**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**COURSE TITLE**

***Submitted by***

**NISHCHAL NANDAGOPAL (1BM19CS105)**

***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 “MACHINE LEARNING” carried out by **NISHCHAL NANDAGOPAL (1BM19CS105),** 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** work prescribed for the said degree.

Name of the Lab-Incharge               **Dr. Jyothi S Nayak**

Designation Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **SL NO** | **EXPERIMENT** | **PAGE NO** |
| **1** | **FIND S ALGORITHM** | **3** |
| **2** | **CANDIDATE ELIMINATION ALGORITHM** | **4** |
| **3** | **DECISION TREE USING ID3 ALGORITHM** | **6** |
| **4** | **LINEAR REGRESSION** | **10** |
| **5** | **NAÏVE BAYES NETWORK** | **12** |
| **6** | **BAYESIAN NETWORK** | **14** |
| **7** | **EM ALGORITHM** | **16** |
| **8** | **K NEAREST NEIGHBOUR ALGORITHM** | **18** |
| **9** | **LOCALLY WEIGHTED REGRESSION** | **19** |
| **10** | **K MEANS ALGORITHM** | **21** |

**WEEK 1**

**FIND S ALGORITHM**

**import** pandas **as** pd

**import** numpy **as** np

*#to read the data in the csv file*

print("USN:1BM19CS105")

data **=** pd**.**read\_csv(r"C:\Users\admin\Downloads\data.csv")

print(data,"\n")

*#making an array of all the attributes*

d **=** np**.**array(data)[:,:**-**1]

print("The attributes are: ",d)

*#segragating the target that has positive and negative examples*

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

print("The target is: ",target)

*#training function to implement find-s algorithm*

**def** train(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**:

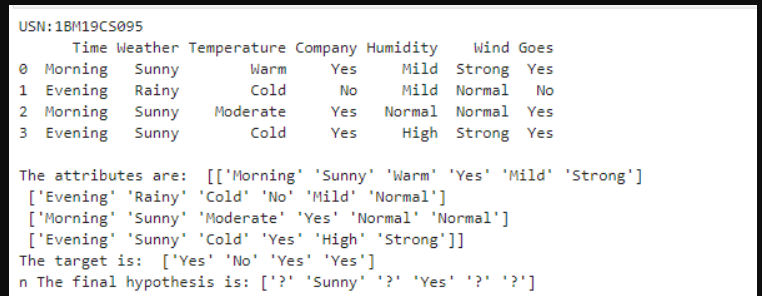
**pass**

**return** specific\_hypothesis

*#obtaining the final hypothesis*

print("n The final hypothesis is:",train(d,target))

