w11+x2[i]*w21
            z_in2=b2+x1[i]*w12+x2[i]*w22
            z1=round(sigmoid(z_in1),4)
            z2=round(sigmoid(z_in2),4)

            y_in=b3+z1*w13+z2*w23
            y=round(sigmoid(y_in),4)

            del_k=round((t[i]-y)*y*(1-y),4)
            error=del_k

            w13=round(w13+del_k*z1,4)
            w23=round(w23+del_k*z2,4)
            b3=round(b3+del_k,4)

            del_1=del_k*w13*z1*(1-z1)
            del_2=del_k*w23*z2*(1-z2)

            b1=round(b1+del_1,4)
            w11=round(w11+del_1*x1[i],4)
            w12=round(w12+del_1*x1[i],4)

            b2=round(b2+del_2,4)
            w21=round(w21+del_2*x2[i],4)
            w22=round(w22+del_2*x2[i],4)

            print("iteration: ",iteration)
            print("w11:%5.4f w12:%5.4f w21:%5.4fw22:%5.4f w13:%5.4f w23:%5.4f "%
         (w11,w12,w21,w22,w13,w23))
            print("Error:%5.3f"%del_k)
            iteration=iteration+1
      if(iteration==1000):  train=False
```

4output:(it will display all iterations from 1-999)
iteration: 997
w11:0.8530 w12:0.2430 w21:0.2374 w22:0.1874 w13:-0.2086 w23:0.3359
Error:0.140
iteration: 998
w11:0.8513 w12:0.2413 w21:0.2374 w22:0.1874 w13:-0.1030 w23:0.4420
Error:0.125
iteration: 999
w11:0.8548 w12:0.2448 w21:0.2325 w22:0.1825 w13:-0.2265 w23:0.3187
Error:-0.141

5Dataset:5.csv
6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.627,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.284,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
10,139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
5,166,72,19,175,25.8,0.587,51,1
7,100,0,0,0,30,0.484,32,1

5output:
Size of dataset is: 768
537
{0: [[1.0, 107.0, 68.0, 19.0, 0.0, 26.5, 0.165, 24.0, 0.0], [1.0, 144.0, 82.0, 40.0, 0.0, 41.3, 0.607, 28.0, 0.0], [1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0, 0.0]
{0: [(3.454022988505747, 3.1284989024698904), (110.01724137931035, 26.938498454745453), (67.92528735632185, 18.368785190361336), (19.612068965517242, 15.312369913377424), (68.95689655172414, 105.42637942980888), (30.54080459770115, 7.710567727617014), (0.4458764367816092, 0.31886309966940785), (31.74712643678161, 12.079437732209673)], 1: [(4.64021164021164, 3.7823318201241096), (143.07407407407408, 32.13758346670748), (72.03174603174604, 19.92883742963596), (22.49206349206349, 18.234179691371473), (99.04232804232804, 127.80927573836007), (35.351851851851855, 7.308750166698269), (0.5427301587301587, 0.3832947121639522), (36.43386243386244, 10.813315097901606)]}

Accuracy: 78.78787878787878

Program: 5. NAÏVE BAYESIAN CLASSIFIER

```
import csv
import math
import random
import statistics

def cal_probability(x,mean,stdev):
   exponent=math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
   return(1/(math.sqrt(2*math.pi)*stdev))*exponent
```

```
dataset=[]
dataset_size=0
with open('lab5.csv') as csvfile:
    lines=csv.reader(csvfile)
    for row in lines:
        dataset.append([float(attr) for attr in row])
dataset_size=len(dataset)
print("Size of dataset is: ",dataset_size)

train_size=int(0.7*dataset_size)
print(train_size)
X_train=[]
X_test=dataset.copy()
training_indexes=random.sample(range(dataset_size),train_size)

for i in training_indexes:
    X_train.append(dataset[i])
    X_test.remove(dataset[i])

classes={}
for samples in X_train:
    last=int(samples[-1])
    if last not in classes:
        classes[last]=[]
    classes[last].append(samples)

print(classes)
summaries={}
for classValue,training_data in classes.items():
    summary=[(statistics.mean(attribute),statistics.stdev(attribute)) for attribute in
zip(*training_data)]
    del summary[-1]
    summaries[classValue]=summary

print(summaries)
X_prediction=[]

for i in X_test:
    probabilities={}
    for classValue,classSummary in summaries.items():
        probabilities[classValue]=1
        for index,attr in enumerate(classSummary):
            probabilities[classValue]*=cal_probability(i[index],attr[0],attr[1])

    best_label,best_prob=None,-1
    for classValue,probability in probabilities.items():
        if best_label is None or probability>best_prob:
            best_prob=probability
            best_label=classValue
    X_prediction.append(best_label)

correct=0
for index,key in enumerate(X_test):
    if X_test[index][-1]==X_prediction[index]:
        correct+=1
print("Accuracy: ",correct/(float(len(X_test)))*100)
```

Program: 6 NAÏVE BAYES TEXT CLASSIFIER

