BALLARI INSTITUTE OF TECHNOLOGY AND MANAGEMENT BALLARI

Department of Artificial Intelligence and Machine Learning

ML LAB MANUAL

Subject Code : (22AIL54)

5TH SEMESTER

SEC - A, B, C

FOR THE YEAR(2024-2025)

1) Illustrate and Demonstrate the working model and principle of Find-S algorithm. Program: For a given set of training data examples stored in a .CSV file, implement and demonstrate the Find-S algorithm to output a description of the set of all hypotheses consistent with the training examples

```
import pandas as pd
import numpy as np
data = pd.read_csv('lab1.csv')
data
features = np.array(data)[:,:-1]
features
target = np.array(data)[:,-1]
target

for i, val in enumerate(target):
    if val == 'yes':
        specific_h = features[i].copy()
        break
print(specific_h)
for i, val in enumerate(features):
    if target[i] == 'yes':
        for x in range(len(specific_h)):
            if val[x] != specific_h[x]:
                specific_h[x] = '?'
print(specific_h)
```

Output:

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

Dataset:

```
sky,temp,humidity,wind,water,forecast,enjoysport
sunny,warm,normal,strong,warm,same,yes
sunny,warm,high,strong,warm,same,yes
rainy,cold,high,stong,warm,change,no
sunny,warm,high,strong,cold,change,yes
```

2) Demonstrate the working model and principle of candidate elimination algorithm. Program: 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('lab1.csv')
features = np.array(data)[:,:-1]
target = np.array(data)[:,-1]
specific_h = features[0].copy()
print("Initialization of specific_h and general_h")
print(specific_h)
general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
print(general_h)
for i, h in enumerate(features):
    #print("for loop starts")
    if target[i] == "yes":
        #print("if instance is positive")
        for x in range(len(specific_h)):
            if h[x] != specific_h[x]:
```

```
                specific_h[x] = '?'
                general_h[x][x] = '?'

    if target[i] == "no":
        #print("if instance is negative")
        for x in range(len(specific_h)):
            if h[x] != specific_h[x]:
                general_h[x][x] = specific_h[x]
            else:
                general_h[x][x] = '?'
print(specific_h,"\n")
print(general_h,"\n")
indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?','?']]
for i in indices:
    general_h.remove(['?', '?', '?', '?', '?','?'])
print("\nFinal Specific_h:", specific_h, sep="\n")
print("Final General_h:", general_h, sep="\n")
```

Output :

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

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
 '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


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

Dataset :

```
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
```

3) To construct the Decision tree using the training data sets under supervised learning concept. Program: 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
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']
   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("\n Gain=",gain)
      maximum = gain.index(max(gain))
      best = a[maximum]
      print("\nBest Attribute:",best)
      tree = {best:{}}

names = list(tennis.columns)
print("\nList of Attributes:", names)
names.remove('PlayTennis')
print("\nPredicting Attributes:", names)

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

Output:

Given Play Tennis Data Set:

	Outlook	Temperature	Humidity	Wind	PlayTennis
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
13   Rain   Mild   High Strong   No

List of Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']

Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']

Gain= [0.2467498197744391, 0.029222565658954647, 0.15183550136234136, 0.048127030408 26927]

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'}}}}

Dataset :
day outlook temp humidity wind play
D1 sunny   hot high   weak no
D2 sunny   hot high   strong no
D3 overcast mild high   weak   yes
D4 rain   mild high   weak   yes
D5 rain   cool normal weak   yes
D6 rain   cool normal strong no
D7 overcast cool normal strong yes
D8 sunny   mild high   weak   no
D9 sunny   cool normal weak   yes
D10 rain   mild normal strong yes
D11 sunny   mild normal strong yes
D12 overcast mild high   strong yes
D13 overcast hot normal weak   yes
D14 rain   mild high   strong no

4) To understand the working principle of Artificial Neural network with feed forward and feed backward principle. Program: Build an Artificial Neural Network by implementing the Back propagation 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
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1

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):
    hinp1=np.dot(X,wh)
    hinp=hinp1+bh
    hlayer_act=sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp=outinp1+bout
    output=sigmoid(outinp)

    E0=y-output
    outgrad=derivatives_sigmoid(output)
    d_output=E0*outgrad
    EH=d_output.dot(wout.T)
    hiddengrad=derivatives_sigmoid(hlayer_act)
    d_hiddenlayer=EH*hiddengrad
    wout+=hlayer_act.T.dot(d_output)*lr
print("Input:\n"+str(X))
print("Actual Output:\n"+str(y))
print("Predicted Output:\n",output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89462043]
 [0.88465226]
 [0.89071107]]
```

5) Demonstrate the text classifier using Naïve bayes classifier algorithm. Program: Write a program to implement the naive 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
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics

data=pd.read_csv('textdata.csv',names=['message','label'])
print('The dataset is',data)
print('The dimensions of the dataset',data.shape)
data['labelnum']=data.label.map({'pos':1,'neg':0})
X=data.message
y=data.labelnum
print(X)
print(y)
vectorizer = TfidfVectorizer()
data = vectorizer.fit_transform(X)
print('\n the Features of dataset:\n')
df=pd.DataFrame(data.toarray(),columns=vectorizer.get_feature_names_out())
df.head()
print('\n Train Test Split')
xtrain,xtest,ytrain,ytest = train_test_split(data,y,test_size=0.3,random_state=42)
print('\n the total number of training data:',ytrain.shape)
print('\n the total number of test data:',ytest.shape)
clf=MultinomialNB().fit(xtrain,ytrain)
predict=clf.predict(xtest)
predicted=clf.predict(xtest)
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))
print('\nConfusion Matrix is\n',metrics.confusion_matrix(ytest,predicted))
print('\n classification report is\n',metrics.classification_report(ytest,predicted))
print('\n Value of precision is\n',metrics.precision_score(ytest,predicted))
print('\n Value of recall is\n',metrics.recall_score(ytest,predicted))
```

Output:

```
The dataset is                                  message label
0                        i love sandwitch   pos
1                  this is an amazing place   pos
2      i feel very good about these beers   pos
3                      this is my best work   pos
4                      what an awesome view   pos
5             i do not like this restraunt   neg
6                  i am tired of this stuff   neg
7                    i can't deal with this   neg
8                      he is my sworn enemy   neg
9                       my boss is horrible   neg
10                 this is an awesome place   pos
11  i do not like the taste of this juice   neg
12                          i love to dance   pos
13        i am sick and tired of this place   neg
14                     what a great holiday   pos
15            that is bad locality to stay   neg
16          we will have good fun tommorrow   pos
17         i went to my enemy's house today   neg
The dimensions of the dataset (18, 2)
```

```
0                          i love sandwitch
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 restraunt
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 bad locality to stay
16           we will have good fun tommorrow
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

 the Features of dataset:


 Train Test Split

 the total number of training data: (12,)

 the total number of test data: (6,)

 Accuracy of the classifier is 0.8333333333333334

Confusion Matrix is
 [[3 0]
 [1 2]]

 classification report is
              precision    recall  f1-score   support
```

```
           0       0.75      1.00      0.86         3
           1       1.00      0.67      0.80         3

    accuracy                           0.83         6
   macro avg       0.88      0.83      0.83         6
weighted avg       0.88      0.83      0.83         6


 Value of precision is
 1.0

 Value of recall is
 0.6666666666666666
```

Dataset :

```
i love sandwitch,pos
this is an amazing place,pos
i feel very good about these beers,pos
this is my best work,pos
what an awesome view,pos
i do not like this restraunt,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 bad locality to stay,neg
we will have good fun tommorrow,pos
i went to my enemy's house today,neg
```

6) Demonstrate and Analyse the results sets obtained from Bayesian belief network Principle. Program: Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Python ML library classes/API.

```
import pandas as pd
col=['Age','Gender','Familylist','Diet','LifeStyle','Cholesterol','HeartDisease']
data = pd.read_csv('heart.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))
```

Output:

```
Age  Gender Familylist    Diet LifeStyle Cholesterol  \
0   SuperSeniorCitizen    Male       Yes  Medium  Sedetary       High
1   SuperSeniorCitizen  Female       Yes  Medium  Sedetary       High
2       SeniorCitizen    Male        No    High  Moderate  BorderLine
3                Teen    Male       Yes  Medium  Sedetary      Normal
4               Youth  Female       Yes    High   Athlete      Normal
5          MiddleAged    Male       Yes  Medium    Active       High
6                Teen    Male       Yes    High  Moderate       High
7   SuperSeniorCitizen    Male       Yes  Medium  Sedetary       High
8               Youth  Female       Yes    High   Athlete      Normal
9       SeniorCitizen  Female        No    High   Athlete      Normal
10               Teen  Female        No  Medium  Moderate       High
11               Teen    Male       Yes  Medium  Sedetary      Normal
12         MiddleAged  Female        No    High   Athlete       High
13         MiddleAged    Male       Yes  Medium    Active       High
14              Youth  Female       Yes    High   Athlete  BorderLine
15  SuperSeniorCitizen    Male       Yes    High   Athlete      Normal
16      SeniorCitizen  Female        No  Medium  Moderate  BorderLine
17              Youth  Female       Yes  Medium   Athlete  BorderLine
18               Teen    Male       Yes  Medium  Sedetary      Normal

   HeartDisease
0           Yes
1           Yes
2           Yes
3            No
4            No
5           Yes
6           Yes
7           Yes
8            No
9           Yes
10          Yes
11           No
12           No
13          Yes
14           No
15          Yes
16          Yes
17           No
18           No
Confusion matrix:
[0  1]
[2  3]
```

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

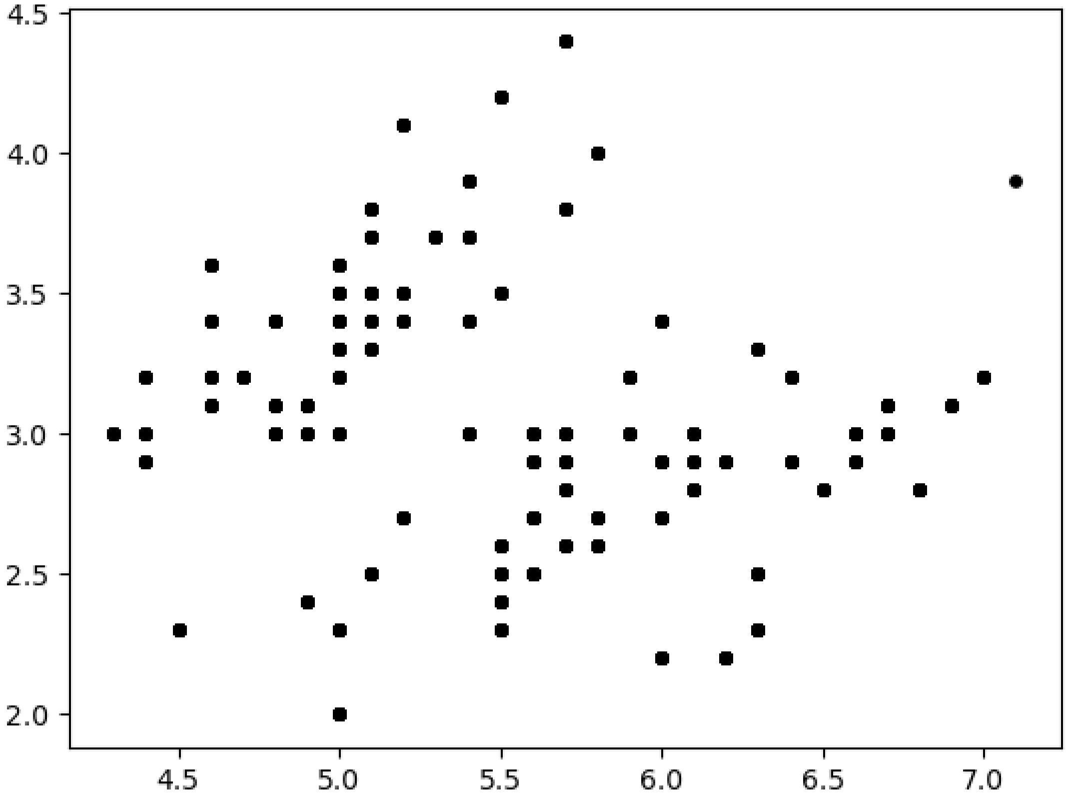
7) Implement and demonstrate the working model of K-means clustering algorithm with Expectation Maximization Concept. Program: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Python ML library classes/API in the program.

