**INDEX**

|  |  |
| --- | --- |
| **Sr.no.** | **Practical** |
| **1** | **A]Design a simple machine learning model to train the training instances and test the**  **same.** |
|  | **B]Implement and demonstrate the FIND-S algorithm for finding the**  **most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file** |
| **2** | **A]Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.** |
|  | **B] 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.** |
| **3** | **A]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.** |
|  | **B] Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.** |
| **4** | **A]For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.** |
|  | **B] For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.** |
| **5** | **A]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.** |
|  | **B] Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.** |
| **6** | **A]Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.** |
|  | **B]Implement the classification model using clustering for the following techniques with K-means clustering with Prediction, Test Score and Confusion Matrix.** |
| **7** | **A]Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix** |
|  | **B]Implement the Rule based method and test the same.** |

|  |  |
| --- | --- |
| **8** | **A]Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.** |
|  | **B]Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.** |
| **9** | **A]Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.** |
|  | **B] Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.** |
| **10** | **A]Write A Program To Demonstrate The Working Of The Decision Tree Based ID3Algorithm. Use An Appropriate Data Set For Building The Decision Tree And Apply This Knowledge To Classify A New Sample.** |
|  | **B] Build An Artificial Neural Network By Implementing The**  **Back propagation Algorithm And Test The Same Using Appropriate Data Sets.** |

**PRACTICAL: 1**

**1A] Design a simple machine learning model to train the training instances and test the same.**

import pandas as pd

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt

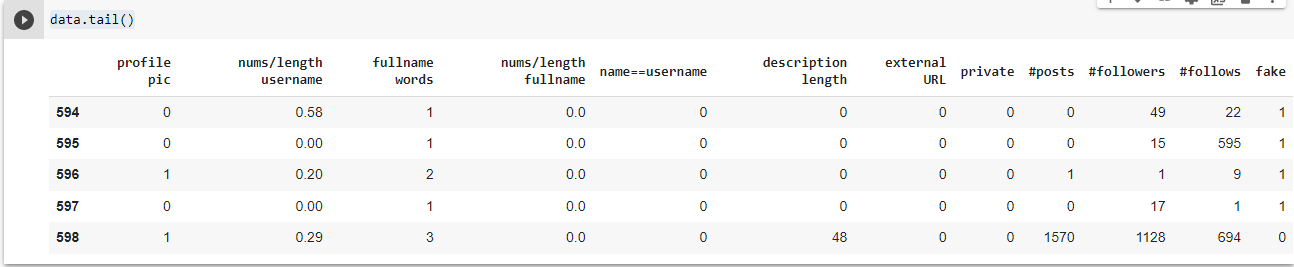
from sklearn.metrics import confusion\_matrix import seaborn as sns

from sklearn.metrics import accuracy\_score from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score from sklearn.metrics import precision\_score

from sklearn.ensemble import Random Forest Classifier

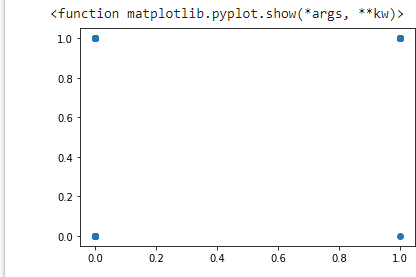
data = pd.read\_csv("/content/drive/MyDrive/excledata/instagram dataset.csv")



data.shape



plt.scatter(data["name==username"],data["fake"]) plt.show



y\_data = data[["fake"]]

x\_data = data.drop(columns={'fake'}) y\_data.head()

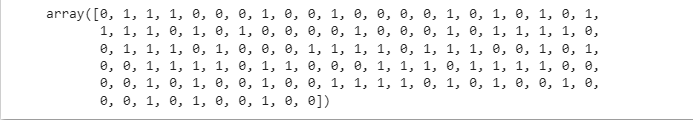


X\_train , X\_test , Y\_train , Y\_test = train\_test\_split(x\_data, y\_data, test\_size= 0.2 , random\_state

