**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**MACHINE LEARNING**

**(20CS6PCMAL)**

***Submitted by***

**George Abraham(1BM19CS197)**

***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 **GEORGE ABRAHAM(1BM19CS197),** 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.

Name of the Lab-Incharge               **Dr.Asha.G.R**

Designation Assistant Professor

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**LAB PROGRAM 1:**

Implement and demonstrate the FIND-S algorithm for finding the most specific

hypothesis based on a given set of training data samples.

Code:

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

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

data**=**pd**.**read\_csv('find\_s.csv')

data

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

concepts

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

target

**def** train(con,tar):

**for** i,val **in** enumerate(tar):

**if** val**==**'yes':

specific\_h**=**con[i]**.**copy()

print(specific\_h)

**break**

**for** i,val **in** enumerate(con):

**if** tar[i]**==**'yes':

**for** x **in** range(len(specific\_h)):

**if** val[x]**!=**specific\_h[x]:

specific\_h[x]**=**'?'

**else**:

**pass**

**return** specific\_h

print(train(concepts,target))



**LAB PROGRAM 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.

Code:

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

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

data **=** pd**.**read\_csv('data.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:

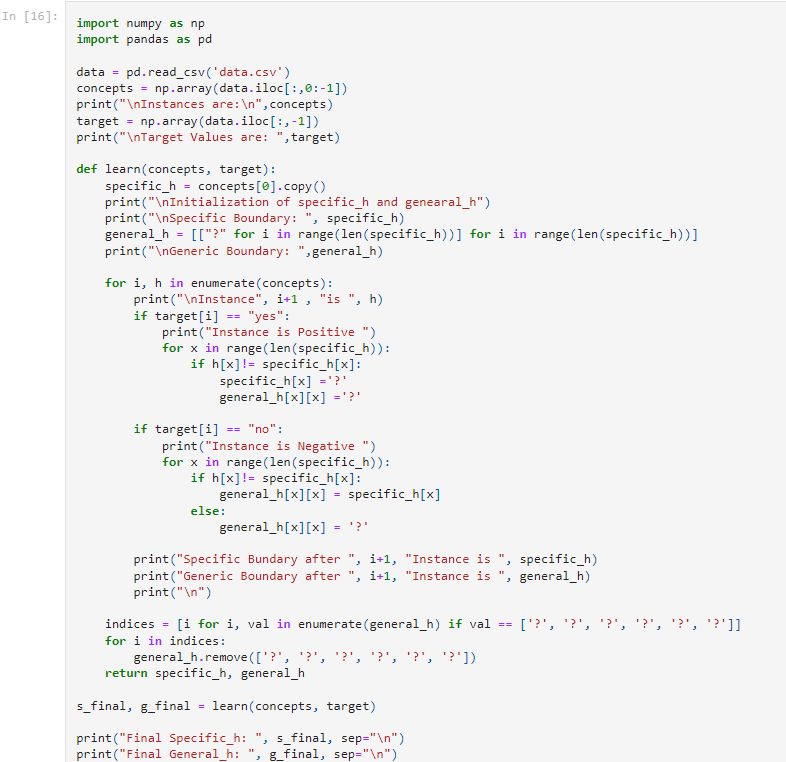
general\_h**.**remove(['?', '?', '?', '?', '?', '?'])