```
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('knnvsempgm2.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=15)
plt.show()

kmeans = KMeans(10, random_state=42)
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');
```

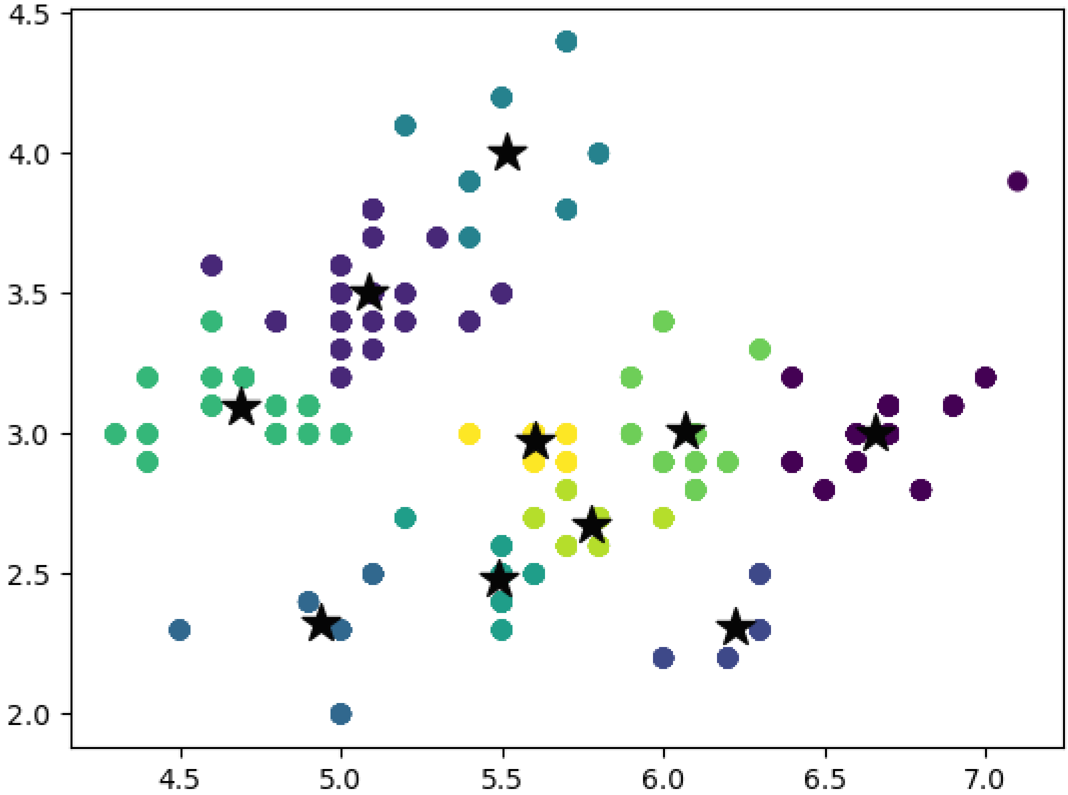
plt.show()

Output:

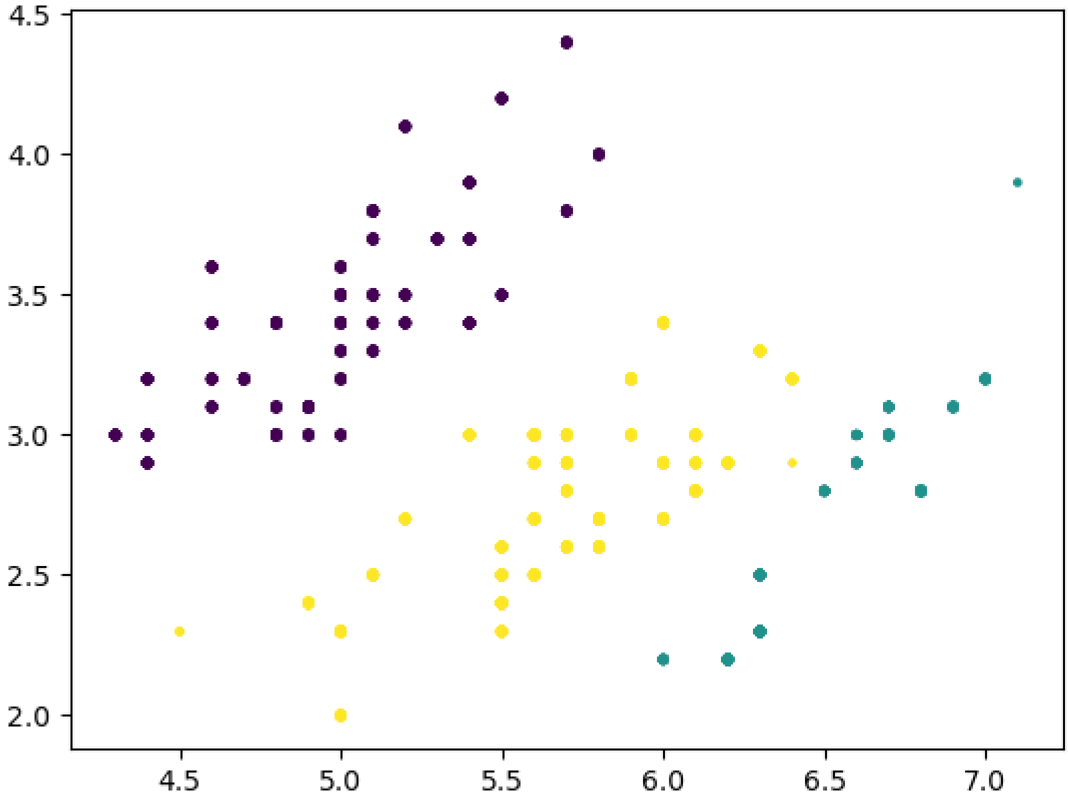
```
Input Data and Shape
(1261, 2)
X  [[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 ...
 [6.2 2.9]
 [5.1 2.5]
 [7.1 3.9]]
Graph for whole dataset
```