**OUTPUT**

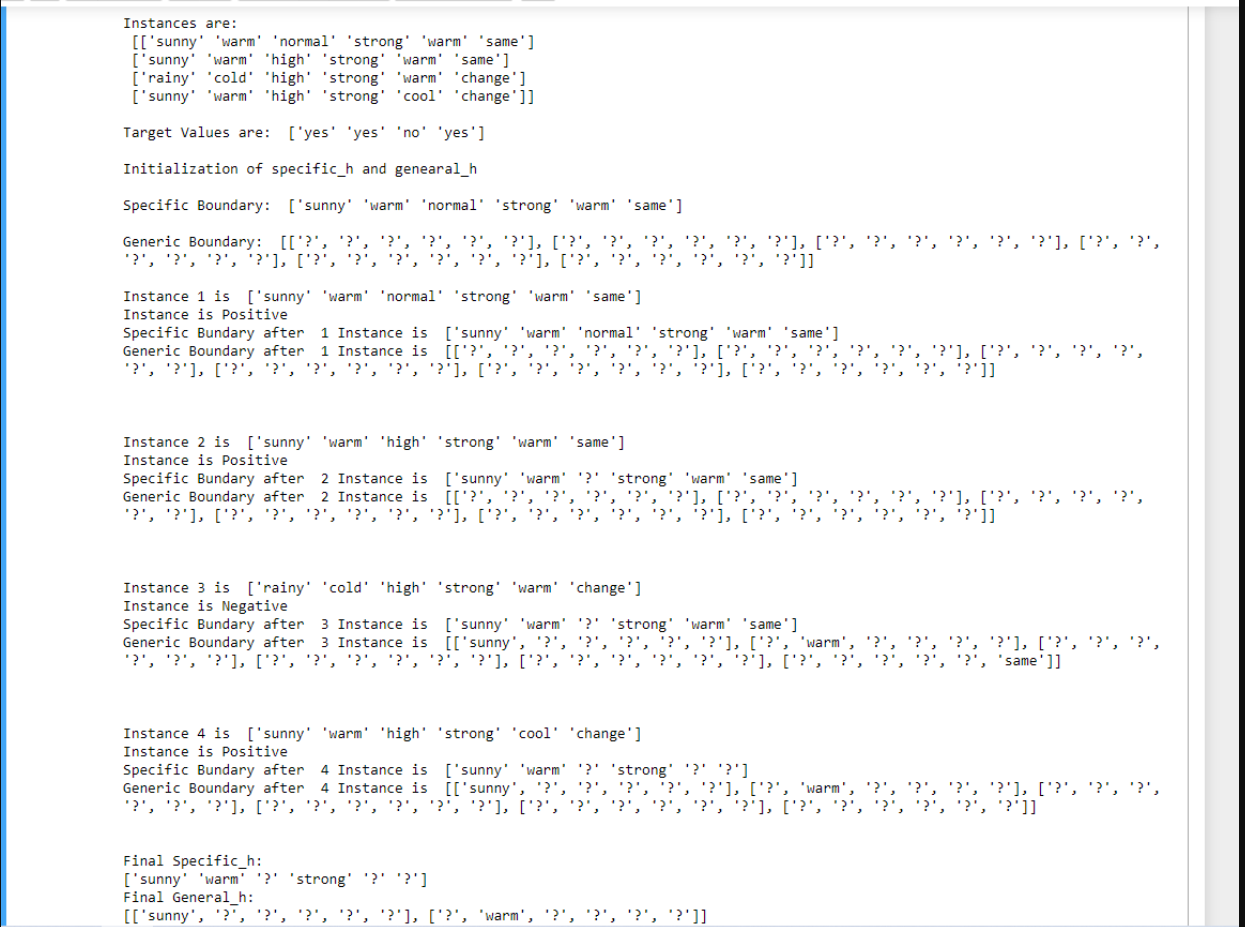


**WEEK 2**

**CANDIDATE ELIMINATION ALGORITHM**

|  |
| --- |
| Import numpy as np |
|  | import pandas as pd |
|  |  |
|  | data = pd.read\_csv(r'C:\Users\admin\Downloads\enjoysport.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") |

**OUTPUT**

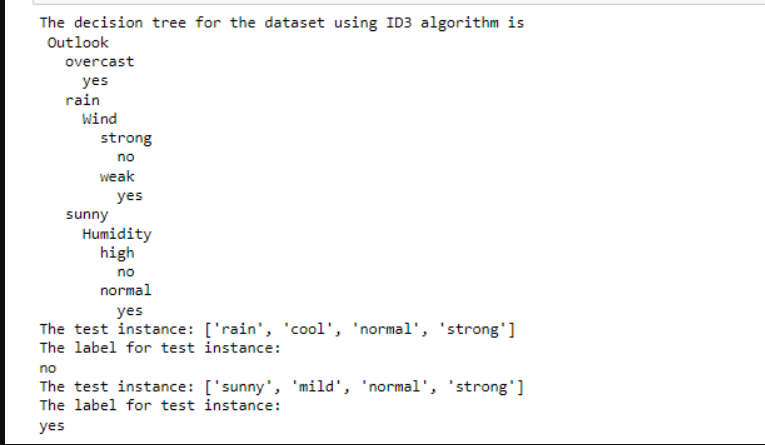


**WEEK 3**

**DECISION TREE USING ID3 ALGORITHM**

|  |
| --- |
| 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(r"C:\Users\admin\Downloads\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(r"C:\Users\admin\Downloads\id3\_test.csv") |
|  |  |
|  | for xtest in testdata: |
|  | print("The test instance:",xtest) |
|  | print("The label for test instance:") |
|  | classify(node1,xtest,features) |

**OUTPUT**



**WEEK 4**

**LINEAR REGRESSION**

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** pandas **as** pd

dataset **=** pd**.**read\_csv('salary\_data.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)

*# 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()

**OUTPUT**





**LAB 5**

**NAÏVE BAYES NETWORK**

**import** pandas **as** pd

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn **import** metrics

df **=** pd**.**read\_csv(r"C:\Users\admin\Downloads\data5.csv")

col\_names **=** ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness', 'insulin', 'bmi', 'diab\_pred', 'age']

predicted\_class **=** ['diabetes']