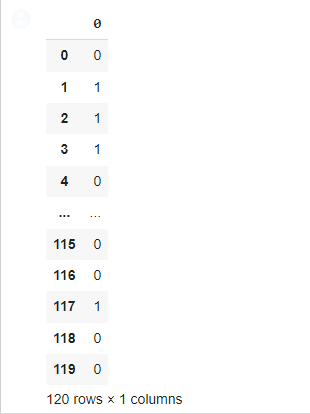
=1)

logistic\_model = LogisticRegression() logistic\_model.fit(X\_train, Y\_train) predicted = logistic\_model.predict(X\_test)

predicted

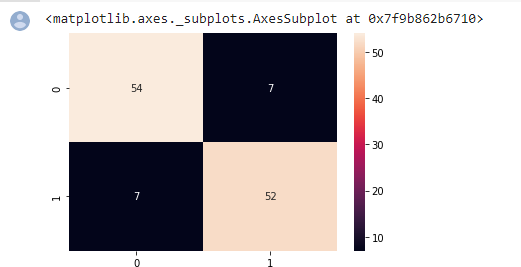


predicted\_df = pd.DataFrame(data=predicted) predicted\_df



fc = confusion\_matrix(Y\_test, predicted\_df)

sns.heatmap(fc, annot = True)



accuracy = ((fc[0,0] + fc[1,1]) /(fc[0,0]+fc[0,1]+fc[1,0]+fc[1,1]))\*100 print(round(accuracy,2))



error = 100 - accuracy print(round(error,2))



score = logistic\_model.score(X\_test, Y\_test) score\*100



score1 = accuracy\_score(Y\_test, predicted\_df) score1



print('Precision %.3f'% precision\_score(Y\_test, predicted\_df))



print('Recall\_score.%3f'% roc\_auc\_score(Y\_test, predicted\_df))



print('F1\_score.%3f'% f1\_score(Y\_test, predicted\_df))

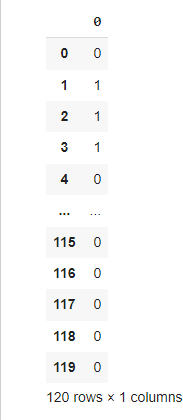


* #random forest

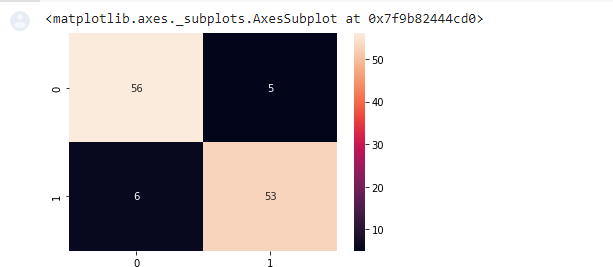
clf = RandomForestClassifier() clf.fit(X\_train, Y\_train)

predicted =clf.predict(X\_test)

predicted\_df = pd.DataFrame(data=predicted) predicted\_df



fc = confusion\_matrix(Y\_test, predicted\_df) sns.heatmap(fc, annot = True)



accuracy = ((fc[0,0] + fc[1,1]) /(fc[0,0]+fc[0,1]+fc[1,0]+fc[1,1]))\*100 print(round(accuracy,2))



**1B] Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file**

# -\*- coding: utf-8 -\*- """Find S Algo.ipynb

Automatically generated by Colaboratory.

Original file is located at

# \*\*PART1: IMPORTING PACKAGES\*\* """

Importnumpy as np import pandas as pd

"""# \*\*PART2: READING DATA\*\*"""

data = pd.read\_csv("ws.csv") data

data.shape

"""# \*\*PART3: SPLITTING X AND Y PART FROM THE DATA\*\*"""

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

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

"""# \*\*PART4: TRAINING PART\*\*"""

def train(c,t):

for i, val in enumerate(t): ifval == "Yes":

specific\_hypothesis = c[i].copy() break

for i, val in enumerate(c): if t[i] == "Yes":

for x in range(len(specific\_hypothesis)):

ifval[x] != specific\_hypothesis[x]: specific\_hypothesis[x] = '?'