```
labels [1 6 6 ... 7 3 0]
centroids  [[6.6602649  3.00198675]
 [5.08333333 3.5       ]
 [6.225      2.3125    ]
 [4.93636364 2.32424242]
 [5.5125     4.        ]
 [5.48965517 2.48448276]
 [4.68823529 3.09411765]
 [6.06538462 3.01282051]
 [5.77627119 2.6779661 ]
 [5.6        2.972     ]]
Graph using Kmeans Algorithm
```



Graph using EM Algorithm



Dataset :

V1,V2
5.1,3.5
4.9,3.0
4.7,3.2
4.6,3.1
5.0,3.6
5.4,3.9
4.6,3.4
5.0,3.4
4.4,2.9
4.9,3.1
5.4,3.7
4.8,3.4
4.8,3.0
4.3,3.0
5.8,4.0
5.7,4.4
5.4,3.9
5.1,3.5
5.7,3.8
5.1,3.8
5.4,3.4
5.1,3.7
4.6,3.6
5.1,3.3
4.8,3.4

8) Demonstrate and analyse the results of classification based on KNN Algorithm. Program: 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.

```
import csv
import random
import math
import operator
def loadDataset(filename,split,trainingSet=[],testSet=[]):
    with open(filename) as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
```

```
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

def main():
# prepare data
    trainingSet=[]
    testSet=[]
    split = 0.67
    loadDataset('iris_data.csv', split, trainingSet, testSet)
    print ('\n Number of Training data: ' + (repr(len(trainingSet))))
    print (' Number of Test Data: ' + (repr(len(testSet))))
# generate predictions
    predictions=[]
    k = 3
    print('\n The predictions are: ')
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print(' predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
    accuracy = getAccuracy(testSet, predictions)
    print('\n The Accuracy is: ' + repr(accuracy) + '%')

main()
```

Output:

```
 Number of Training data: 93
 Number of Test Data: 56

 The predictions are:
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
 predicted='Iris-setosa', actual='Iris-setosa'
```

```
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-versicolor', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'

The Accuracy is: 96.42857142857143%
```

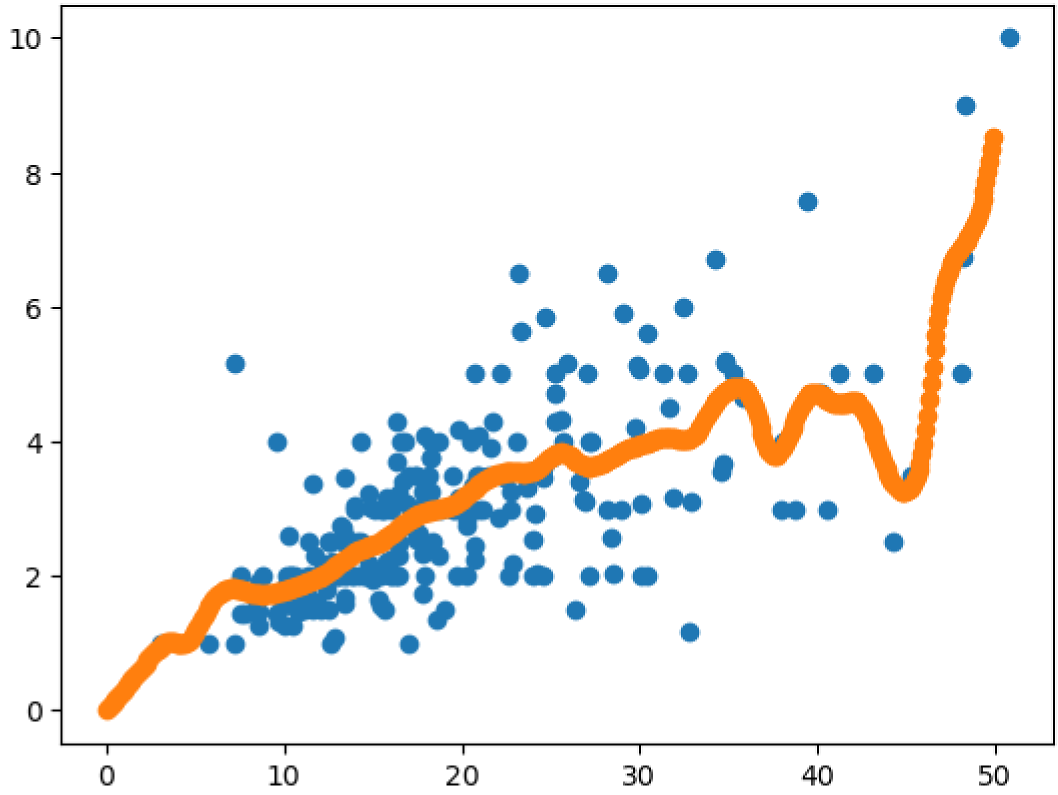
9) Understand and analyse the concept of Regression algorithm techniques. Program: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import pandas as pd
tou = 1
data=pd.read_csv("tips.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)) #factor=XT.W.X
    parameters = factor1.dot(X_train.T).dot(W).dot(y_train) #parameters=factor.XT.W.Y
    prediction = X_test[r].dot(parameters) #X.Theta
    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()
```

