VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

COURSE TITLE

Submitted by

MUSKAN GUPTA(1BM19CS091)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "LAB COURSE **MACHINE LEARNING**" carried out by **MUSKAN GUPTA (1BM19CS091),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning **- (20CS6PCMAL)** work prescribed for the said degree.

Saritha A. N	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Course Outcome

CO1	Ability to apply the different learning algorithms.
CO2	Ability to analyze the learning techniques for given dataset
CO3	Ability to design a model using machine learning to solve a problem.
CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning Techniques.

1.Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

```
import pandas as pd
import numpy as np
data = pd.read_csv("data - Sheet1.csv")
print(data,"\n")
d = np.array(data)[:,:-1]
print("\n The attributes are: ",d)
target = np.array(data)[:,-1]
def findS(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass
                return specific_hypothesis
print("\n The final hypothesis is:",findS(d,target))
```

```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong   No


 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The final hypothesis is: ['Morning' 'Sunny' '?' 'Yes' '?' '?']
```

2.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('shape.csv')

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

    specific_h = concepts[0].copy()

    print("\nInitialization of specific_h and genearal_h")

    print("\nSpecific Boundary: ", specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

    print("\nGeneric Boundary: ",general_h)


    for i, h in enumerate(concepts):

        print("\nInstance", i+1 , "is ", h)

        if target[i] == "yes":

            print("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("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 Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h
s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

```
Instances are:
 [['big' 'red' 'circle']
 ['small' 'red' 'triangle']
 ['small' 'red' 'circle']
 ['big' 'blue' 'circle']
 ['small' 'blue' 'circle']]

Target Values are:  ['no' 'no' 'yes' 'no' 'yes']

Initialization of specific_h and genearal_h

Specific Boundary:  ['big' 'red' 'circle']

Generic Boundary:  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]

Instance 1 is  ['big' 'red' 'circle']
Instance is Negative
Specific Bundary after  1 Instance is  ['big' 'red' 'circle']
Generic Boundary after  1 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]



Instance 2 is  ['small' 'red' 'triangle']
Instance is Negative
Specific Bundary after  2 Instance is  ['big' 'red' 'circle']
Generic Boundary after  2 Instance is  [['big', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]



Instance 3 is  ['small' 'red' 'circle']
Instance is Positive
Specific Bundary after  3 Instance is  ['?' 'red' 'circle']
Generic Boundary after  3 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]
```


```
Instance 2 is  ['small' 'red' 'triangle']
Instance is Negative
Specific Bundary after  2 Instance is  ['big' 'red' 'circle']
Generic Boundary after  2 Instance is  [['big', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]



Instance 3 is  ['small' 'red' 'circle']
Instance is Positive
Specific Bundary after  3 Instance is  ['?' 'red' 'circle']
Generic Boundary after  3 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', 'circle']]
```


```
Instance 4 is  ['big' 'blue' 'circle']
Instance is Negative
Specific Bundary after  4 Instance is  ['?' 'red' 'circle']
Generic Boundary after  4 Instance is  [['?', '?', '?'], ['?', 'red', '?'], ['?', '?', '?']]



Instance 5 is  ['small' 'blue' 'circle']
Instance is Positive
Specific Bundary after  5 Instance is  ['?' '?' 'circle']
Generic Boundary after  5 Instance is  [['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]


Final Specific_h:
['?' '?' 'circle']
Final General_h:
[['?', '?', '?'], ['?', '?', '?'], ['?', '?', '?']]
```

3.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 math
import csv
def load_csv(filename):
    lines=csv.reader(open(filename,"r"))
    dataset = list(lines)
    headers = dataset.pop(0)
    return dataset,headers

class Node:
    def __init__(self,attribute):
        self.attribute=attribute
        self.children=[]
        self.answer=""

def subtables(data,col,delete):
    dic={}
    coldata=[row[col] for row in data]
    attr=list(set(coldata))

    counts=[0]*len(attr)
    r=len(data)
    c=len(data[0])
    for x in range(len(attr)):
        for y in range(r):
            if data[y][col]==attr[x]:
                counts[x]+=1

    for x in range(len(attr)):
        dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]
        pos=0
        for y in range(r):
            if data[y][col]==attr[x]:
                if delete:
                    del data[y][col]
                dic[attr[x]][pos]=data[y]
                pos+=1
    return attr,dic
```