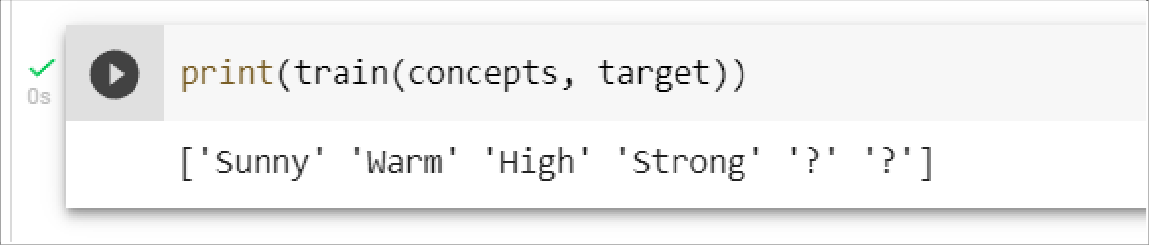
else:

pass returnspecific\_hypothesis

"""# \*\*PART 5: TESTING PART\*\*"""

print(train(concepts, target))

**Output**



**PRACTICAL: 2**

**2A] Perform Data Loading, Feature selection (Principal Component analysis) andFeature Scoring and Ranking.**

import pandas as pd

df = pd.read\_csv("/content/heart.csv") df.head()

df.shape

df.describe() df[df.Cholesterol>(df.Cholesterol.mean()+3\*df.Cholesterol.std())]

df.shape df[df.MaxHR>(df.MaxHR.mean()+3\*df.MaxHR.std())] df[df.FastingBS>(df.FastingBS.mean()+3\*df.FastingBS.std())] df[df.Oldpeak>(df.Oldpeak.mean()+3\*df.Oldpeak.std())]

df2 = df1[df1.Oldpeak<=(df1.Oldpeak.mean()+3\*df1.Oldpeak.std())] df2.shape df[df.RestingBP>(df.RestingBP.mean()+3\*df.RestingBP.std())]

df3 = df2[df2.RestingBP<=(df2.RestingBP.mean()+3\*df2.RestingBP.std())] df3.shape

df.ChestPainType.unique() df.RestingECG.unique() df.ExerciseAngina.unique() df.ST\_Slope.unique()

df4 = df3.copy() df4.ExerciseAngina.replace(

{

'N': 0,

'Y': 1

},

inplace=True)

df4.ST\_Slope.replace(

{

'Down': 1,

'Flat': 2,

'Up': 3

},

inplace=True

)

df4.RestingECG.replace(

{

'Normal': 1,

'ST': 2,

'LVH': 3

},

inplace=True)

df4.head()

df5 = pd.get\_dummies(df4, drop\_first=True) df5.head()

X = df5.drop("HeartDisease",axis='columns') y = df5.HeartDisease

1. head()

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X)

X\_scaled

from sklearn.model\_selection import train\_test\_split X\_train.shape

X\_test.shape

from sklearn.ensemble import RandomForestClassifier from sklearn.decomposition import PCA

pca = PCA(0.95)

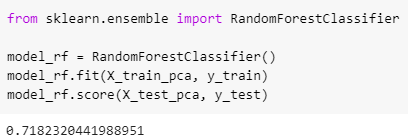
X\_pca = pca.fit\_transform(X) X\_pca

X\_train\_pca, X\_test\_pca, y\_train, y\_test = train\_test\_split(X\_pca, y, test\_size=0.2, random\_state

=30)

from sklearn.ensemble import RandomForestClassifier model\_rf = RandomForestClassifier() model\_rf.fit(X\_train\_pca, y\_train) model\_rf.score(X\_test\_pca, y\_test)

**Output**



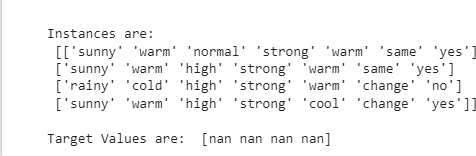
**2B] For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.**

import numpy as np import pandas as pd

data = pd.read\_csv('/content/drive/MyDrive/data set/sport.csv') concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts) target = np.array(data.iloc[:,-1]) print("\nTarget Values are: ",target)

**Output**



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] = '?'