Output:

```
[16.99 10.34 21.01 23.68 24.59 25.29  8.77 26.88 15.04 14.78 10.27 35.26
 15.42 18.43 14.83 21.58 10.33 16.29 16.97 20.65 17.92 20.29 15.77 39.42
 19.82 17.81 13.37 12.69 21.7  19.65  9.55 18.35 15.06 20.69 17.78 24.06
 16.31 16.93 18.69 31.27 16.04 17.46 13.94  9.68 30.4  18.29 22.23 32.4
 28.55 18.04 12.54 10.29 34.81  9.94 25.56 19.49 38.01 26.41 11.24 48.27
 20.29 13.81 11.02 18.29 17.59 20.08 16.45  3.07 20.23 15.01 12.02 17.07
 26.86 25.28 14.73 10.51 17.92 27.2  22.76 17.29 19.44 16.66 10.07 32.68
 15.98 34.83 13.03 18.28 24.71 21.16 28.97 22.49  5.75 16.32 22.75 40.17
 27.28 12.03 21.01 12.46 11.35 15.38 44.3  22.42 20.92 15.36 20.49 25.21
 18.24 14.31 14.    7.25 38.07 23.95 25.71 17.31 29.93 10.65 12.43 24.08
 11.69 13.42 14.26 15.95 12.48 29.8   8.52 14.52 11.38 22.82 19.08 20.27
 11.17 12.26 18.26  8.51 10.33 14.15 16.   13.16 17.47 34.3  41.19 27.05
 16.43  8.35 18.64 11.87  9.78  7.51 14.07 13.13 17.26 24.55 19.77 29.85
 48.17 25.   13.39 16.49 21.5  12.66 16.21 13.81 17.51 24.52 20.76 31.71
 10.59 10.63 50.81 15.81  7.25 31.85 16.82 32.9  17.89 14.48  9.6  34.63
 34.65 23.33 45.35 23.17 40.55 20.69 20.9  30.46 18.15 23.1  15.69 19.81
 28.44 15.48 16.58  7.56 10.34 43.11 13.   13.51 18.71 12.74 13.   16.4
 20.53 16.47 26.59 38.73 24.27 12.76 30.06 25.89 48.33 13.27 28.17 12.9
 28.15 11.59  7.74 30.14 12.16 13.42  8.58 15.98 13.42 16.27 10.09 20.45
 13.28 22.12 24.01 15.69 11.61 10.77 15.53 10.07 12.6  32.83 35.83 29.03
 27.18 22.67 17.82 18.78]
244
```

500



Dataset :

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
14.78,3.23,Male,No,Sun,Dinner,2
10.27,1.71,Male,No,Sun,Dinner,2
35.26,5,Female,No,Sun,Dinner,4
15.42,1.57,Male,No,Sun,Dinner,2
18.43,3,Male,No,Sun,Dinner,4
14.83,3.02,Female,No,Sun,Dinner,2
21.58,3.92,Male,No,Sun,Dinner,2
10.33,1.67,Female,No,Sun,Dinner,3
16.29,3.71,Male,No,Sun,Dinner,3
16.97,3.5,Female,No,Sun,Dinner,3
20.65,3.35,Male,No,Sat,Dinner,3
17.92,4.08,Male,No,Sat,Dinner,2
20.29,2.75,Female,No,Sat,Dinner,2
15.77,2.23,Female,No,Sat,Dinner,2
39.42,7.58,Male,No,Sat,Dinner,4
19.82,3.18,Male,No,Sat,Dinner,2
17.81,2.34,Male,No,Sat,Dinner,4

13.37,2,Male,No,Sat,Dinner,2
12.69,2,Male,No,Sat,Dinner,2
21.7,4.3,Male,No,Sat,Dinner,2
19.65,3,Female,No,Sat,Dinner,2
9.55,1.45,Male,No,Sat,Dinner,2
18.35,2.5,Male,No,Sat,Dinner,4
15.06,3,Female,No,Sat,Dinner,2
20.69,2.45,Female,No,Sat,Dinner,4
17.78,3.27,Male,No,Sat,Dinner,2
24.06,3.6,Male,No,Sat,Dinner,3
16.31,2,Male,No,Sat,Dinner,3
16.93,3.07,Female,No,Sat,Dinner,3
18.69,2.31,Male,No,Sat,Dinner,3
31.27,5,Male,No,Sat,Dinner,3
16.04,2.24,Male,No,Sat,Dinner,3
17.46,2.54,Male,No,Sun,Dinner,2
13.94,3.06,Male,No,Sun,Dinner,2
9.68,1.32,Male,No,Sun,Dinner,2
30.4,5.6,Male,No,Sun,Dinner,4
18.29,3,Male,No,Sun,Dinner,2
22.23,5,Male,No,Sun,Dinner,2
32.4,6,Male,No,Sun,Dinner,4
28.55,2.05,Male,No,Sun,Dinner,3
18.04,3,Male,No,Sun,Dinner,2
12.54,2.5,Male,No,Sun,Dinner,2
10.29,2.6,Female,No,Sun,Dinner,2
34.81,5.2,Female,No,Sun,Dinner,4
9.94,1.56,Male,No,Sun,Dinner,2
25.56,4.34,Male,No,Sun,Dinner,4
19.49,3.51,Male,No,Sun,Dinner,2
38.01,3,Male,Yes,Sat,Dinner,4
26.41,1.5,Female,No,Sat,Dinner,2
11.24,1.76,Male,Yes,Sat,Dinner,2
48.27,6.73,Male,No,Sat,Dinner,4
20.29,3.21,Male,Yes,Sat,Dinner,2
13.81,2,Male,Yes,Sat,Dinner,2
11.02,1.98,Male,Yes,Sat,Dinner,2
18.29,3.76,Male,Yes,Sat,Dinner,4
17.59,2.64,Male,No,Sat,Dinner,3
20.08,3.15,Male,No,Sat,Dinner,3
16.45,2.47,Female,No,Sat,Dinner,2
3.07,1,Female,Yes,Sat,Dinner,1
20.23,2.01,Male,No,Sat,Dinner,2
15.01,2.09,Male,Yes,Sat,Dinner,2
12.02,1.97,Male,No,Sat,Dinner,2
17.07,3,Female,No,Sat,Dinner,3
26.86,3.14,Female,Yes,Sat,Dinner,2
25.28,5,Female,Yes,Sat,Dinner,2
14.73,2.2,Female,No,Sat,Dinner,2
10.51,1.25,Male,No,Sat,Dinner,2
17.92,3.08,Male,Yes,Sat,Dinner,2
27.2,4,Male,No,Thur,Lunch,4
22.76,3,Male,No,Thur,Lunch,2
17.29,2.71,Male,No,Thur,Lunch,2
19.44,3,Male,Yes,Thur,Lunch,2
16.66,3.4,Male,No,Thur,Lunch,2
10.07,1.83,Female,No,Thur,Lunch,1
32.68,5,Male,Yes,Thur,Lunch,2
15.98,2.03,Male,No,Thur,Lunch,2
34.83,5.17,Female,No,Thur,Lunch,4