X **=** df[col\_names]**.**values

y **=** df[predicted\_class]**.**values

print(df**.**head)

xtrain,xtest,ytrain,ytest**=**train\_test\_split(X,y,test\_size**=**0.4)

print ('\n the total number of Training Data :',ytrain**.**shape)

print ('\n the total number of Test Data :',ytest**.**shape)

clf **=** GaussianNB()**.**fit(xtrain,ytrain**.**ravel())

predicted **=** clf**.**predict(xtest)

predictTestData**=** clf**.**predict([[6,148,72,35,0,33.6,0.627,50]])

print('\n Confusion matrix')

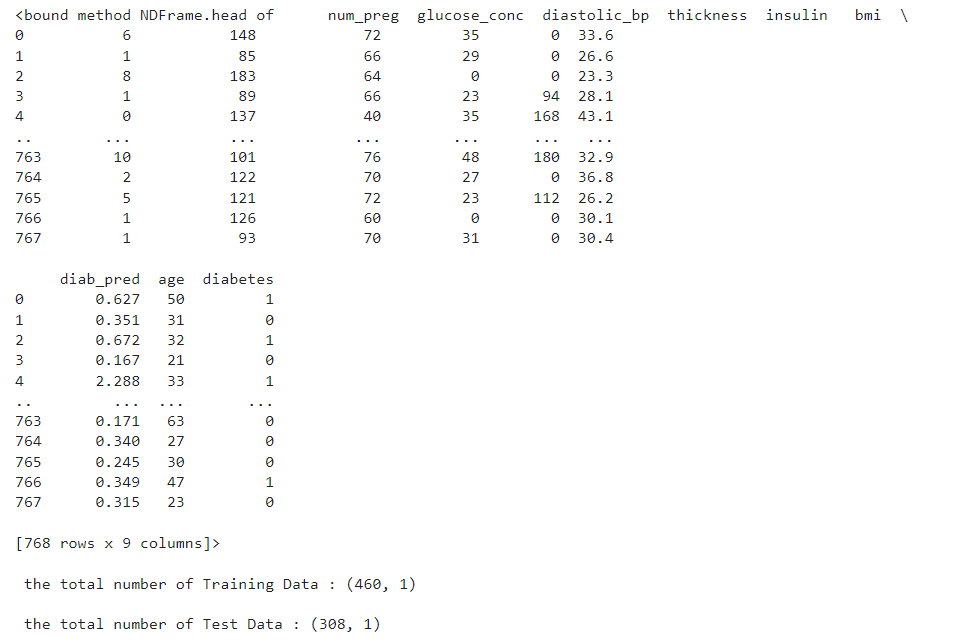
print(metrics**.**confusion\_matrix(ytest,predicted))

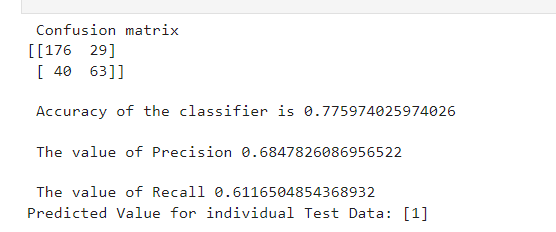
print('\n Accuracy of the classifier is',metrics**.**accuracy\_score(ytest,predicted))

print('\n The value of Precision', metrics**.**precision\_score(ytest,predicted))

print('\n The value of Recall', metrics**.**recall\_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)





**LAB6**

**BAYESIAN NETWORK**

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

cancer\_model = BayesianModel([('Pollution', 'Cancer'),

('Smoker', 'Cancer'),

('Cancer', 'Xray'),

('Cancer', 'Dyspnoea')])

print('Bayesian network nodes:')

print('\t', cancer\_model.nodes())

print('Bayesian network edges:')

print('\t', cancer\_model.edges())

cpd\_poll **=** TabularCPD(variable**=**'Pollution', variable\_card**=**2,

values**=**[[0.9], [0.1]])

cpd\_smoke **=** TabularCPD(variable**=**'Smoker', variable\_card**=**2,

values**=**[[0.3], [0.7]])

cpd\_cancer **=** TabularCPD(variable**=**'Cancer', variable\_card**=**2,

values**=**[[0.03, 0.05, 0.001, 0.02],

[0.97, 0.95, 0.999, 0.98]],

evidence**=**['Smoker', 'Pollution'],

evidence\_card**=**[2, 2])

cpd\_xray **=** TabularCPD(variable**=**'Xray', variable\_card**=**2,

values**=**[[0.9, 0.2], [0.1, 0.8]],

evidence**=**['Cancer'], evidence\_card**=**[2])

cpd\_dysp **=** TabularCPD(variable**=**'Dyspnoea', variable\_card**=**2,

values**=**[[0.65, 0.3], [0.35, 0.7]],

evidence**=**['Cancer'], evidence\_card**=**[2])