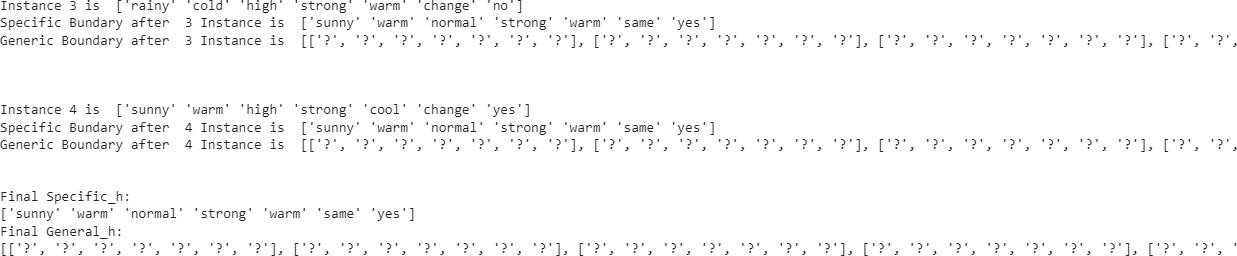
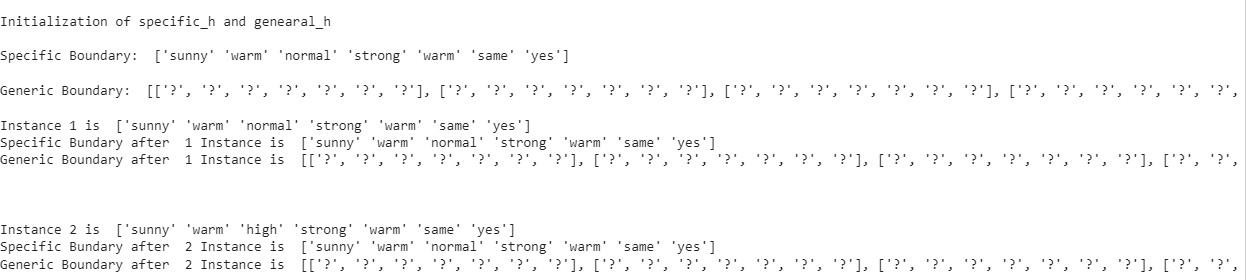
print("Specific Bundary after ", i+1, "Instance is ", specific\_h) print("Generic Boundary after ", i+1, "Instance is ", general\_h) print("\n")

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")



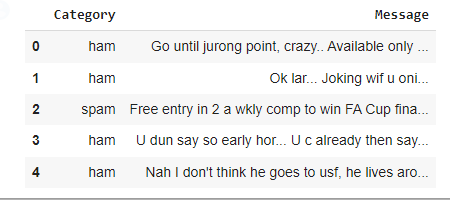
**PRACTICAL: 3**

**3A]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.**

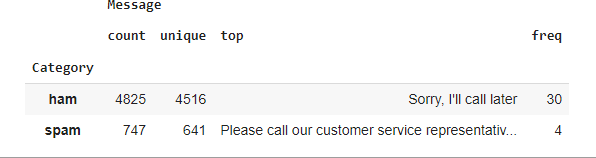
import pandas as pd

df = pd.read\_csv("spam.csv") df.head()

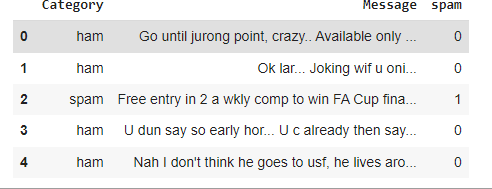
**output**



df.groupby('Category').describe()



df['spam']=df['Category'].apply(lambda x: 1 if x=='spam' else 0) df.head()



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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.Message,df.spam)

from sklearn.feature\_extraction.text import CountVectorizer v = CountVectorizer()

X\_train\_count = v.fit\_transform(X\_train.values) X\_train\_count.toarray()[:2]



from sklearn.naive\_bayes import MultinomialNB model = MultinomialNB() model.fit(X\_train\_count,y\_train)



emails = [

'Hey mohan, can we get together to watch footbal game tomorrow?',

'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'

]

emails\_count = v.transform(emails) model.predict(emails\_count)



X\_test\_count = v.transform(X\_test) model.score(X\_test\_count, y\_test)



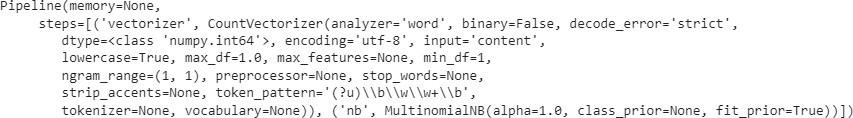
**Sklearn Pipeline**

from sklearn.pipeline import Pipeline clf = Pipeline([

('vectorizer', CountVectorizer()), ('nb', MultinomialNB())