13.03,2,Male,No,Thur,Lunch,2
18.28,4,Male,No,Thur,Lunch,2
24.71,5.85,Male,No,Thur,Lunch,2
21.16,3,Male,No,Thur,Lunch,2
28.97,3,Male,Yes,Fri,Dinner,2
22.49,3.5,Male,No,Fri,Dinner,2
5.75,1,Female,Yes,Fri,Dinner,2
16.32,4.3,Female,Yes,Fri,Dinner,2
22.75,3.25,Female,No,Fri,Dinner,2
40.17,4.73,Male,Yes,Fri,Dinner,4
27.28,4,Male,Yes,Fri,Dinner,2
12.03,1.5,Male,Yes,Fri,Dinner,2
21.01,3,Male,Yes,Fri,Dinner,2
12.46,1.5,Male,No,Fri,Dinner,2
11.35,2.5,Female,Yes,Fri,Dinner,2
15.38,3,Female,Yes,Fri,Dinner,2
44.3,2.5,Female,Yes,Sat,Dinner,3
22.42,3.48,Female,Yes,Sat,Dinner,2
20.92,4.08,Female,No,Sat,Dinner,2
15.36,1.64,Male,Yes,Sat,Dinner,2
20.49,4.06,Male,Yes,Sat,Dinner,2
25.21,4.29,Male,Yes,Sat,Dinner,2
18.24,3.76,Male,No,Sat,Dinner,2
14.31,4,Female,Yes,Sat,Dinner,2
14,3,Male,No,Sat,Dinner,2
7.25,1,Female,No,Sat,Dinner,1
38.07,4,Male,No,Sun,Dinner,3
23.95,2.55,Male,No,Sun,Dinner,2
25.71,4,Female,No,Sun,Dinner,3
17.31,3.5,Female,No,Sun,Dinner,2
29.93,5.07,Male,No,Sun,Dinner,4
10.65,1.5,Female,No,Thur,Lunch,2
12.43,1.8,Female,No,Thur,Lunch,2
24.08,2.92,Female,No,Thur,Lunch,4
11.69,2.31,Male,No,Thur,Lunch,2
13.42,1.68,Female,No,Thur,Lunch,2
14.26,2.5,Male,No,Thur,Lunch,2
15.95,2,Male,No,Thur,Lunch,2
12.48,2.52,Female,No,Thur,Lunch,2
29.8,4.2,Female,No,Thur,Lunch,6
8.52,1.48,Male,No,Thur,Lunch,2
14.52,2,Female,No,Thur,Lunch,2
11.38,2,Female,No,Thur,Lunch,2
22.82,2.18,Male,No,Thur,Lunch,3
19.08,1.5,Male,No,Thur,Lunch,2
20.27,2.83,Female,No,Thur,Lunch,2
11.17,1.5,Female,No,Thur,Lunch,2
12.26,2,Female,No,Thur,Lunch,2
18.26,3.25,Female,No,Thur,Lunch,2
8.51,1.25,Female,No,Thur,Lunch,2
10.33,2,Female,No,Thur,Lunch,2
14.15,2,Female,No,Thur,Lunch,2
16,2,Male,Yes,Thur,Lunch,2
13.16,2.75,Female,No,Thur,Lunch,2
17.47,3.5,Female,No,Thur,Lunch,2
34.3,6.7,Male,No,Thur,Lunch,6
41.19,5,Male,No,Thur,Lunch,5
27.05,5,Female,No,Thur,Lunch,6
16.43,2.3,Female,No,Thur,Lunch,2
8.35,1.5,Female,No,Thur,Lunch,2