In [6]:

cancer\_model**.**add\_cpds(cpd\_poll, cpd\_smoke, cpd\_cancer, cpd\_xray, cpd\_dysp)

print('Model generated bt adding conditional probability distribution(cpds)')

*# Checking if the cpds are valid for the model.*

print('Checking for Correctness of model:', end**=**'')

print(cancer\_model**.**check\_model())

'''print('All local dependencies are as follows')

cancer\_model.get\_independencies()

'''

print('Displaying CPDs')

print(cancer\_model**.**get\_cpds('Pollution'))

print(cancer\_model**.**get\_cpds('Smoker'))

print(cancer\_model**.**get\_cpds('Cancer'))

print(cancer\_model**.**get\_cpds('Xray'))

print(cancer\_model**.**get\_cpds('Dyspnoea'))

cancer\_infer **=** VariableElimination(cancer\_model)

print('\nInferencing with Bayesian Network')

print('\nProbability of Cancer given Smoker')

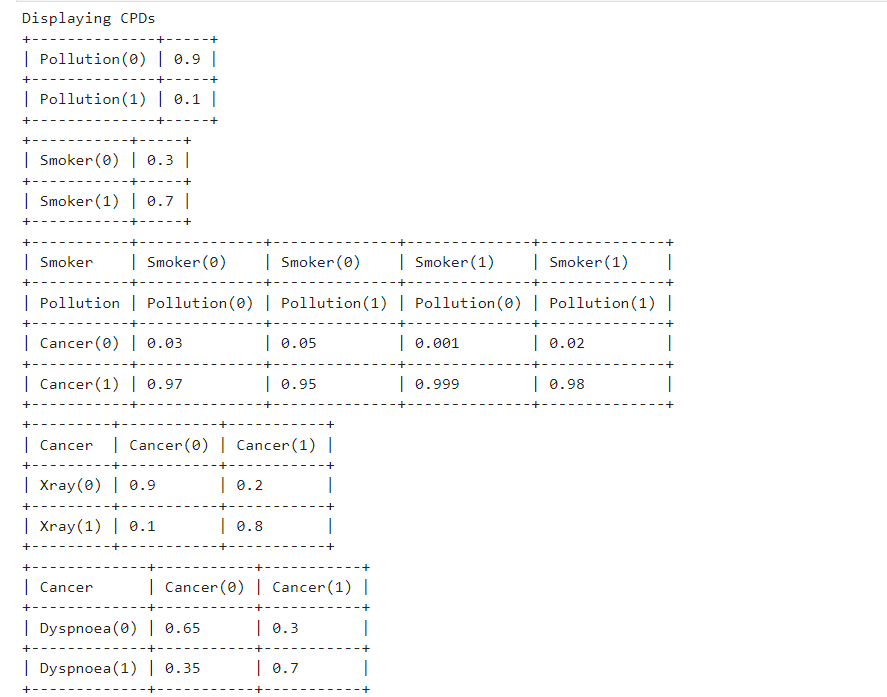
q **=** cancer\_infer**.**query(variables**=**['Cancer'], evidence**=**{'Smoker': 1})

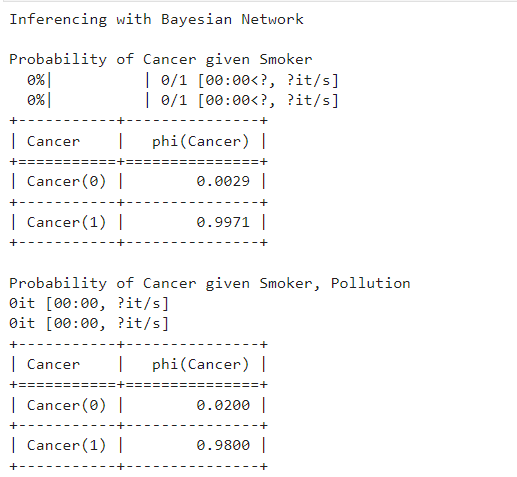
print(q)

print('\nProbability of Cancer given Smoker, Pollution')

q **=** cancer\_infer**.**query(variables**=**['Cancer'], evidence**=**{'Smoker': 1,'Pollution': 1})

print(q)