```
def entropy(S):
    attr=list(set(S))
    if len(attr)==1:
        return 0

    counts=[0,0]
    for i in range(2):
        counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)*1.0)

    sums=0
    for cnt in counts:
        sums+=-1*cnt*math.log(cnt,2)
    return sums

def compute_gain(data,col):
    attr,dic = subtables(data,col,delete=False)

    total_size=len(data)
    entropies=[0]*len(attr)
    ratio=[0]*len(attr)

    total_entropy=entropy([row[-1] for row in data])
    for x in range(len(attr)):
        ratio[x]=len(dic[attr[x]])/(total_size*1.0)
        entropies[x]=entropy([row[-1] for row in dic[attr[x]]])
        total_entropy-=ratio[x]*entropies[x]
    return total_entropy

def build_tree(data,features):
    lastcol=[row[-1] for row in data]
    if(len(set(lastcol)))==1:
        node=Node("")
        node.answer=lastcol[0]
        return node

    n=len(data[0])-1
    gains=[0]*n
    for col in range(n):
        gains[col]=compute_gain(data,col)
    split=gains.index(max(gains))
    node=Node(features[split])
```

```
    fea = features[:split]+features[split+1:]


    attr,dic=subtables(data,split,delete=True)

    for x in range(len(attr)):
        child=build_tree(dic[attr[x]],fea)
        node.children.append((attr[x],child))
    return node

def print_tree(node,level):
    if node.answer!="":
        print("  "*level,node.answer)
        return

    print("  "*level,node.attribute)
    for value,n in node.children:
        print("  "*(level+1),value)
        print_tree(n,level+2)


def classify(node,x_test,features):
    if node.answer!="":
        print(node.answer)
        return
    pos=features.index(node.attribute)
    for value, n in node.children:
        if x_test[pos]==value:
            classify(n,x_test,features)

'''Main program'''
dataset,features=load_csv("id3.csv")
node1=build_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")
print_tree(node1,0)
testdata,features=load_csv("id3_test.csv")

for xtest in testdata:
    print("The test instance:",xtest)
    print("The label for test instance:")
    classify(node1,xtest,features)
```

```
classify(node,xtest,features)
```

```
The decision tree for the dataset using ID3 algorithm is
 Outlook
   sunny
     Humidity
       normal
         yes
       high
         no
   rain
     Wind
       weak
         yes
       strong
         no
   overcast
     yes
The test instance: ['sunny', 'hot', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'hot', 'high', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:
yes
```

```
The label for test instance:
no
The test instance: ['overcast', 'hot', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'cool', 'normal', 'strong', 'no']
The label for test instance:
no
The test instance: ['overcast', 'cool', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['sunny', 'mild', 'high', 'weak', 'no']
The label for test instance:
no
The test instance: ['sunny', 'cool', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['sunny', 'mild', 'normal', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'mild', 'high', 'strong', 'yes']
The label for test instance:
yes
The test instance: ['overcast', 'hot', 'normal', 'weak', 'yes']
The label for test instance:
yes
The test instance: ['rain', 'mild', 'high', 'strong', 'no']
The label for test instance:
no
```

4.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

```
import csv
import random
import math
def loadcsv(filename):
lines = csv.reader(open("naive.csv", "r"));
dataset = list(lines)
for i in range(len(dataset)):
    #converting strings into numbers for processing
        dataset[i] = [float(x) for x in dataset[i]]

return dataset

def splitdataset(dataset, splitratio):
  #67% training size
trainsize = int(len(dataset) * splitratio);
trainset = []
copy = list(dataset);
while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
        index = random.randrange(len(copy));
        trainset.append(copy.pop(index))
return [trainset, copy]

def separatebyclass(dataset):
separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
                separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
return separated

