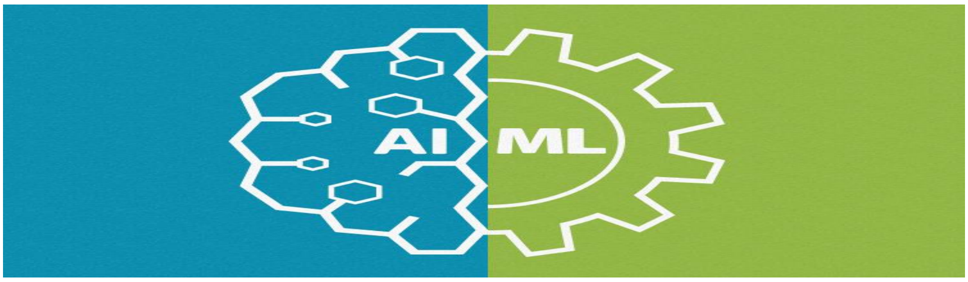
Department of Computer Science Engg.

VII Semester

AI & ML LAB PROGRAMS (18CSL76)



Syllabus

1. Implement A* Search algorithm.

2. Implement AO* Search algorithm.

3. 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.

4. 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.

5. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

6. 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.

7. 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.

8. 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.

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

1. Implement A* Search algorithm.

Source Code:

```
import sys
inf = 99999
g = [
    [0,1,inf,inf,inf,10],
    [1,0,2,1,inf,inf],
    [inf,2,0,inf,5,inf],
    [inf,1,inf,0,3,4],
    [inf,inf,3,0,2],
    [10,inf,inf,4,2,0],
  ]
h = [5,3,4,2,6,0]

src = 0
goal = 5
class obj:
   def __init__(self,cost,path):
      self.cost = cost
      self.path = path
arr = []
new_item = obj(h[src],[src])
arr.append(new_item)
# a* algorithm
while arr:
   cur_item = arr[0]
   cur_node = cur_item.path[-1]
   cur_cost = cur_item.cost
   cur_path = cur_item.path
   for i in range(0,len(h)):
      if g[cur_node][i]!=inf and g[cur_node][i]!=0:
         new_cost = cur_cost - h[cur_node] + h[i] + g[cur_node][i]
         new_path = cur_path.copy()
         new_path.append(i)
         if i==goal:
            print(new_cost)
            print(new_path)
#             sys.exit()
         new_item = obj(new_cost,new_path)
         arr.append(new_item)
   arr.pop(0)
   arr = sorted(arr,key=lambda item:item.cost)
```