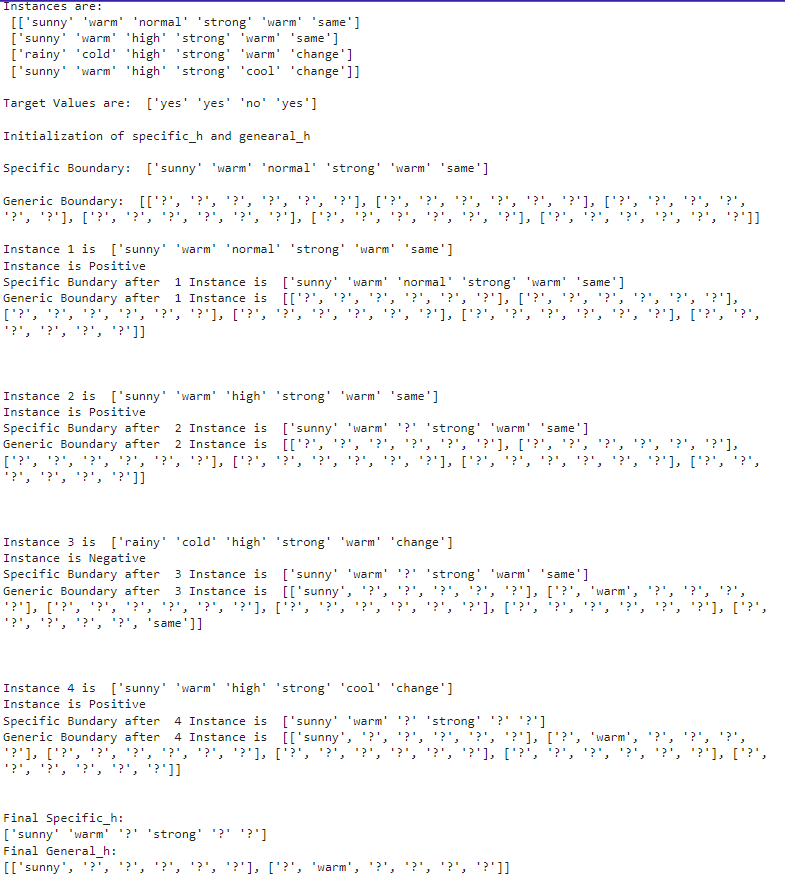
**return** specific\_h, general\_h

s\_final, g\_final **=** learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep**=**"\n")

print("Final General\_h: ", g\_final, sep**=**"\n")





**LAB PROGRAM 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.

Code:

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

**import** math

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

data **=** pd**.**read\_csv(r"C:\Users\admin\Desktop\3-dataset.csv")

features **=** [feat **for** feat **in** data]

features**.**remove("answer")

**class** Node:

**def** \_\_init\_\_(self):

self**.**children **=** []

self**.**value **=** ""

self**.**isLeaf **=** **False**

self**.**pred **=** ""

**def** entropy(examples):

pos **=** 0.0

neg **=** 0.0

**for** \_, row **in** examples**.**iterrows():

**if** row["answer"] **==** "yes":

pos **+=** 1

**else**:

neg **+=** 1

**if** pos **==** 0.0 **or** neg **==** 0.0:

**return** 0.0

**else**:

p **=** pos **/** (pos **+** neg)

n **=** neg **/** (pos **+** neg)

**return** **-**(p **\*** math**.**log(p, 2) **+** n **\*** math**.**log(n, 2))

**def** info\_gain(examples, attr):

uniq **=** np**.**unique(examples[attr])

*#print ("\n",uniq)*

gain **=** entropy(examples)

*#print ("\n",gain)*

**for** u **in** uniq:

subdata **=** examples[examples[attr] **==** u]

*#print ("\n",subdata)*

sub\_e **=** entropy(subdata)

gain **-=** (float(len(subdata)) **/** float(len(examples))) **\*** sub\_e

*#print ("\n",gain)*

**return** gain

**def** ID3(examples, attrs):

root **=** Node()

max\_gain **=** 0

max\_feat **=** ""

**for** feature **in** attrs:

*#print ("\n",examples)*

gain **=** info\_gain(examples, feature)

**if** gain **>** max\_gain:

max\_gain **=** gain

max\_feat **=** feature

root**.**value **=** max\_feat

*#print ("\nMax feature attr",max\_feat)*

uniq **=** np**.**unique(examples[max\_feat])

*#print ("\n",uniq)*

**for** u **in** uniq:

*#print ("\n",u)*

subdata **=** examples[examples[max\_feat] **==** u]

*#print ("\n",subdata)*

**if** entropy(subdata) **==** 0.0:

newNode **=** Node()

newNode**.**isLeaf **=** **True**

newNode**.**value **=** u

newNode**.**pred **=** np**.**unique(subdata["answer"])

root**.**children**.**append(newNode)

