AIML LAB PROGRAMS

1. Implement A Search algorithm.

```
import sys
inf=99999
g=[
   [0,4,3,inf,inf,inf,inf],
   [inf,0,inf,inf,12,5,inf],
   [inf,inf,0,7,10,inf,inf],
   [inf,inf,inf,0,2,inf,inf],
   [inf,inf,inf,inf,0,inf,5],
   [inf,inf,inf,inf,inf,0,16],
   [inf,inf,inf,inf,inf,inf,0],
   ]
h=[14,12,11,6,4,11,0]
src=0
goal=6
class obj:
    def __init__(self,cost,path):
        self.cost=cost
        self.path=path
arr=[]
new_item=obj(h[src],[src])
arr.append(new_item)
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:

```
In [1]: runfile('D:/3BR20CS163/untitled9.py', wdir='D:/3BR20CS163')
17
[0, 2, 3, 4, 6]
18
[0, 2, 4, 6]
21
[0, 1, 4, 6]
25
[0, 1, 5, 6]

In [2]:
```

2. Implement AO* Search algorithm.

```
import os
import time

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_nod in edge:
                if child_nod 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)
    return


def ans_print(mark_rode,node_tree):
    print("The final connection is as follows")
    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')
```

```python
    return
def AOstar(nodes_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 :

```
runfile('D:/3BR20CS163/untitled9.py', wdir='D:/3BR20CS163')
The connection process is as follows
[0]--[1]
[1]--[2]
[0]--[4, 5]
[4]--[8]
[5]--[7, 8]
The final connection is as follows
[0]-->[4, 5]
[4]-->[8]
[5]-->[7, 8]
```

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.

```
import csv
a = []
csvfile = open('pgm3.csv', 'r')
reader = csv.reader(csvfile)
print("Data present in csv file is: ")
for row in reader:
    a.append(row)
    print(row)
num_attributes = len(a[0]) - 1
print("\nInitial 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("\nThe candidate algorithm")
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("\nfor instance {0} the space hypothesis is s{0}\n".format(i +
1), s)
        if (len(temp) == 0):
            print("\nfor instance {0} the hypothesis is G{0}\n".format(i + 1),
g)
        else:
            print("\nfor instance {0} the hypothesis is G{0}\n".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("\nFor instance{0} the hypothesis is s{0}\n".format(i + 1), s)
        print("\nFor instance{0} the hypothesis is g{0}\n".format(i + 1),
temp)
```

Output:

```
In [1]: runfile('C:/Users/Dell/Desktop/AIML Lab Programs/AIML Lab Programs/pgm3.py',
Data present in csv file is:
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cool', '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 space hypothesis is s1
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

for instance 1 the hypothesis is G1
 ['?', '?', '?', '?', '?', '?']

for instance 2 the space hypothesis is s2
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']

for instance 2 the hypothesis is G2
 ['?', '?', '?', '?', '?', '?']

For instance3 the hypothesis is s3
 ['sunny', 'warm', '?', 'strong', 'warm', 'same']
```

```
For instance3 the hypothesis is g3
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

for instance 4 the space hypothesis is s4
 ['sunny', 'warm', '?', 'strong', '?', '?']

for instance 4 the hypothesis is G4
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

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

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

# Read the data
tennis = pd.read_csv('pgm4.csv')
print("\nGiven PlayTennis 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]})
    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])
    if len(count) == 1:
        return next(iter(count))
    else:
        gain = [information_gain(sub, attr, target) for attr in a]
        print("\nGain =", gain)
        maximum = gain.index(max(gain))
        best = a[maximum]
        print("\nBest 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("\nList of Attributes:", names)
names.remove('PlayTennis')
print("\nPredicting Attributes:", names)
```

```
# Convert the 'observations' column to a dictionary
tree = id3(tennis, 'PlayTennis', names)
print("\n\nThe Resultant Decision Tree is:\n")
print(tree)
```

Output:

```
In [1]: runfile('C:/Users/Dell/Desktop/AIML Lab Programs/AIML Lab Programs/pgm4.py', wdir='C:/User

Given PlayTennis 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'}}}}
```

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

```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_neuron=2
hiddenlayer_neuron=3
output_neuron=1
wh=np.random.uniform(size=(inputlayer_neuron,hiddenlayer_neuron))
bh=np.random.uniform(size=(1,hiddenlayer_neuron))
wout=np.random.uniform(size=(hiddenlayer_neuron,output_neuron))
bout=np.random.uniform(size=(1,output_neuron))
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:

```
In [2]: runfile('C:/Users/Dell/Desk
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89389789]
 [0.88302226]
 [0.89281756]]
```

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

```
import 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('pgm6.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)
```

Output:

```
In [3]: runfile('C:/Users/Dell/Desktop/AIML Lab Programs/AIML Lab Programs/pgm6.py', wdir='C:/Users/Dell/Desktop/AIML Lab
Programs/AIML Lab Programs')
Size of dataset is:  768
537
{1: [[9.0, 165.0, 88.0, 0.0, 0.0, 30.4, 0.302, 49.0, 1.0], [9.0, 171.0, 110.0, 24.0, 240.0, 45.4, 0.721, 54.0, 1.0], [2.0, 197.0,
70.0, 45.0, 543.0, 30.5, 0.158, 53.0, 1.0], [10.0, 90.0, 85.0, 32.0, 0.0, 34.9, 0.825, 56.0, 1.0], [7.0, 109.0, 80.0, 31.0, 0.0,
35.9, 1.127, 43.0, 1.0], [10.0, 168.0, 74.0, 0.0, 0.0, 38.0, 0.537, 34.0, 1.0], [8.0, 108.0, 70.0, 0.0, 0.0, 30.5, 0.955, 33.0,
1.0], [0.0, 104.0, 64.0, 37.0, 64.0, 33.6, 0.51, 22.0, 1.0], [9.0, 140.0, 94.0, 0.0, 0.0, 32.7, 0.734, 45.0, 1.0], [0.0, 162.0,
76.0, 56.0, 100.0, 53.2, 0.759, 25.0, 1.0], [7.0, 194.0, 68.0, 28.0, 0.0, 35.9, 0.745, 41.0, 1.0], [14.0, 100.0, 78.0, 25.0,
184.0, 36.6, 0.412, 46.0, 1.0], [4.0, 111.0, 72.0, 47.0, 207.0, 37.1, 1.39, 56.0, 1.0], [4.0, 184.0, 78.0, 39.0, 277.0, 37.0,
0.264, 31.0, 1.0], [1.0, 168.0, 88.0, 29.0, 0.0, 35.0, 0.905, 52.0, 1.0], [9.0, 122.0, 56.0, 0.0, 0.0, 33.3, 1.114, 33.0, 1.0],
[0.0, 123.0, 72.0, 0.0, 0.0, 36.3, 0.258, 52.0, 1.0], [1.0, 181.0, 64.0, 30.0, 180.0, 34.1, 0.328, 38.0, 1.0], [0.0, 121.0, 66.0,
30.0, 165.0, 34.3, 0.203, 33.0, 1.0], [6.0, 190.0, 92.0, 0.0, 0.0, 35.5, 0.278, 66.0, 1.0], [4.0, 146.0, 92.0, 0.0, 0.0, 31.2,
0.539, 61.0, 1.0], [6.0, 119.0, 50.0, 22.0, 176.0, 27.1, 1.318, 33.0, 1.0], [10.0, 111.0, 70.0, 27.0, 0.0, 27.5, 0.141, 40.0,
```