OUTPUT

```
17
[0, 2, 3, 4, 6]
18
[0, 2, 4, 6]
21
[0, 1, 4, 6]
25
[0, 1, 5, 6]
```

2. **Implement AO* Search algorithm**

```
import time
import os
def get_node(mark_road,extended):
   temp=[0]
   i=0
   while 1:
      current=temp[i]
      if current not in extended:
         return current
      else:
         for child in mark_road[current]:
            if child not in temp:
               temp.append(child)
         i+=1
def get_current(s,nodes_tree):
   if len(s)==1:
     return s[0]
   for node in s:
      flag=True
      for edge in nodes_tree(node):
         for child_node in edge:
            if child_node in s:
               flag=False
      if flag:
        return node
def get_pre(current,pre,pre_list):
   if current==0:
     return
   for pre_node in pre[current]:
      if pre_node not in pre_list:
         pre_list.append(pre_node)
      get_pre(pre_node,pre,pre_list)
   return
def ans_print(mark_rode,node_tree):
   print("The final connection is as follow:")
   temp=[0]
   while temp:
      time.sleep(1)
      print(f"[{temp[0]}]-----> {mark_rode[temp[0]]}")
      for child in mark_rode[temp[0]]:
         if node_tree[child]!=[[child]]:
            temp.append(child)
      temp.pop(0)
   time.sleep(5)
```

```
    os.system('cls')
    return
def AOstar(node_tree,h_val):
    futility=0xfff
    extended=[]
    choice=[]
    mark_rode={0:None}
    solved={}
    pre={0:[]}
    for i in range(1,9):
        pre[i]=[]
    for i in range(len(nodes_tree)):
        solved[i]=False
    os.system('cls')
    print("The connection process is as follows")
    time.sleep(1)
    while not solved[0] and h_val[0]<futility:
        node=get_node(mark_rode,extended)
        extended.append(node)
        if nodes_tree[node] is None:
            h_val[node]=futility
            continue
        for suc_edge in nodes_tree[node]:
            for suc_node in suc_edge:
                if nodes_tree[suc_node]==[[suc_node]]:
                    solved[suc_node]=True
        s=[node]
        while s:
         current = get_current(s,nodes_tree)
         s.remove(current)
         origen_h=h_val[current]
         origen_s=solved[current]
         min_h=0xfff
         for edge in nodes_tree[current]:
            edge_h=0
            for node in edge:
                edge_h+=h_val[node]+1
            if edge_h<min_h:
                min_h=edge_h
                h_val[current]=min_h
                mark_rode[current]=edge
         if mark_rode[current] not in choice:
            choice.append(mark_rode[current])
            print(f"[{current}]-----{mark_rode[current]}")
            time.sleep(1)
```

```
        for child_node in mark_rode[current]:
          pre[child_node].append(current)
        solved[current]=True
        for node in mark_rode[current]:
          solved[current]=solved[current] and solved[node]
        if origen_s!=solved[current] or origen_h!=h_val[current]:
          pre_list=[]
          if current !=0:
            get_pre(current,pre,pre_list)
          s.extend(pre_list)
    if not solved[0]:
      print("The query failed, the path could not be found!")
    else:
      ans_print(mark_rode,nodes_tree)
    return
if __name__=="__main__":
  nodes_tree={}
  nodes_tree[0]=[[1],[4,5]]
  nodes_tree[1]=[[2],[3]]
  nodes_tree[2]=[[3],[2,5]]
  nodes_tree[3]=[[5,6]]
  nodes_tree[4]=[[5],[8]]
  nodes_tree[5]=[[6],[7,8]]
  nodes_tree[6]=[[7,8]]
  nodes_tree[7]=[[7]]
  nodes_tree[8]=[[8]]
  h_val=[3,2,4,4,1,1,2,0,0]
  AOstar(nodes_tree,h_val)
```

OUTPUT:

```
The connection process is as follows
[0]-----[1]
[1]-----[2]
[0]-----[4, 5]
[4]-----[8]
[5]-----[7, 8]
The final connection is as follow:
[0]-----> [4, 5]
[4]-----> [8]
[5]-----> [7, 8]
```

Program: 3. 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)
```

output:

['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', '?', '?', '?', '?']]

Program: 4.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)
```

Data 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

output:

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: 5. BACKPROPOGATION

Source Code

```python
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) # maximum of X array longitudinally
y = y/100
#Sigmoid Function
def sigmoid (x):
  return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
  return x * (1 - x)
#Variable initialization
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
#weight and bias initialization
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))
#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)
    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO* outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) *lr# dotproduct of nextlayererror and currentlayerop
    # bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    #bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

```python
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)

    EO=y-output
    outgrad=derivatives_sigmoid(output)
    d_output=EO*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:
 [[0.89282584]
 [0.87763012]
[0.89905218]]
```

Program: 6. 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)
```