])

clf.fit(X\_train, y\_train)



clf.score(X\_test,y\_test)



clf.predict(emails)



**3B]Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.**

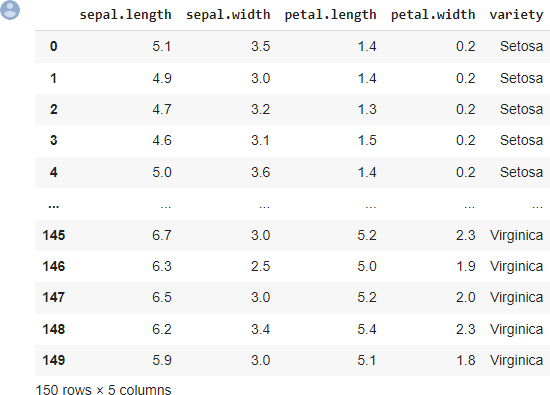
import numpy as nm

import matplotlib.pyplot as mtp import pandas as pd

from sklearn.preprocessing import LabelEncoder from sklearn import preprocessing

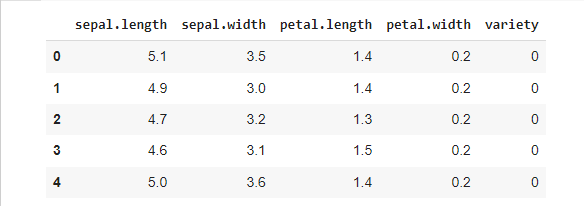
from google.colab import drive drive.mount('/content/drive') #importing datasets

data\_set= pd.read\_csv("/content/drive/MyDrive/dataset/iris.csv") data\_set



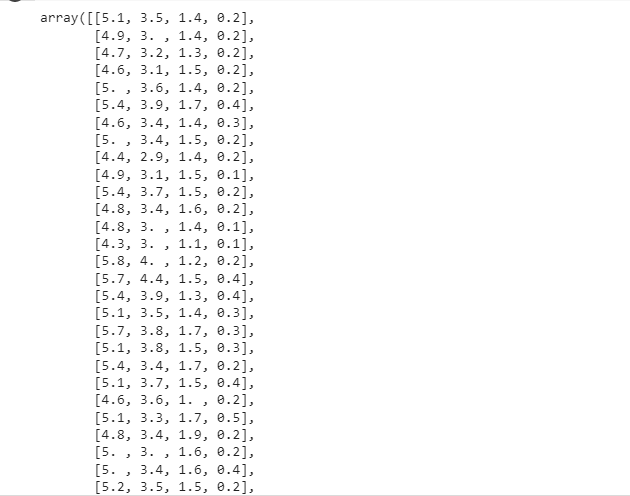
le = preprocessing.LabelEncoder()

data\_set["variety"] = le.fit\_transform(data\_set["variety"]) data\_set.head()

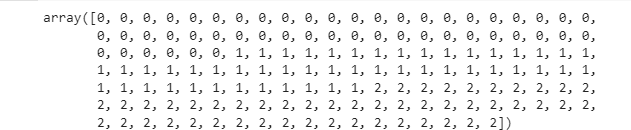


x\_data= data\_set.iloc[:, [0,1,2,3]].values y\_data= data\_set.iloc[:, 4].values

x\_data



y\_data



# Splitting the dataset into training and test set.

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

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x\_data, y\_data, test\_size= 0.20, random\_state=1) #feature Scaling

from sklearn.preprocessing import StandardScaler st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train) x\_test= st\_x.transform(x\_test)

#Fitting Decision Tree classifier to the training set from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy") classifier.fit(x\_train, y\_train)

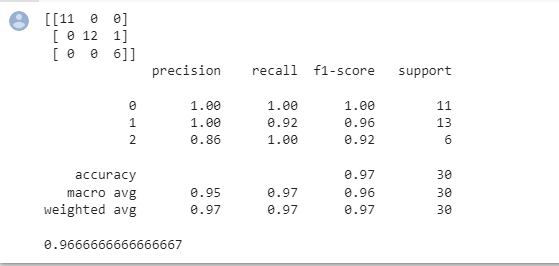
#Predicting the test set result y\_pred= classifier.predict(x\_test) y\_pred