**LAB7**

**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)

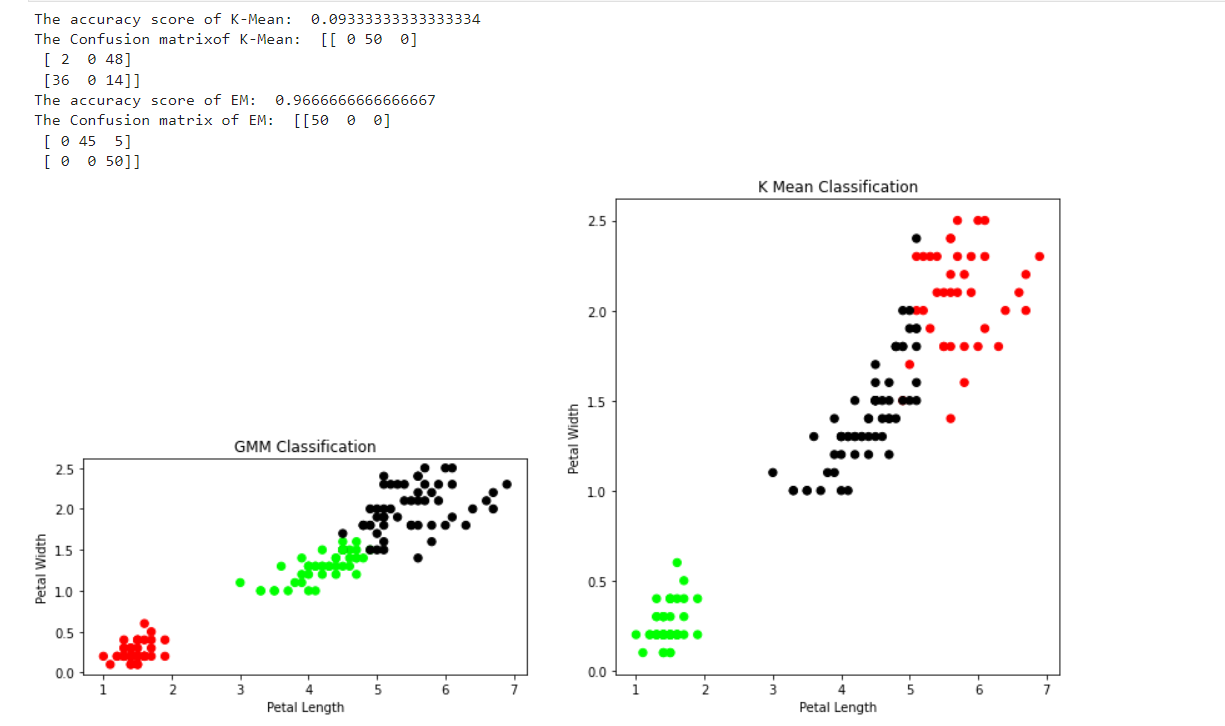
plt.title('GMM Classification')

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))



**LAB8**

**K NEAREST NEIGHBOUR ALGORITHM**

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

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*

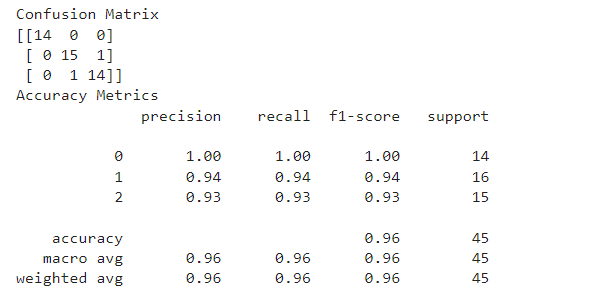
y\_pred=classifier.predict(x\_test)

print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

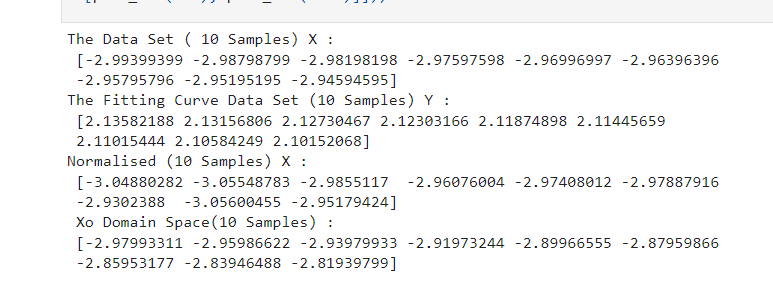
print(classification\_report(y\_test,y\_pred))