18.64,1.36,Female,No,Thur,Lunch,3
11.87,1.63,Female,No,Thur,Lunch,2
9.78,1.73,Male,No,Thur,Lunch,2
7.51,2,Male,No,Thur,Lunch,2
14.07,2.5,Male,No,Sun,Dinner,2
13.13,2,Male,No,Sun,Dinner,2
17.26,2.74,Male,No,Sun,Dinner,3
24.55,2,Male,No,Sun,Dinner,4
19.77,2,Male,No,Sun,Dinner,4
29.85,5.14,Female,No,Sun,Dinner,5
48.17,5,Male,No,Sun,Dinner,6
25,3.75,Female,No,Sun,Dinner,4
13.39,2.61,Female,No,Sun,Dinner,2
16.49,2,Male,No,Sun,Dinner,4
21.5,3.5,Male,No,Sun,Dinner,4
12.66,2.5,Male,No,Sun,Dinner,2
16.21,2,Female,No,Sun,Dinner,3
13.81,2,Male,No,Sun,Dinner,2
17.51,3,Female,Yes,Sun,Dinner,2
24.52,3.48,Male,No,Sun,Dinner,3
20.76,2.24,Male,No,Sun,Dinner,2
31.71,4.5,Male,No,Sun,Dinner,4
10.59,1.61,Female,Yes,Sat,Dinner,2
10.63,2,Female,Yes,Sat,Dinner,2
50.81,10,Male,Yes,Sat,Dinner,3
15.81,3.16,Male,Yes,Sat,Dinner,2
7.25,5.15,Male,Yes,Sun,Dinner,2
31.85,3.18,Male,Yes,Sun,Dinner,2
16.82,4,Male,Yes,Sun,Dinner,2
32.9,3.11,Male,Yes,Sun,Dinner,2
17.89,2,Male,Yes,Sun,Dinner,2
14.48,2,Male,Yes,Sun,Dinner,2
9.6,4,Female,Yes,Sun,Dinner,2
34.63,3.55,Male,Yes,Sun,Dinner,2
34.65,3.68,Male,Yes,Sun,Dinner,4
23.33,5.65,Male,Yes,Sun,Dinner,2
45.35,3.5,Male,Yes,Sun,Dinner,3
23.17,6.5,Male,Yes,Sun,Dinner,4
40.55,3,Male,Yes,Sun,Dinner,2
20.69,5,Male,No,Sun,Dinner,5
20.9,3.5,Female,Yes,Sun,Dinner,3
30.46,2,Male,Yes,Sun,Dinner,5
18.15,3.5,Female,Yes,Sun,Dinner,3
23.1,4,Male,Yes,Sun,Dinner,3
15.69,1.5,Male,Yes,Sun,Dinner,2
19.81,4.19,Female,Yes,Thur,Lunch,2
28.44,2.56,Male,Yes,Thur,Lunch,2
15.48,2.02,Male,Yes,Thur,Lunch,2
16.58,4,Male,Yes,Thur,Lunch,2
7.56,1.44,Male,No,Thur,Lunch,2
10.34,2,Male,Yes,Thur,Lunch,2
43.11,5,Female,Yes,Thur,Lunch,4
13,2,Female,Yes,Thur,Lunch,2
13.51,2,Male,Yes,Thur,Lunch,2
18.71,4,Male,Yes,Thur,Lunch,3
12.74,2.01,Female,Yes,Thur,Lunch,2
13,2,Female,Yes,Thur,Lunch,2
16.4,2.5,Female,Yes,Thur,Lunch,2
20.53,4,Male,Yes,Thur,Lunch,4
16.47,3.23,Female,Yes,Thur,Lunch,3

```
26.59,3.41,Male,Yes,Sat,Dinner,3
38.73,3,Male,Yes,Sat,Dinner,4
24.27,2.03,Male,Yes,Sat,Dinner,2
12.76,2.23,Female,Yes,Sat,Dinner,2
30.06,2,Male,Yes,Sat,Dinner,3
25.89,5.16,Male,Yes,Sat,Dinner,4
48.33,9,Male,No,Sat,Dinner,4
13.27,2.5,Female,Yes,Sat,Dinner,2
28.17,6.5,Female,Yes,Sat,Dinner,3
12.9,1.1,Female,Yes,Sat,Dinner,2
28.15,3,Male,Yes,Sat,Dinner,5
11.59,1.5,Male,Yes,Sat,Dinner,2
7.74,1.44,Male,Yes,Sat,Dinner,2
30.14,3.09,Female,Yes,Sat,Dinner,4
12.16,2.2,Male,Yes,Fri,Lunch,2
13.42,3.48,Female,Yes,Fri,Lunch,2
8.58,1.92,Male,Yes,Fri,Lunch,1
15.98,3,Female,No,Fri,Lunch,3
13.42,1.58,Male,Yes,Fri,Lunch,2
16.27,2.5,Female,Yes,Fri,Lunch,2
10.09,2,Female,Yes,Fri,Lunch,2
20.45,3,Male,No,Sat,Dinner,4
13.28,2.72,Male,No,Sat,Dinner,2
22.12,2.88,Female,Yes,Sat,Dinner,2
24.01,2,Male,Yes,Sat,Dinner,4
15.69,3,Male,Yes,Sat,Dinner,3
11.61,3.39,Male,No,Sat,Dinner,2
10.77,1.47,Male,No,Sat,Dinner,2
15.53,3,Male,Yes,Sat,Dinner,2
10.07,1.25,Male,No,Sat,Dinner,2
12.6,1,Male,Yes,Sat,Dinner,2
32.83,1.17,Male,Yes,Sat,Dinner,2
35.83,4.67,Female,No,Sat,Dinner,3
29.03,5.92,Male,No,Sat,Dinner,3
27.18,2,Female,Yes,Sat,Dinner,2
22.67,2,Male,Yes,Sat,Dinner,2
17.82,1.75,Male,No,Sat,Dinner,2
18.78,3,Female,No,Thur,Dinner,2
```

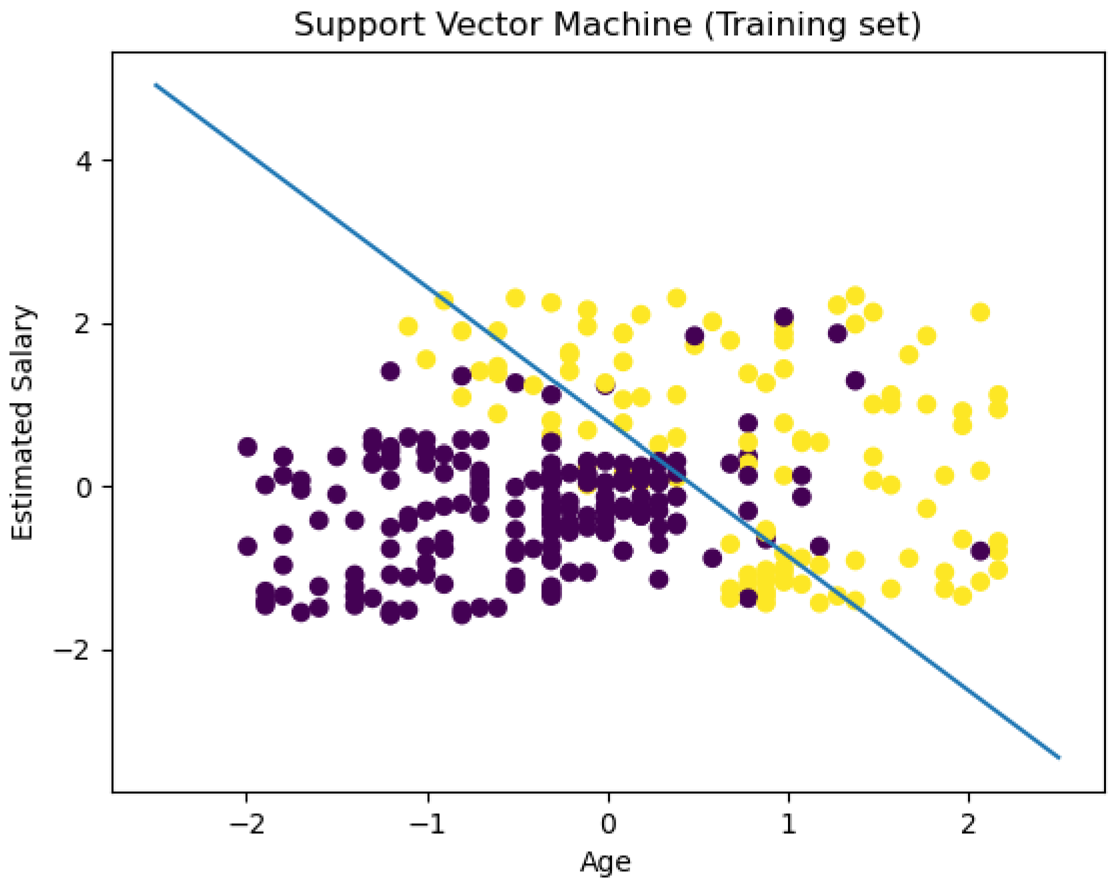
10) Implement and demonstrate classification algorithm using Support vector machine Algorithm.
    Program: Implement and demonstrate the working of SVM algorithm for classification.