**else**:

dummyNode **=** Node()

dummyNode**.**value **=** u

new\_attrs **=** attrs**.**copy()

new\_attrs**.**remove(max\_feat)

child **=** ID3(subdata, new\_attrs)

dummyNode**.**children**.**append(child)

root**.**children**.**append(dummyNode)

**return** root

**def** printTree(root: Node, depth**=**0):

**for** i **in** range(depth):

print("\t", end**=**"")

print(root**.**value, end**=**"")

**if** root**.**isLeaf:

print(" -> ", root**.**pred)

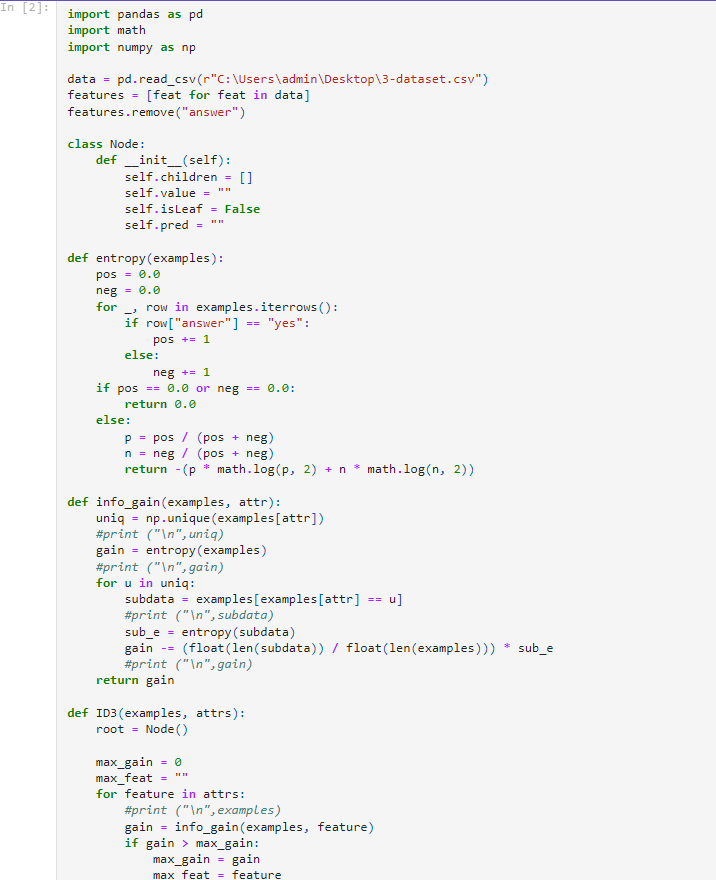
print()

**for** child **in** root**.**children:

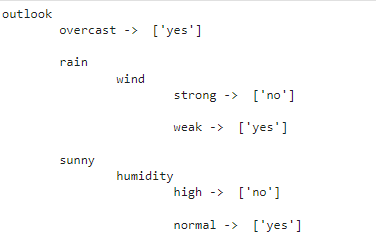
printTree(child, depth **+** 1)

root **=** ID3(data, features)

printTree(root)







**LAB PROGRAM 4:**

Implement the Linear Regression algorithm in order to fit data points. Select

appropriate data set for your experiment and draw graphs.

Code:

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

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

**from** sklearn **import** linear\_model

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

df **=** pd**.**read\_csv(r'/content/homeprices.csv')

df

**%**matplotlib inline

plt**.**xlabel('area')

plt**.**ylabel('price')

plt**.**scatter(df**.**area,df**.**price,color**=**'red',marker**=**'.')

new\_df **=** df**.**drop('price',axis**=**'columns')

df

Price **=** df**.**price

Price

reg **=** linear\_model**.**LinearRegression()

reg**.**fit(new\_df,Price)

reg**.**predict([[3300]])

reg**.**predict([[5000]])