def mean(numbers):
return sum(numbers)/float(len(numbers))
```

```
def stdev(numbers):
avg = mean(numbers)
variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
del summaries[-1] #excluding labels +ve or -ve
return summaries

def summarizebyclass(dataset):
separated = separatebyclass(dataset);
   #print(separated)
summaries = {}
for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
          summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and
std
return summaries

def calculateprobability(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

def calculateclassprobabilities(summaries, inputvector):
probabilities = {} # probabilities contains the all prob of all class of test data
for classvalue, classsummaries in summaries.items():#class and attribute information as mean
and sd
          probabilities[classvalue] = 1
          for i in range(len(classsummaries)):
                    mean, stdev = classsummaries[i] #take mean and sd of every attribute for class 0
and 1 seperaely
                    x = inputvector[i] #testvector's first attribute
                    probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal
dist
return probabilities
def predict(summaries, inputvector): #training and test data is passed
probabilities = calculateclassprobabilities(summaries, inputvector)
bestLabel, bestProb = None, -1
for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
          if bestLabel is None or probability > bestProb:
```

```
                bestProb = probability
                bestLabel = classvalue
return bestLabel

def getpredictions(summaries, testset):
predictions = []
for i in range(len(testset)):
        result = predict(summaries, testset[i])
        predictions.append(result)
return predictions

def getaccuracy(testset, predictions):
correct = 0
for i in range(len(testset)):
        if testset[i][-1] == predictions[i]:
                correct += 1
return (correct/float(len(testset))) * 100.0
def main():
filename = 'naivedata.csv'
splitratio = 0.67
dataset = loadcsv(filename);

trainingset, testset = splitdataset(dataset, splitratio)
print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))
# prepare model
summaries = summarizebyclass(trainingset);
#print(summaries)
  # test model
predictions = getpredictions(summaries, testset) #find the predictions of test data with the
training data
accuracy = getaccuracy(testset, predictions)
print('Accuracy of the classifier is : {0}%'.format(accuracy))
main()
```

```
Confusion matrix
[[139  21]
 [ 38  56]]

Accuracy of the classifier: 0.7677165354330708
The value of Precision: 0.7272727272727273
The value of Recall: 0.5957446808510638
Predicted Value for individual Test Data: [1]
```

5.Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

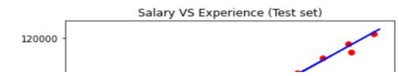
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('salary.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
LinearRegression()
# Predicting the Test set results
y_pred = regressor.predict(X_test)
# Visualizing the Training set results
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.title('Salary VS Experience (Training set)')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()

# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

jupyter Linear regression Last Checkpoint: 4 minutes ago (autosaved)

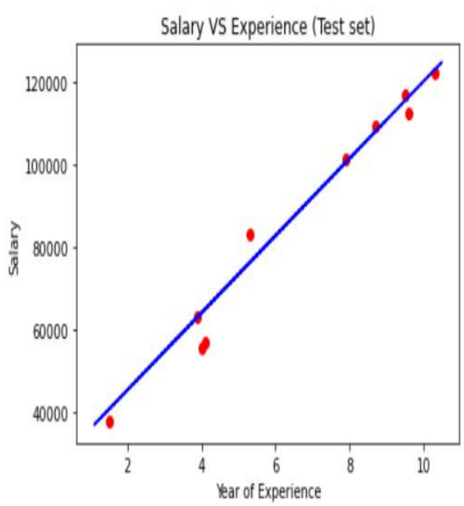
File Edit View Insert Cell Kernel Widgets Help Trusted Pyt

```
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.show()
```

In [5]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```



In [5]:
```
# Visualizing the Test set results
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [ ]:

6. Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

```
import math;
import sys;
import pandas as pd
import numpy as np
from random import choice
from matplotlib import pyplot
from random import shuffle, uniform;

def ReadData(fileName):
    f = open(fileName,'r')
    lines = f.read().splitlines()
    f.close()
    items = []
    for i in range(1,len(lines)):
        line = lines[i].split(',')
        itemFeatures = []
        for j in range(len(line)-1):
            v = float(line[j])
            itemFeatures.append(v)
        items.append(itemFeatures)
    shuffle(items)
    return items


def FindColMinMax(items):
    n = len(items[0])
```