#Creating the Confusion matrix

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

print(confusion\_matrix(y\_test,y\_pred)) print(classification\_report(y\_test,y\_pred)) print(accuracy\_score(y\_test, y\_pred))



**PRACTICAL: 4**

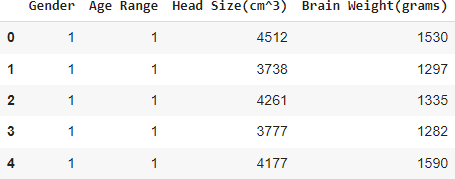
**4A]For a given set of training data examples stored in a .CSV file implement LeastSquare Regression algorithm.**

**Code:**

importnumpy as np import pandas as pd

importmatplotlib.pyplot as plt

data= pd.read\_csv('/content/drive/MyDrive/data set/pract4A.csv') data.head()



# Coomputing X and Y

X = data['Head Size(cm^3)'].values

Y = data['Brain Weight(grams)'].values # Mean X and Y

mean\_x = np.mean(X) mean\_y = np.mean(Y)

# Total number of values n = len(X)

# Using the formula to calculate 'm' and 'c' numer = 0

denom = 0

for i in range(n):

numer += (X[i] - mean\_x) \* (Y[i] - mean\_y) denom += (X[i] - mean\_x) \*\* 2

m = numer / denom

c = mean\_y - (m \* mean\_x)

# Printing coefficients print("Coefficients") print(m, c)

# Plotting Values and Regression Line max\_x = np.max(X) + 100

min\_x = np.min(X) - 100

# Calculating line values x and y

x = np.linspace(min\_x, max\_x, 1000) y = c + m \* x

# Ploting Line

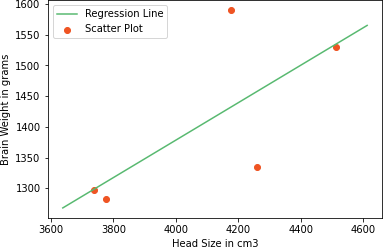
plt.plot(x, y, color='#58b970', label='Regression Line') # Ploting Scatter Points

plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')

plt.xlabel('Head Size in cm3') plt.ylabel('Brain Weight in grams') plt.legend()

plt.show()

**output:**



# Calculating Root Mean Squares Error rmse = 0

for i in range(n):

y\_pred = c + m \* X[i]

rmse += (Y[i] - y\_pred) \*\* 2 rmse = np.sqrt(rmse/n) print("RMSE")

print(rmse)



**4B] For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.**

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

%matplotlib inline

train = pd.read\_csv('titanic\_train.csv') [train.info](http://train.info/)()

sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis') def impute\_age(cols):

Age = cols[0] Pclass = cols[1]

if pd.isnull(Age): if Pclass == 1:

return 37

elif Pclass == 2: return 29

else:

return 24

else:

return Age

train['Age'] = train[['Age','Pclass']].apply(impute\_age,axis=1) sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis') train.drop('Cabin',axis=1,inplace=True)

train.dropna(inplace=True)

sex = pd.get\_dummies(train['Sex'],drop\_first=True)

embark = pd.get\_dummies(train['Embarked'],drop\_first=True) train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True) train = pd.concat([train,sex,embark],axis=1)

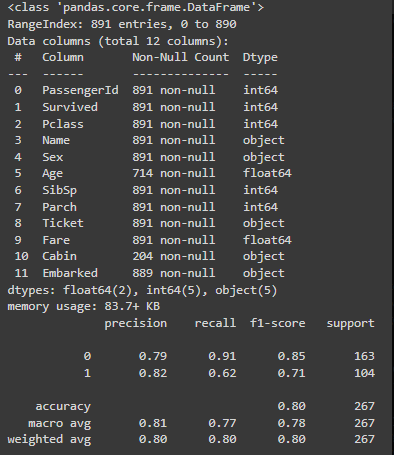
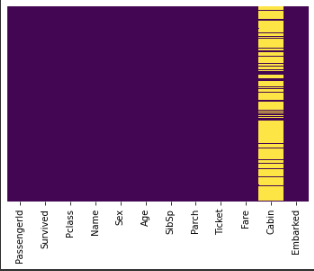
from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train.drop('Survived',axis=1),

train['Survived'], test\_size=0.30, random\_state=101) from sklearn.linear\_model import LogisticRegression

logmodel = LogisticRegression() logmodel.fit(X\_train,y\_train) predictions = logmodel.predict(X\_test)

from sklearn.metrics import classification\_report print(classification\_report(y\_test,predictions))