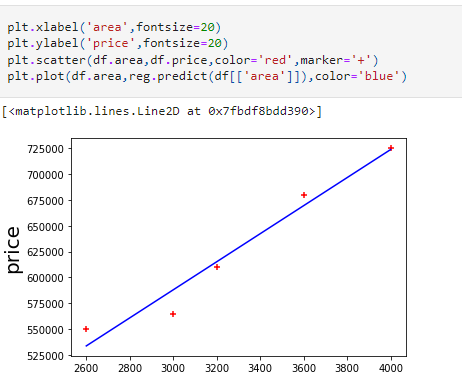
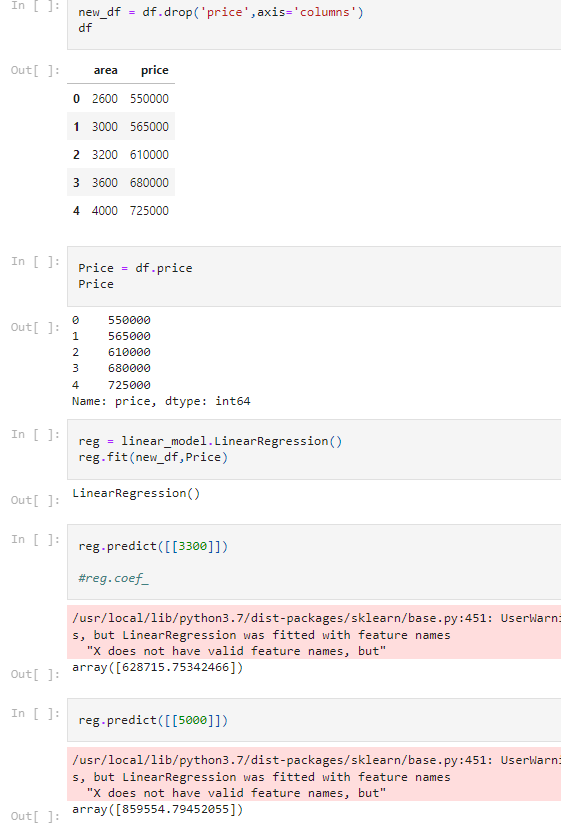
plt**.**xlabel('area',fontsize**=**20)

plt**.**ylabel('price',fontsize**=**20)

plt**.**scatter(df**.**area,df**.**price,color**=**'red',marker**=**'+')

plt**.**plot(df**.**area,reg**.**predict(df[['area']]),color**=**'blue')





**LAB PROGRAM 5:**

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

Code:

**from** sklearn.datasets **import** fetch\_20newsgroups

data **=** fetch\_20newsgroups()

data**.**target\_names

categories **=** ['talk.religion.misc', 'soc.religion.christian','rec.motorcycles',

'sci.space', 'comp.graphics']

train **=** fetch\_20newsgroups(subset**=**'train', categories**=**categories)

test **=** fetch\_20newsgroups(subset**=**'test', categories**=**categories)

**from** sklearn.feature\_extraction.text **import** TfidfVectorizer

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

**from** sklearn.pipeline **import** make\_pipeline

model **=** make\_pipeline(TfidfVectorizer(), MultinomialNB())

model**.**fit(train**.**data, train**.**target)

labels **=** model**.**predict(test**.**data)

**from** sklearn.metrics **import** confusion\_matrix

**import** seaborn **as** sns

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

mat **=** confusion\_matrix(test**.**target, labels)

sns**.**heatmap(mat**.**T, square**=True**, annot**=True**, fmt**=**'d', cbar**=False**,

xticklabels**=**train**.**target\_names, yticklabels**=**train**.**target\_names)

plt**.**xlabel('true label')

plt**.**ylabel('predicted label');