6Dataset: 6.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
```

6output:

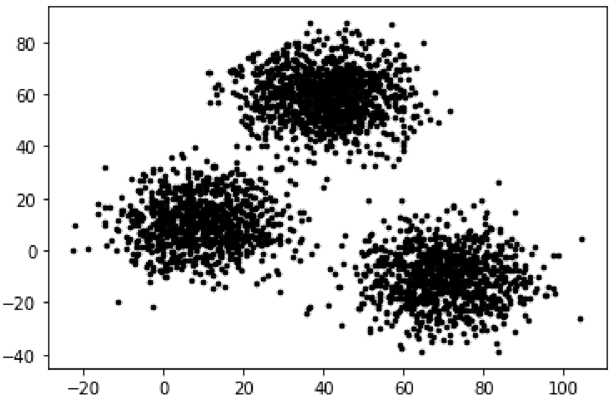
```
Size of dataset is:  768
537
{0: [[1.0, 107.0, 68.0, 19.0, 0.0, 26.5, 0.165, 24.0, 0.0], [1.0, 144.0, 82.0, 40.0, 0.0, 41.3, 0.607, 28.0, 0.0], [1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0, 0.0]
{0: [(3.454022988505747, 3.1284989024698904), (110.01724137931035, 26.938498454745453), (67.92528735632185, 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: 7. 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()
```

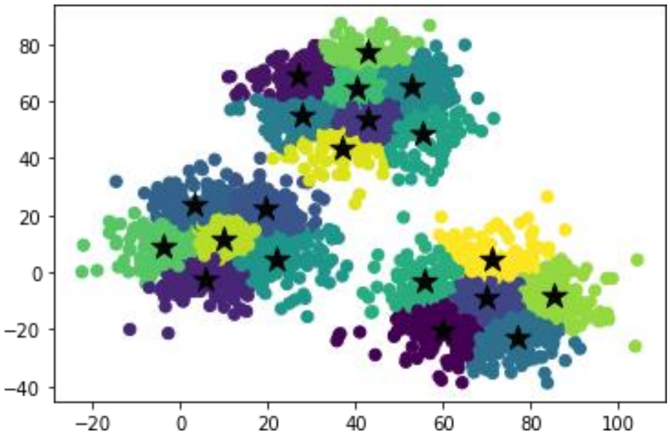
OUTPUT:

```
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
```

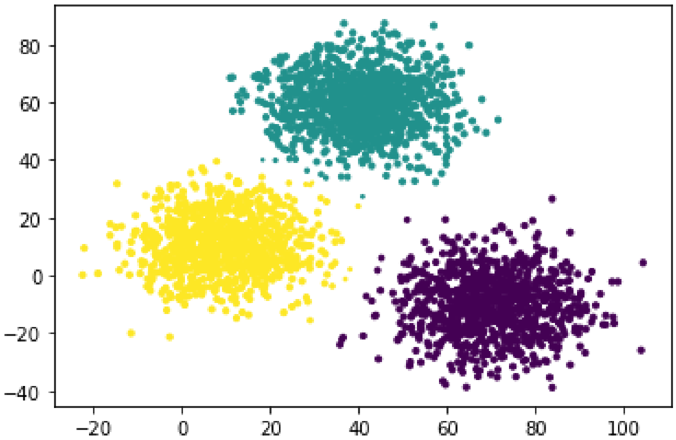


```
labels      [ 2  5 14 ...  4 16  0]
centroids     [[ 59.83204156 -20.27127019]
 [ 26.93926814  68.72877415]
 [  5.74728456  -2.4354335 ]
 [ 42.74508801  53.78669448]
 [ 69.93697849  -8.99255106]
 [ 19.32058349  22.32585954]
 [  3.32731778  23.630905  ]
 [ 76.820093   -23.03153657]
 [ 27.80251033  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: 8.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
```

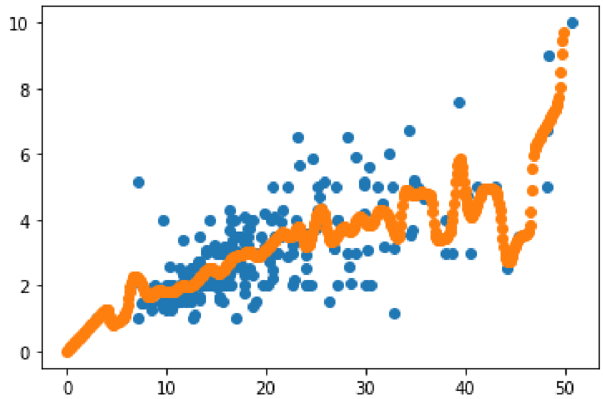
Program: 9. LOCALLY WEIGHTED REGRESSION ALGORITHM

```
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



Design Based Programs

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)
```

output:

```
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', '?', '?']
```