```
27.0, 0.0, 25.0, 0.206, 27.0, 0.0], [0.0, 161.0, 50.0, 0.0, 0.0, 21.9, 0.254, 65.0, 0.0], [9.0, 120.0, 72.0, 22.0, 56.0, 20.8,
0.733, 48.0, 0.0], [4.0, 127.0, 88.0, 11.0, 155.0, 34.5, 0.598, 28.0, 0.0]]}
{1: [(4.73469387755102, 3.7073759340084314), (141.30102040816325, 33.126117962434414), (70.84693877551021, 21.2877005735484),
(22.872448979591837, 17.891383289813692), (103.15816326530613, 141.65049253561904), (35.487755102040815, 7.092629114447057),
(0.5603673469387755, 0.3804677522708769), (36.63265306122449, 10.885782546198449)], 0: [(3.4457478005865103, 3.1315502336095697),
(109.43988269794721, 25.781777513408684), (67.43401759530792, 18.756470679231366), (19.00293255131965, 14.887124032635866),
(65.51026392961877, 91.63487670988607), (30.1574780058651, 7.90323713109066), (0.418782991202346, 0.2850083896667573),
(31.346041055718477, 11.604406763789605)]}
Accuracy: 75.32467532467533
```

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.

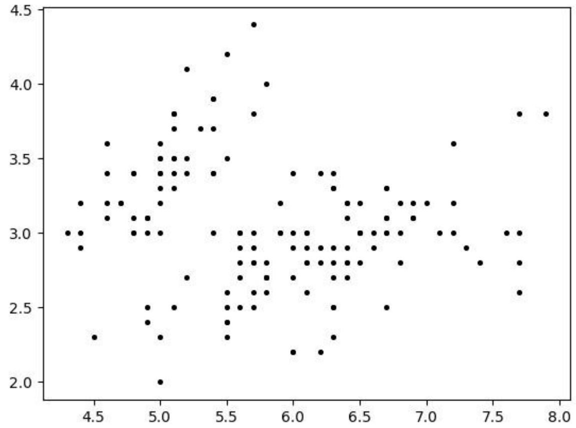
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
data=pd.read_csv("pgm7.csv")
print("Input data and shape")
print(data.shape)
data.head()
f1=data['v1'].values
f2=data['v2'].values
x=np.array(list(zip(f1,f2)))
print("X",x)
print("Graph for which 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("centeroids",centroids)
plt.scatter(x[:,0],x[:,1],c=labels,s=40,cmap="viridis");
print("Grapg 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:

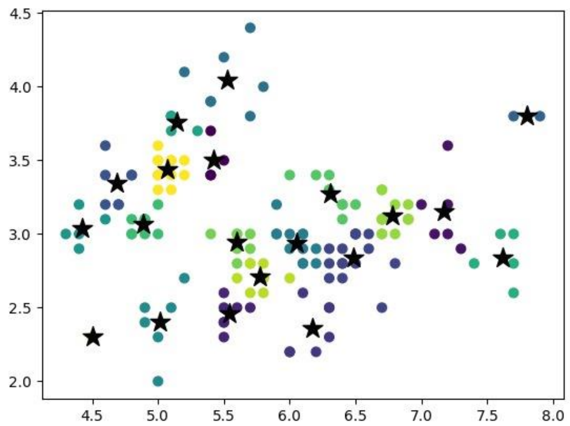
```
input data and shape
(150, 2)
X [[5.1 3.5]
 [4.9 3.  ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]
 [5.4 3.9]
 [4.6 3.4]
 [5.  3.4]
 [4.4 2.9]
 [4.9 3.1]
 [5.4 3.7]
 [4.8 3.4]
 [4.8 3.  ]
 [4.3 3.  ]
 [5.8 4.  ]
 [5.7 4.4]
 [5.4 3.9]
 [5.1 3.5]
 [5.7 3.8]
 [5.1 3.8]
 [5.4 3.4]
 [5.1 3.7]
 [4.6 3.6]
 [5.1 3.3]
```

```
 [6.4 3.1]
 [6.  3. ]
 [6.9 3.1]
 [6.7 3.1]
 [6.9 3.1]
 [5.8 2.7]
 [6.8 3.2]
 [6.7 3.3]
 [6.7 3. ]
 [6.3 2.5]
 [6.5 3. ]
 [6.2 3.4]
 [5.9 3. ]]
graph for whole dataset
```

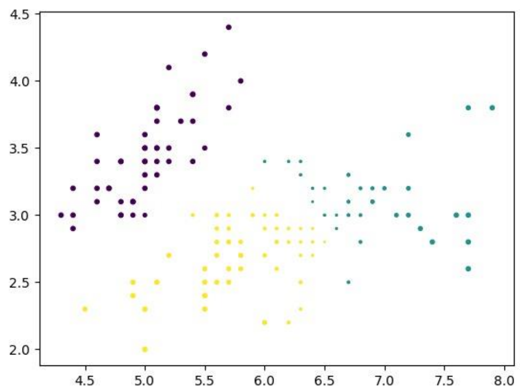
Graph for Whole Dataset



Graph using Kmeans Algorithm



Graph Using EM Algorithm



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.

```
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
11   setosa    setosa
12   virginica    virginica
13   versicolor    versicolor
14   versicolor    versicolor
15   setosa    setosa
16   versicolor    versicolor
17   versicolor    versicolor
18   setosa    setosa
19   setosa    setosa
20   versicolor    versicolor
21   versicolor    versicolor
22   versicolor    versicolor
23   setosa    setosa
24   virginica    virginica
25   versicolor    versicolor
26   setosa    setosa
27   setosa    setosa
28   versicolor    versicolor
29   virginica    virginica
30   versicolor    versicolor
31   virginica    virginica
```

```
32   versicolor    versicolor
33   virginica    virginica
34   virginica    virginica
35   setosa    setosa
36   versicolor    versicolor
37   setosa    setosa
38   versicolor    versicolor
39   virginica    virginica
40   virginica    virginica
41   setosa    setosa
42   versicolor    virginica
43   virginica    virginica
44   versicolor    versicolor
45   virginica    virginica
46   setosa    setosa
47   setosa    setosa
48   setosa    setosa
49   versicolor    versicolor
50   setosa    setosa
51   setosa    setosa
52   virginica    virginica
53   virginica    virginica
54   virginica    virginica
55   virginica    virginica
56   virginica    virginica
57   versicolor    versicolor
58   virginica    virginica
59   versicolor    versicolor
```

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.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
tou=0.5
data=pd.read_csv("pgm9.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()
```

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
```