**def** predict\_category(s, train**=**train, model**=**model):

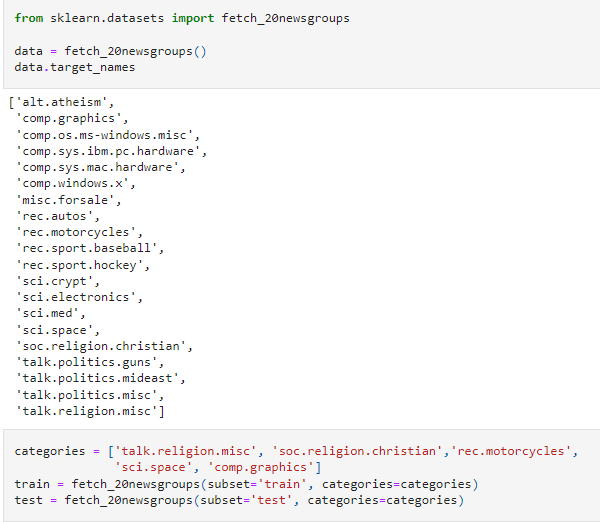
pred **=** model**.**predict([s])

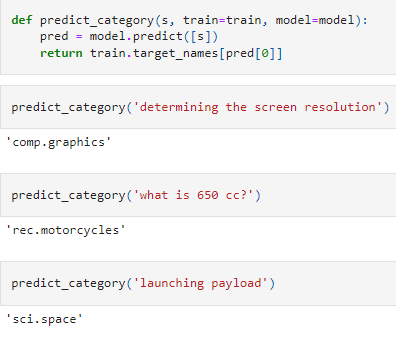
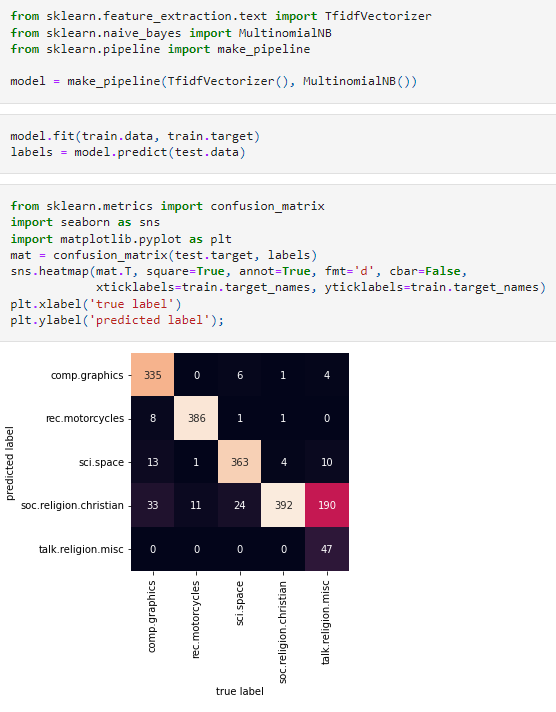
**return** train**.**target\_names[pred[0]]

predict\_category('determining the screen resolution')

predict\_category('what is 650 cc?')

predict\_category('launching payload')





**LAB PROGRAM 6:**

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

Code:

**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)

**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')

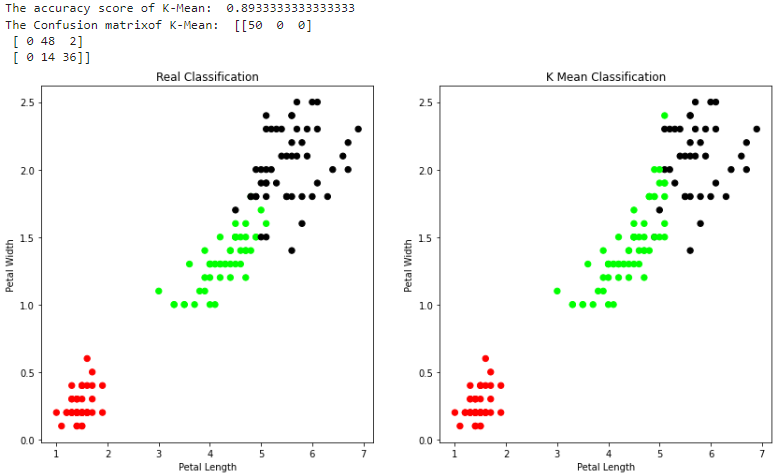
plt**.**xlabel('Petal Length')