```
import pandas as pd
msg=pd.read_csv('6.txt',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)

from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print(xtest.shape)
print(xtrain.shape)
print(ytest.shape)
print(ytrain.shape)

from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print(count_vect.get_feature_names())

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
print(df)


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

from sklearn import metrics
print('Accuracy metrics')
print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('Recall and Precison ')
print(metrics.recall_score(ytest,predicted))
print(metrics.precision_score(ytest,predicted))
```

```
The dimensions of the dataset (18, 2)
0                   I love this sandwich
1               This is an amazing place
2         I feel very good about these beers
3                   This is my best work
4                   What an awesome view
5           I do not like this restaurant
6               I am tired of this stuff
7                 I can't deal with this
8                   He is my sworn enemy
9                    My boss is horrible
10               This is an awesome place
11    I do not like the taste of this juice
12                      I love to dance
13         I am sick and tired of this place
14                   What a great holiday
15           That is a bad locality to stay
16           We will have good fun tomorrow
17          I went to my enemy's house today
```

```
Name: message, dtype: object
0    1
1    1
2    1
3    1
4    1
5    0
6    0
7    0
8    0
9    0
10   1
11   0
12   1
13   0
14   1
15   0
16   1
17   0
Name: labelnum, dtype: int64
(5,)
(13,)
(5,)
(13,)
['about', 'am', 'amazing', 'an', 'awesome', 'bad', 'beers', 'best', 'boss', 'dance', 'do', 'feel', 'good', 'great', 'holiday', 'horrible', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'stay', 'stuff', 'taste', 'that', 'the', 'these', 'this', 'tired', 'to', 'very', 'view', 'what', 'work']
   about  am  amazing  an  awesome  bad  beers  best  boss  dance  ...  that  \
0      0   0        0   0        0    0      0     0     0      0  ...     0
1      0   0        0   0        0    0      0     0     1      0  ...     0
2      0   0        0   0        0    0      0     0     0      1  ...     0
3      0   1        0   0        0    0      0     0     0      0  ...     0
4      0   0        0   0        0    0      0     0     0      0  ...     0
5      0   0        0   1        1    0      0     0     0      0  ...     0
6      0   0        0   0        0    1      0     0     0      0  ...     1
7      0   0        0   1        1    0      0     0     0      0  ...     0
8      0   0        0   0        0    0      0     0     0      0  ...     0
9      0   0        1   1        0    0      0     0     0      0  ...     0
10     0   0        0   0        0    0      0     0     0      0  ...     0
11     0   0        0   0        0    0      0     1     0      0  ...     0
12     1   0        0   0        0    0      1     0     0      0  ...     0
[13 rows x 40 columns]
Accuracy metrics
Accuracy of the classifer is 0.4
Confusion matrix
[[1 3]
 [0 1]]
Recall and Precison
1.0
0.25
```

Program: **7. BAYESIAN NETWORK**

```
import pandas as pd
col=['Age','Gender','FamilyHist','Diet','LifeStyle','Cholesterol','HeartDisease']
data = pd.read_csv('lab7.csv',names =col )
print(data)

#encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
for i in range(len(col)):
   data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])

#spliting data
X = data.iloc[:,0:6]
y = data.iloc[:,-1]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

#prediction
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)