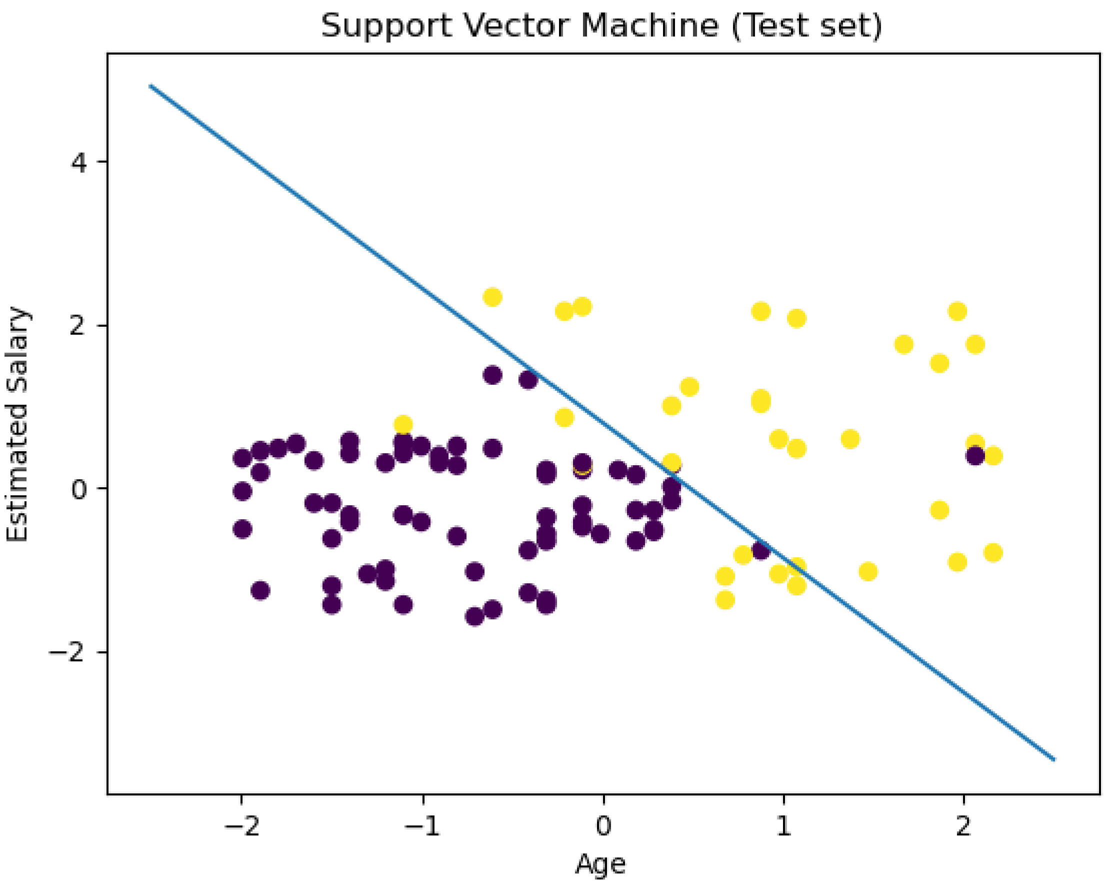
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
datasets = pd.read_csv('10.csv')
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_Train, Y_Train)
Y_Pred = classifier.predict(X_Test)
```

```
from sklearn import metrics
print("Accuracy score ",metrics.accuracy_score(Y_Test, Y_Pred))
plt.scatter(X_Train[:,0], X_Train[:, 1],c=Y_Train)
plt.title('Support Vector Machine (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
w=classifier.coef_[0]
a=-w[0]/w[1]
xx=np.linspace(-2.5,2.5)
yy=a*xx -(classifier.intercept_[0])/w[1]
plt.plot(xx,yy)
plt.show();
plt.scatter(X_Test[:,0], X_Test[:, 1],c=Y_Test)
plt.title('Support Vector Machine (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
w=classifier.coef_[0]
a=-w[0]/w[1]
xx=np.linspace(-2.5,2.5)
yy=a*xx -(classifier.intercept_[0])/w[1]
plt.plot(xx,yy)
plt.show();
```

Output:

Accuracy score 0.9





Dataset :

UserID,Gender,Age,EstimatedSalary,Purchased
15624510,Male,19,19000,0
15810944,Male,35,20000,0
15668575,Female,26,43000,0
15603246,Female,27,57000,0
15804002,Male,19,76000,0
15728773,Male,27,58000,0
15598044,Female,27,84000,0
15694829,Female,32,150000,1
15600575,Male,25,33000,0
15727311,Female,35,65000,0
15570769,Female,26,80000,0
15606274,Female,26,52000,0