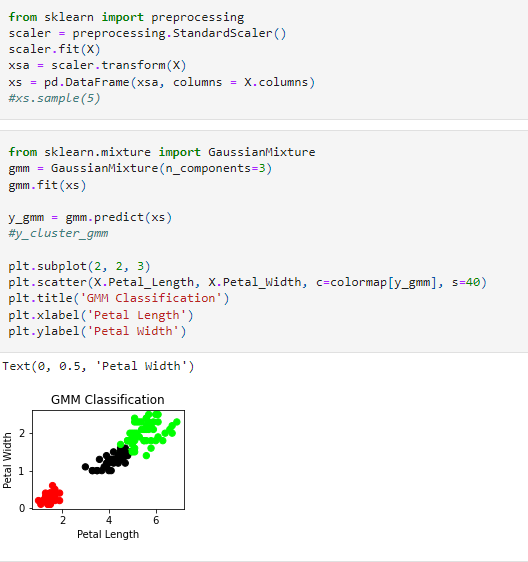
plt**.**ylabel('Petal Width')

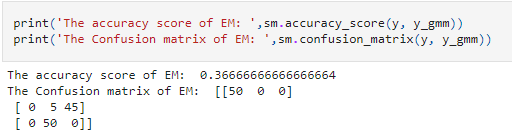
print('The accuracy score of EM: ',sm**.**accuracy\_score(y, y\_gmm))

print('The Confusion matrix of EM: ',sm**.**confusion\_matrix(y, y\_gmm))









**LAB PROGRAM 7:**

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

Code:

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

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

**import** csv

**from** pgmpy.estimators **import** MaximumLikelihoodEstimator

**from** pgmpy.models **import** BayesianModel

**from** pgmpy.inference **import** VariableElimination

heartDisease **=** pd**.**read\_csv('heart.csv')

heartDisease **=** heartDisease**.**replace('?',np**.**nan)

print('Sample instances from the dataset are given below')

print('\n Attributes and datatypes')

print(heartDisease**.**dtypes)

model**=** BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model**.**fit(heartDisease,estimator**=**MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest\_infer **=** VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')

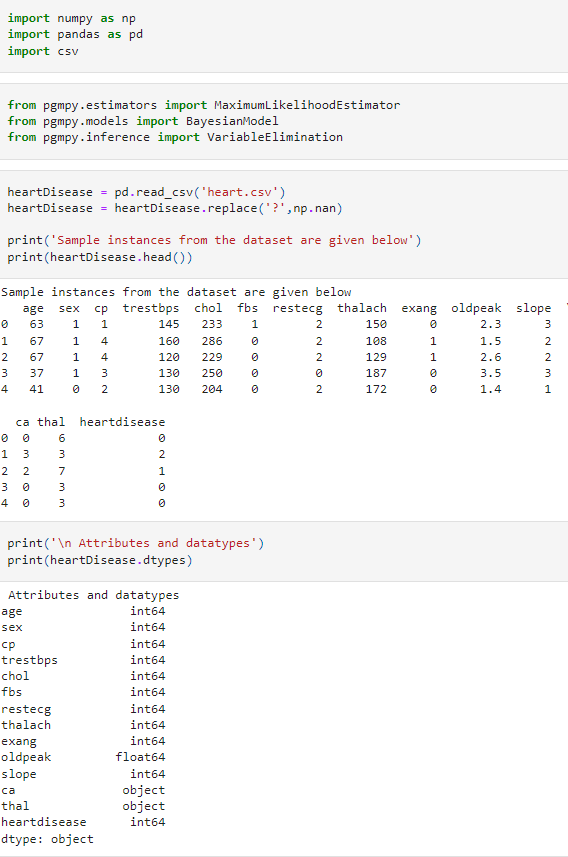
q1**=**HeartDiseasetest\_infer**.**query(variables**=**['heartdisease'],evidence**=**{'restecg':1})