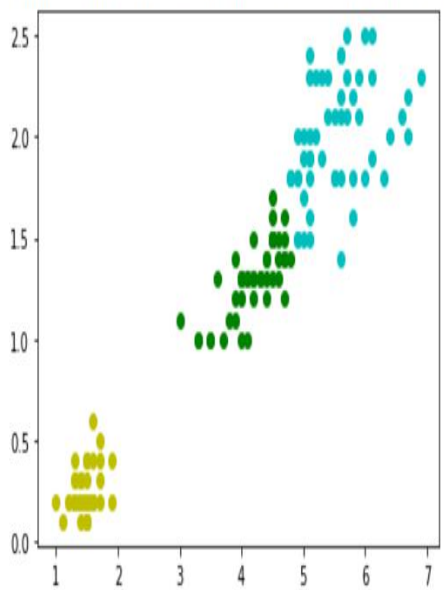
```
    minima = [float('inf') for i in range(n)]
    maxima = [float('-inf') -1 for i in range(n)]
    for item in items:
        for f in range(len(item)):
            if(item[f] < minima[f]):
                minima[f] = item[f]
            if(item[f] > maxima[f]):
                maxima[f] = item[f]
    return minima,maxima

def EuclideanDistance(x,y):
    S = 0
    for i in range(len(x)):
        S += math.pow(x[i]-y[i],2)
    return math.sqrt(S)

def InitializeMeans(items,k,cMin,cMax):
    f = len(items[0])
    means = [[0 for i in range(f)] for j in range(k)]
    for mean in means:
        for i in range(len(mean)):
            mean[i] = uniform(cMin[i]+1,cMax[i]-1)
    return means

def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i]
        m = (m*(n-1)+item[i])/float(n)
```

```
  mean[i] = round(m,3)
    return mean


def FindClusters(means,items):
    clusters = [[] for i in range(len(means))]
    for item in items:
        index = Classify(means,item)
        clusters[index].append(item)
    return clusters

def Classify(means,item):
    minimum = float('inf');
    index = -1
    for i in range(len(means)):
        dis = EuclideanDistance(item,means[i])
        if(dis < minimum):
            minimum = dis
            index = i
    return index

def CalculateMeans(k,items,maxIterations=100000):
    cMin, cMax = FindColMinMax(items)
    means = InitializeMeans(items,k,cMin,cMax)
    clusterSizes = [0 for i in range(len(means))]
    belongsTo = [0 for i in range(len(items))]
    for e in range(maxIterations):
        noChange = True;
```

```
        for i in range(len(items)):
            item = items[i];
            index = Classify(means,item)
            clusterSizes[index] += 1
            cSize = clusterSizes[index]
            means[index] = UpdateMean(cSize,means[index],item)
            if(index != belongsTo[i]):
                noChange = False
            belongsTo[i] = index
        if (noChange):
            break
    return means

def CutToTwoFeatures(items,indexA,indexB):
    n = len(items)
    X = []
    for i in range(n):
        item = items[i]
        newItem = [item[indexA],item[indexB]]
        X.append(newItem)
    return X

def PlotClusters(clusters):
    n = len(clusters)
    X = [[] for i in range(n)]
    for i in range(n):
        cluster = clusters[i]
        for item in cluster:
```

```
            X[i].append(item)
    colors = ['r','b','g','c','m','y']