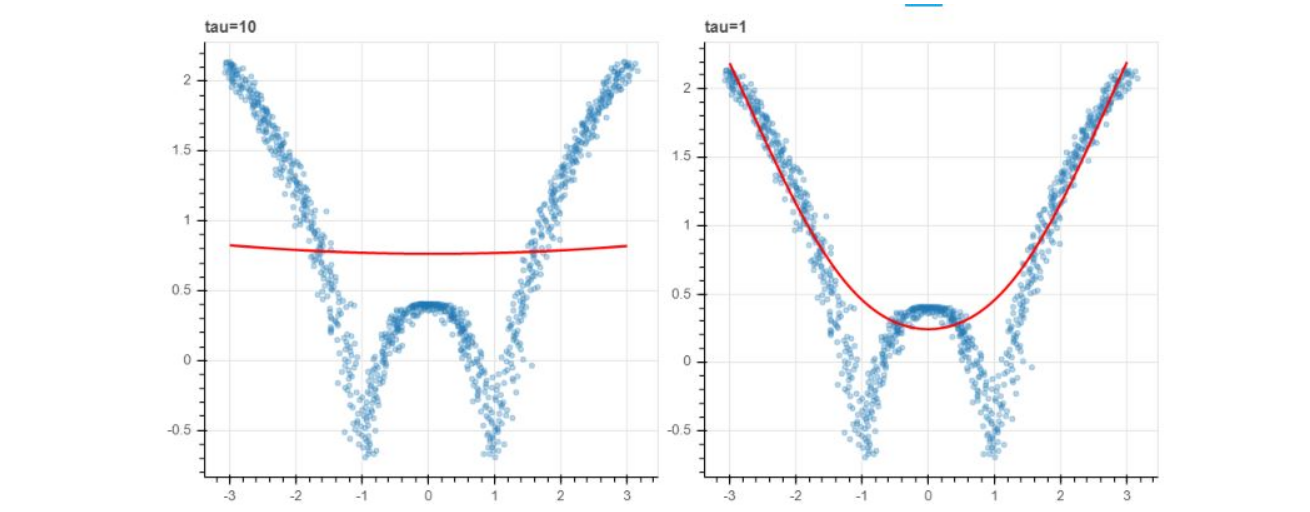


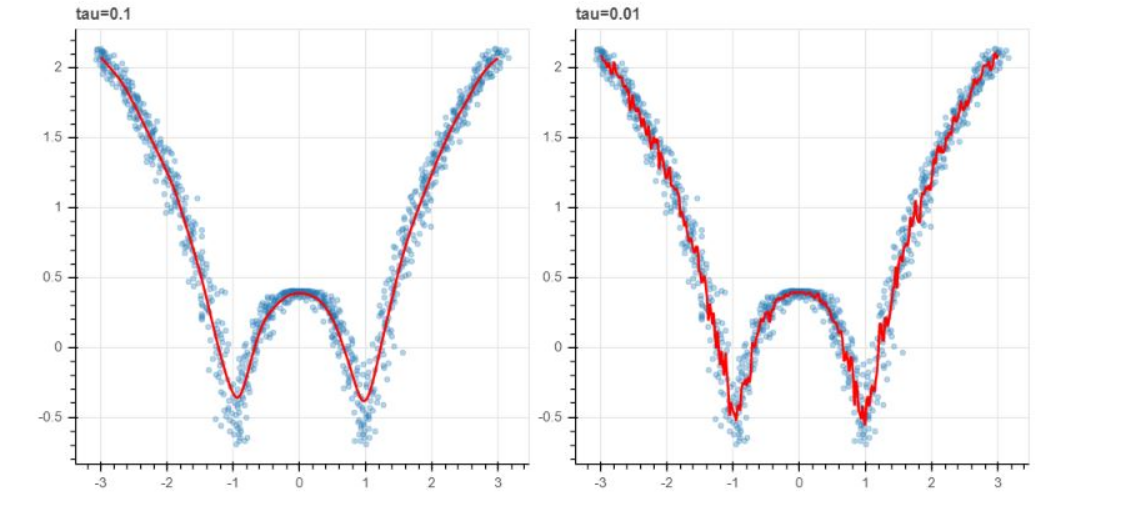
**LAB9**

**LOCALLY WEIGHTED REGRESSION**

|  |
| --- |
| 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]) |
|  |  |
|  | 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)]])) |







**LAB10**

**K MEANS ALGORITHM**

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()