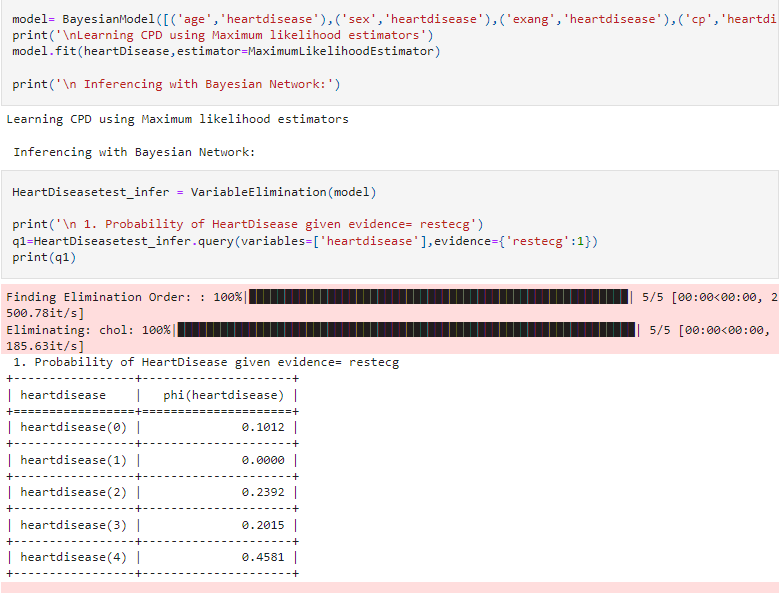
print(q1)

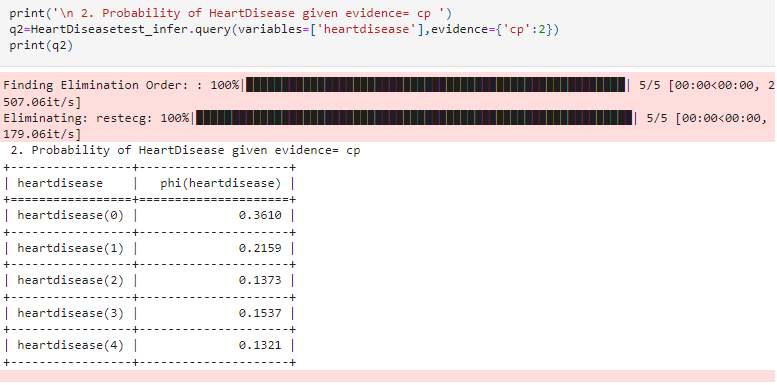
print('\n 2. Probability of HeartDisease given evidence= cp ')

q2**=**HeartDiseasetest\_infer**.**query(variables**=**['heartdisease'],evidence**=**{'cp':2})

print(q2)







**LAB PROGRAM 8:**

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

Code:

**from** sklearn **import** datasets

**from** sklearn.cluster **import** KMeans

**from** sklearn.utils **import** shuffle

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

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

iris**=**datasets**.**load\_iris()

X**=**iris**.**data

Y**=**iris**.**target

*#Shuffle of Data*

X,Y **=** shuffle(X,Y)

model**=**KMeans(n\_clusters**=**3,init**=**'k-means++',max\_iter**=**10,n\_init**=**1,random\_state**=**3425)

model**.**fit(X)

*# This is what KMeans thought (Prediction)*

Y\_Pred**=**model**.**labels\_

**from** sklearn.metrics **import** confusion\_matrix

cm**=**confusion\_matrix(Y,Y\_Pred)

print(cm)

**from** sklearn.metrics **import** accuracy\_score

print(accuracy\_score(Y,Y\_Pred))

**from** sklearn.mixture **import** GaussianMixture

model2**=**GaussianMixture(n\_components**=**3,random\_state**=**3425)

*#Training of the model*

model2**.**fit(X)

Y\_predict2**=** model2**.**predict(X)

*#Accuracy of EM Model*

**from** sklearn.metrics **import** confusion\_matrix

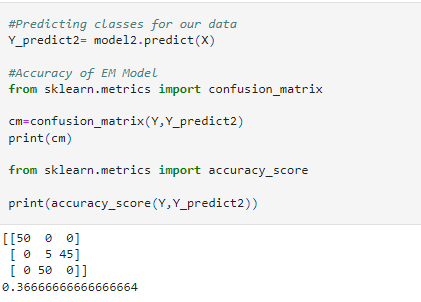
cm**=**confusion\_matrix(Y,Y\_predict2)

print(cm)