    for x in X:
        c = choice(colors)
        colors.remove(c)
        Xa = []
        Xb = []
        for item in x:
            Xa.append(item[0])
            Xb.append(item[1])
        pyplot.plot(Xa,Xb,'o',color=c)
    pyplot.show()

def main():
    items = ReadData('data.txt')
    k = 3
    items = CutToTwoFeatures(items,2,3)
    print(items)
    means = CalculateMeans(k,items)
    print("\nMeans = ", means)
    clusters = FindClusters(means,items)

    PlotClusters(clusters)
    newItem = [1.5,0.2]
    print(Classify(means,newItem))

if __name__ == "__main__":
    main()
```

```
[[4.0, 1.3], [1.5, 0.4], [4.2, 1.3], [6.3, 1.8], [5.6, 2.1], [3.5, 1.0], [4.3, 1.3], [5.7, 2.1], [4.5, 1.7], [5.1, 2.3], [4.8, 1.4], [4.6, 1.5], [1.3,
0.3], [4.7, 1.2], [3.6, 1.3], [5.2, 2.3], [5.9, 2.3], [4.5, 1.5], [5.6, 1.8], [4.1, 1.3], [5.1, 1.9], [1.5, 0.4], [1.4, 0.2], [6.4, 2.0], [5.7, 2.3],
[1.5, 0.1], [5.7, 2.5], [1.6, 0.2], [5.1, 1.5], [4.3, 1.3], [1.6, 0.6], [5.8, 1.8], [3.8, 1.1], [5.5, 2.1], [5.8, 1.6], [5.4, 2.3], [5.1, 2.4], [4.9,
1.5], [1.4, 0.3], [4.6, 1.4], [1.3, 0.3], [4.7, 1.4], [1.6, 0.2], [4.4, 1.3], [6.6, 2.1], [1.5, 0.2], [3.3, 1.0], [1.3, 0.2], [4.4, 1.2], [4.7, 1.4],
[1.4, 0.2], [4.8, 1.8], [1.4, 0.2], [1.7, 0.3], [5.3, 1.9], [4.7, 1.6], [1.2, 0.2], [4.0, 1.3], [4.2, 1.3], [1.2, 0.2], [1.0, 0.2], [5.0, 1.9], [1.7,
0.2], [4.4, 1.4], [5.6, 2.4], [6.1, 1.9], [5.6, 2.4], [1.3, 0.2], [1.4, 0.3], [3.3, 1.0], [4.5, 1.5], [1.6, 0.4], [3.5, 1.0], [1.5, 0.3], [4.5, 1.5],
[5.3, 2.3], [1.5, 0.1], [4.0, 1.2], [4.7, 1.5], [1.5, 0.1], [1.7, 0.5], [1.6, 0.2], [4.0, 1.0], [5.5, 1.8], [4.6, 1.3], [1.3, 0.4], [1.3, 0.2], [5.9,
2.1], [4.4, 1.4], [1.5, 0.2], [1.1, 0.1], [1.6, 0.2], [1.5, 0.2], [1.4, 0.3], [3.9, 1.2], [5.1, 1.6], [3.0, 1.1], [5.0, 2.0], [4.5, 1.5], [5.0, 1.5],
[1.6, 0.2], [1.5, 0.2], [4.2, 1.5], [5.5, 1.8], [6.1, 2.5], [4.2, 1.2], [4.5, 1.6], [6.0, 2.5], [4.9, 1.8], [6.7, 2.0], [5.1, 2.0], [3.9, 1.1], [1.4,
0.2], [1.9, 0.4], [4.1, 1.3], [4.8, 1.8], [1.5, 0.1], [4.5, 1.5], [1.4, 0.1], [5.0, 1.7], [1.3, 0.2], [1.7, 0.4], [1.4, 0.2], [4.5, 1.3], [3.9, 1.4],
[5.1, 1.9], [5.4, 2.1], [4.1, 1.0], [4.0, 1.3], [1.4, 0.2], [5.2, 2.0], [1.5, 0.2], [6.0, 1.8], [1.9, 0.2], [5.6, 2.2], [5.8, 2.2], [1.5, 0.2], [3.7,
1.0], [6.9, 2.3], [4.9, 1.5], [1.4, 0.2], [6.7, 2.2], [4.9, 2.0], [4.8, 1.8], [6.1, 2.3], [1.5, 0.4], [5.6, 1.4], [5.1, 1.8], [4.9, 1.8]]

Means =  [[1.463, 0.258], [5.458, 1.962], [4.164, 1.285]]
```