#confusion mtx output
from sklearn.metrics import confusion_matrix
print('Confusion matrix',confusion_matrix(y_test, y_pred))
```

Dataset:

SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
SuperSeniorCitizen	Female	Yes	Medium	Sedetary	High	Yes
SeniorCitizen	Male	No	High	Moderate	BorderLine	Yes
Teen	Male	Yes	Medium	Sedetary	Normal	No
Youth	Female	Yes	High	Athlete	Normal	No
MiddleAged	Male	Yes	Medium	Active	High	Yes
Teen	Male	Yes	High	Moderate	High	Yes
SuperSeniorCitizen	Male	Yes	Medium	Sedetary	High	Yes
Youth	Female	Yes	High	Athlete	Normal	No
SeniorCitizen	Female	No	High	Athlete	Normal	Yes
Teen	Female	No	Medium	Moderate	High	Yes
Teen	Male	Yes	Medium	Sedetary	Normal	No
MiddleAged	Female	No	High	Athlete	High	No
MiddleAged	Male	Yes	Medium	Active	High	Yes
Youth	Female	Yes	High	Athlete	BorderLine	No
SuperSeniorCitizen	Male	Yes	High	Athlete	Normal	Yes
SeniorCitizen	Female	No	Medium	Moderate	BorderLine	Yes
Youth	Female	Yes	Medium	Athlete	BorderLine	No
Teen	Male	Yes	Medium	Sedetary	Normal	No

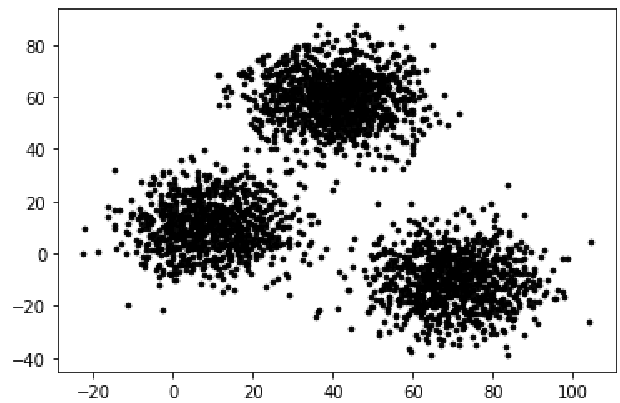
```
Confusion matrix [[0 0]
                  [3 1]]
```

Program: 8. EM ALGORITHM

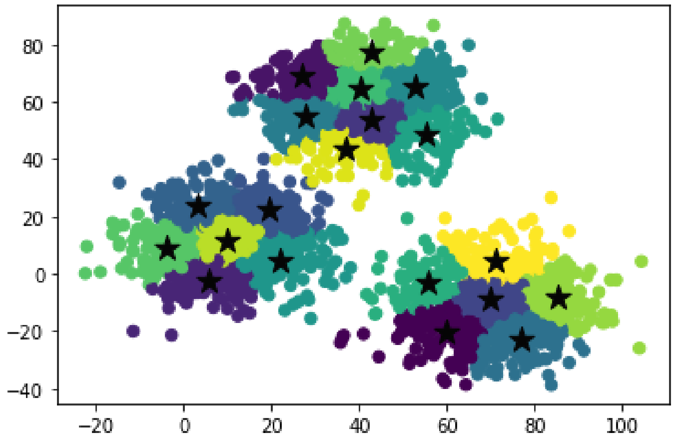
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
data = pd.read_csv('lab8.csv')
print("Input Data and Shape")
print(data.shape)
data.head()

f1 = data['V1'].values
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))

print("X   ", X)
print('Graph for whole dataset')
plt.scatter(f1, f2, c='black', s=7)
plt.show()

kmeans = KMeans(20, random_state=0)
labels = kmeans.fit(X).predict(X)
print("labels    ",labels)
centroids = kmeans.cluster_centers_
print("centroids    ",centroids)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');
print('Graph using Kmeans Algorithm')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='#050505')
plt.show()

gmm = GaussianMixture(n_components=3).fit(X)
labels = gmm.predict(X)

probs = gmm.predict_proba(X)
size = 10 * probs.max(1) ** 3
print('Graph using EM Algorithm')

plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis');
plt.show()
```

OUTPUT:

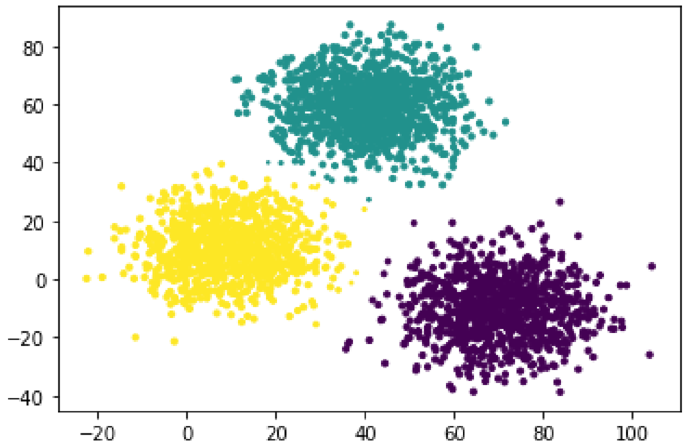
```
Input Data and Shape
(3000, 3)
X    [[  2.072345  -3.241693][ 17.93671   15.78481 ][  1.083576   7.319176]...
 [ 64.46532  -10.50136 ][ 90.72282  -12.25584 ][ 64.87976  -24.87731 ]]
Graph for whole dataset
```



```
labels      [ 2  5 14 ...  4 16  0]
centroids     [[ 59.83204156 -20.27127019]
 [ 26.93926814  68.72877415]
 [  5.74728456  -2.4354335 ]
 [ 42.74508801  53.78669448]
 [ 69.93697849  -8.99255106]
 [ 19.32058349  22.32585954]
 [  3.32731778  23.630905  ]
 [ 76.820093   -23.03153657]
 [ 27.80251033  54.98355311]
 [ 52.85959994  65.33275606]
 [ 22.0826464    4.72511417]
 [ 55.18393576  48.32773467]
 [ 55.89985798  -3.10396622]
 [ 40.09743894  64.23009528]
 [ -4.04689718   8.812598  ]
 [ 42.75426718  77.03129218]
 [ 85.39067866  -8.33454658]
 [  9.89401653  11.85203706]
 [ 37.08384976  43.23678776]
 [ 71.10416952   4.2786267 ]]
Graph using Kmeans Algorithm
```




```
Graph using EM Algorithm
```



Program: 9.K-NEAREST NEIGHBOUR

```
import numpy as np
from sklearn.datasets import load_iris
iris=load_iris()

x=iris.data
y=iris.target
print(x[:5],y[:5])

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest =train_test_split(x,y,test_size=0.4,random_state=1)

print(iris.data.shape)

print(len(xtrain))
print(len(ytest))

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=1)
knn.fit(xtrain,ytrain)
pred=knn.predict(xtest)

from sklearn import metrics
print("Accuracy",metrics.accuracy_score(ytest,pred))
print(iris.target_names[2])
ytestn=[iris.target_names[i] for i in ytest]
predn=[iris.target_names[i] for i in pred]

print("  predicted     Actual")
for i in range(len(pred)):
   print(i,"   ",predn[i],"   ",ytestn[i])
```

OUTPUT:

```
[[5.1 3.5 1.4 0.2]
[4.9 3. 1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5. 3.6 1.4 0.2]] [0 0 0 0 0]
 (150, 4)
 90 60
Accuracy 0.9666666666666667
virginica
  predicted Actual
 0 setosa setosa
1 versicolor versicolor
2 versicolor versicolor
3 setosa setosa
4 virginica virginica
5 virginica versicolor
6 virginica virginica
7 setosa setosa
8 setosa setosa
9 virginica virginica
10 versicolor versicolor
```

Program: 10. LOCALLY WEIGHTED REGRESSION ALGORITHM

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
tou = 0.5
data=pd.read_csv("lab10.csv")
X_train = np.array(data.total_bill)
print(X_train)
X_train = X_train[:, np.newaxis]
print(len(X_train))
y_train = np.array(data.tip)

X_test = np.array([i /10 for i in range(500)])
X_test = X_test[:, np.newaxis]

y_test = []

count = 0
for r in range(len(X_test)):
    wts = np.exp(-np.sum((X_train - X_test[r]) ** 2, axis=1) / (2 * tou ** 2))
    W = np.diag(wts)
    factor1 = np.linalg.inv(X_train.T.dot(W).dot(X_train))
    parameters = factor1.dot(X_train.T).dot(W).dot(y_train)
    prediction = X_test[r].dot(parameters)
    y_test.append(prediction)
    count += 1
print(len(y_test))
y_test = np.array(y_test)
plt.plot(X_train.squeeze(), y_train, 'o')

plt.plot(X_test.squeeze(), y_test, 'o')
plt.show()
```

DATASET:[245 rows]

total_bill	tip	sex	smoker	day	time	size
16.99	1.01	Female	No	Sun	Dinner	2
10.34	1.66	Male	No	Sun	Dinner	3
21.01	3.5	Male	No	Sun	Dinner	3
23.68	3.31	Male	No	Sun	Dinner	2
24.59	3.61	Female	No	Sun	Dinner	4
25.29	4.71	Male	No	Sun	Dinner	4
8.77	2	Male	No	Sun	Dinner	2
26.88	3.12	Male	No	Sun	Dinner	4
15.04	1.96	Male	No	Sun	Dinner	2

Output