**from** sklearn.metrics **import** accuracy\_score

print(accuracy\_score(Y,Y\_predict2))





**LAB PROGRAM 9:**

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

Code:

**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('target')

print(Y)

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(X,Y,test\_size**=**0.3)

classier **=** KNeighborsClassifier(n\_neighbors**=**5)

classier**.**fit(x\_train, y\_train)

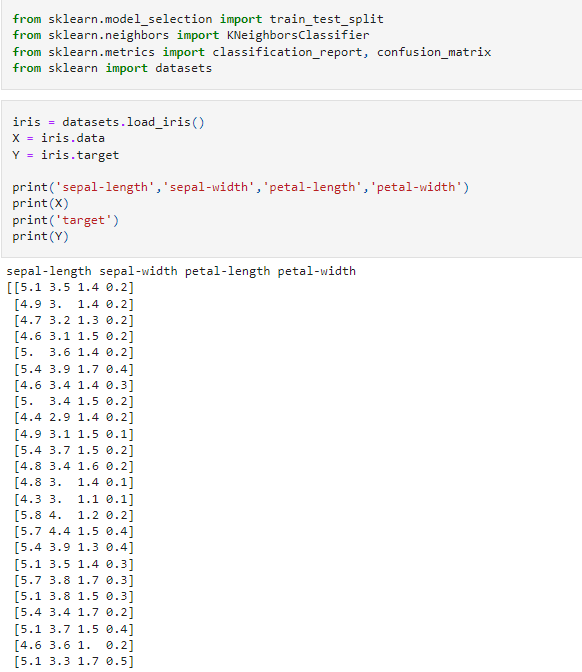
y\_pred**=**classier**.**predict(x\_test)

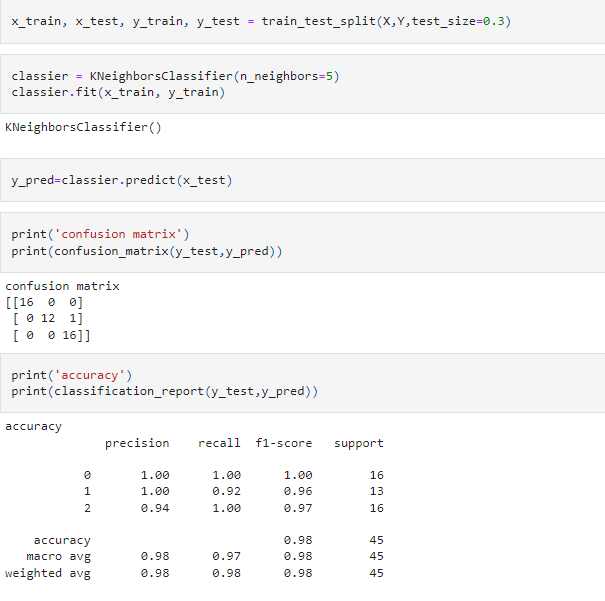
print('confusion matrix')

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

print('accuracy')

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





**LAB PROGRAM 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.

Code:

**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

data **=** pd**.**read\_csv('tips.csv')

bill **=** np1**.**array(data**.**total\_bill)

tip **=** np1**.**array(data**.**tip)

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]

one **=** np1**.**mat(np1**.**ones(m))

X**=** np1**.**hstack((one**.**T,mbill**.**T)) *# create a stack of bill from ONE*

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):

x0 **=** np**.**r\_[1, x0]

X **=** np**.**c\_[np**.**ones(len(X)), X]

xw **=** X**.**T **\*** radial\_kernel(x0, X, tau)

beta **=** np**.**linalg**.**pinv(xw **@** X) **@** xw **@** Y

**return** x0 **@** beta

**def** radial\_kernel(x0, X, tau):

**return** np**.**exp(np**.**sum((X **-** x0) **\*\*** 2, axis**=**1) **/** (**-**2 **\*** tau **\*** tau))

n **=** 1000

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])

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 **=** [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)]]))