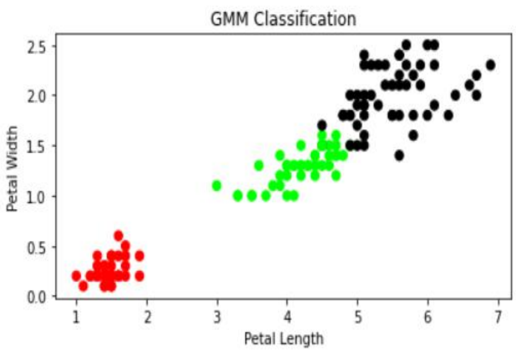


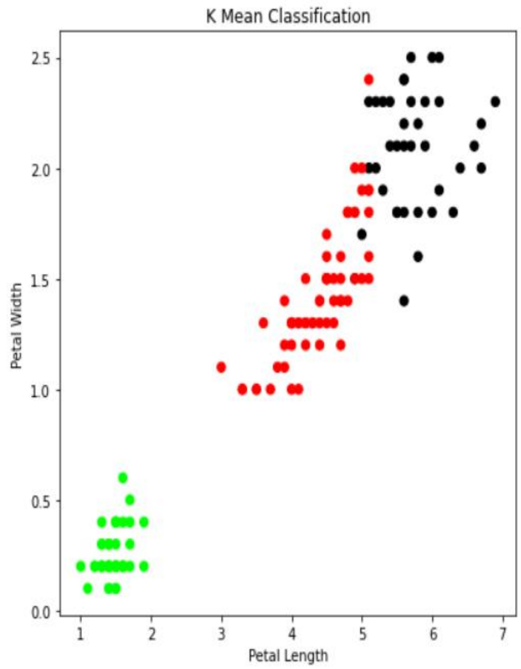
7.Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
model = KMeans(n_clusters=3)
model.fit(X)

plt.figure(figsize=(14,7))
colormap = np.array(['red', 'lime', 'black'])
# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
```

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))

from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)
y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```





8.Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

```
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix

from sklearn import datasets


iris=datasets.load_iris()


x = iris.data

y = iris.target


print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')

print(y)


x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)


#To Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(x_train, y_train)


#To make predictions on our test data

y_pred=classifier.predict(x_test)


print('Confusion Matrix')

print(confusion_matrix(y_test,y_pred))

print('Accuracy Metrics')
```

print(classification_report(y_test,y_pred))

```
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Confusion Matrix
[[16  0  0]
 [ 0 11  0]
 [ 0  2 16]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.85      1.00      0.92        11
           2       1.00      0.89      0.94        18

    accuracy                           0.96        45
   macro avg       0.95      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

9. Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

```
!pip install bayespy

import bayespy as bp

import numpy as np

import csv

!pip3 install colorama

!pip3 install colorama

from colorama import init

from colorama import Fore, Back, Style

init()

ageEnum = {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1,

        'MiddleAged': 2, 'Youth': 3, 'Teen': 4}

# Gender

genderEnum = {'Male': 0, 'Female': 1}

# FamilyHistory

familyHistoryEnum = {'Yes': 0, 'No': 1}

# Diet(Calorie Intake)

dietEnum = {'High': 0, 'Medium': 1, 'Low': 2}

# LifeStyle

lifeStyleEnum = {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}

# Cholesterol

cholesterolEnum = {'High': 0, 'BorderLine': 1, 'Normal': 2}

# HeartDisease

heartDiseaseEnum = {'Yes': 0, 'No': 1}


import pandas as pd

data = pd.read_csv("heart_disease.csv")

data =np.array(data, dtype='int8')
```

```
N = len(data)

# Input data column assignment
p_age = bp.nodes.Dirichlet(1.0*np.ones(5))
age = bp.nodes.Categorical(p_age, plates=(N,))
age.observe(data[:, 0])

p_gender = bp.nodes.Dirichlet(1.0*np.ones(2))
gender = bp.nodes.Categorical(p_gender, plates=(N,))
gender.observe(data[:, 1])

p_familyhistory = bp.nodes.Dirichlet(1.0*np.ones(2))
familyhistory = bp.nodes.Categorical(p_familyhistory, plates=(N,))
familyhistory.observe(data[:, 2])

p_diet = bp.nodes.Dirichlet(1.0*np.ones(3))
diet = bp.nodes.Categorical(p_diet, plates=(N,))
diet.observe(data[:, 3])

p_lifestyle = bp.nodes.Dirichlet(1.0*np.ones(4))
lifestyle = bp.nodes.Categorical(p_lifestyle, plates=(N,))
lifestyle.observe(data[:, 4])

p_cholesterol = bp.nodes.Dirichlet(1.0*np.ones(3))
cholesterol = bp.nodes.Categorical(p_cholesterol, plates=(N,))
cholesterol.observe(data[:, 5])

# Prepare nodes and establish edges
```

```
# np.ones(2) -> HeartDisease has 2 options Yes/No
# plates(5, 2, 2, 3, 4, 3) -> corresponds to options present for domain values
p_heartdisease = bp.nodes.Dirichlet(np.ones(2), plates=(5, 2, 2, 3, 4, 3))
heartdisease = bp.nodes.MultiMixture(
    [age, gender, familyhistory, diet, lifestyle, cholesterol], bp.nodes.Categorical, p_heartdisease)
heartdisease.observe(data[:, 6])
p_heartdisease.update()


m = 0
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(input('Enter FamilyHistory: ' + str(familyHistoryEnum))), int(input('Enter dietEnum: ' + str(
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)))], bp.nodes.Categorical, p_heartdisease).get_moments()[0][heartDiseaseEnum['Yes']]
    print("Probability(HeartDisease) = " + str(res))


# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

```python
while m == 0:
    print("\n")
    res = bp.nodes.MultiMixture([int(input('Enter Age: ' + str(ageEnum))), int(input('Enter Gender: ' + str(genderEnum))), int(ir
        dietEnum))), int(input('Enter LifeStyle: ' + str(lifeStyleEnum))), int(input('Enter Cholesterol: ' + str(cholesterolEnum)
    print("Probability(HeartDisease) = " + str(res))

# print(Style.RESET_ALL)
    m = int(input("Enter for Continue:0, Exit :1 "))
```

```
Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}0
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}0
Probability(HeartDisease) = 0.5
Enter for Continue:0, Exit :1 0


Enter Age: {'SuperSeniorCitizen': 0, 'SeniorCitizen': 1, 'MiddleAged': 2, 'Youth': 3, 'Teen': 4}0
Enter Gender: {'Male': 0, 'Female': 1}0
Enter FamilyHistory: {'Yes': 0, 'No': 1}0
Enter dietEnum: {'High': 0, 'Medium': 1, 'Low': 2}0
Enter LifeStyle: {'Athlete': 0, 'Active': 1, 'Moderate': 2, 'Sedetary': 3}0
Enter Cholesterol: {'High': 0, 'BorderLine': 1, 'Normal': 2}0
Probability(HeartDisease) = 0.5
```

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

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
from scipy.stats.stats import pearsonr
def kernel(point,xmat, k):
 m,n = np1.shape(xmat)
 weights = np1.mat(np1.eye((m)))
 for j in range(m):
    diff = point - X[j]
    weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
 return weights
def localWeight(point,xmat,ymat,k):
 wei = kernel(point,xmat,k)
 W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
 return W
def localWeightRegression(xmat,ymat,k):
 m,n = np1.shape(xmat)
 ypred = np1.zeros(m)
 for i in range(m):
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
 return ypred
# load data points
```

```
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show()
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook
def local_regression(x0, X, Y, tau):# add bias term
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
```

```
    X = np.c_[np.ones(len(X)), X]
    # fit model: normal equations with kernel
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W
    beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product
    # predict value
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction
def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
# Weight or Radial Kernal Bias Function
n = 1000
# generate dataset
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
```

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-2.95983905 -2.77699311 -3.06439147 -3.15903005 -3.19868861 -3.00406048
```

-2.9445708 -2.87933746 -2.94253902]

Xo Domain Space(10 Samples) :

[-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866

-2.85953177 -2.83946488 -2.81939799]

```
def plot_lwr(tau):
# prediction through regression
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot
show(gridplot([
[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))
```