**PRACTICAL: 5**

**5A]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 import math

import numpy as np

data = pd.read\_csv("/content/drive/MyDrive/dataset/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)

def classify(root: Node, new): for child in root.children:

if child.value == new[root.value]: if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred) exit

else:

classify (child.children[0], new) root = ID3(data, features)

print("Decision Tree is:") printTree(root)

print (" ")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"} classify (root, new)

**Output**



**5B] Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.**

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)

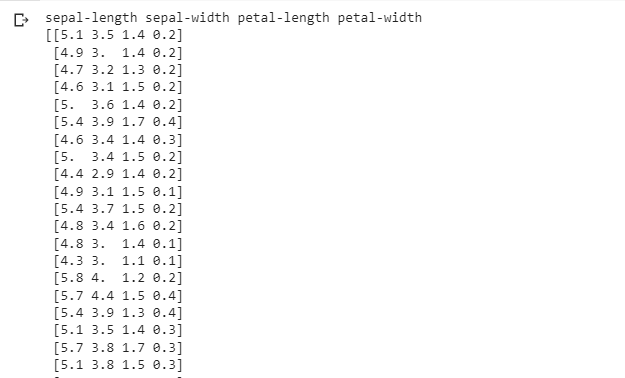
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica') print(y)

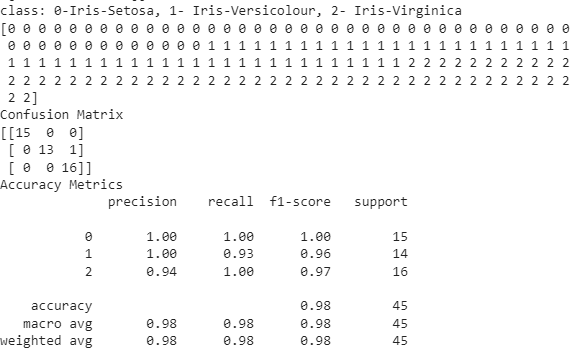
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3) #To Training the model and Nearest nighbors K=5

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))





**PRACTICAL: 6**

**6A]Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.**

fromsklearn.neighbors import KNeighborsClassifier fromsklearn.metrics import confusion\_matrix, accuracy\_score fromsklearn.model\_selection import train\_test\_split importnumpy as np

# Generate some random data for demonstration X = np.random.rand(100, 2)

y = np.random.randint(0, 2, 100)

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Initialize the KNN classifier with the Euclidean distance metric clf = KNeighborsClassifier(metric='euclidean')

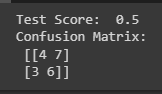
# Fit the model to the training data clf.fit(X\_train, y\_train)

# Make predictions on the test set y\_pred = clf.predict(X\_test)

# Calculate the test score (accuracy) test\_score = accuracy\_score(y\_test, y\_pred) print("Test Score: ", test\_score)

# Calculate the confusion matrix

confusion\_mat = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix: \n", confusion\_mat)



**6B]Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix**

fromsklearn.cluster import KMeans

fromsklearn.metrics import accuracy\_score, confusion\_matrix fromsklearn.model\_selection import train\_test\_split importnumpy as np

# Generate some random data for demonstration X = np.random.rand(100, 2)

y = np.random.randint(0, 2, 100)

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) # Initialize the K-means clustering model

kmeans = KMeans(n\_clusters=2) # Fit the model to the training data kmeans.fit(X\_train)

# Assign labels to the test data based on the cluster centers y\_pred = kmeans.predict(X\_test)

# Create a mapping from the cluster labels to the original labels

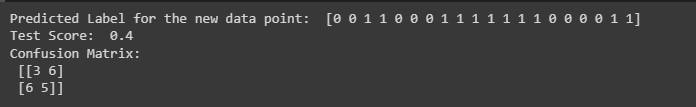
# This step is needed because the k-means algorithm does not guarantee # that the clusters will correspond to the original labels

label\_map = dict(zip(np.unique(y\_train), np.unique(y))) y\_pred = np.array([label\_map[label] for label in y\_pred]) print("Predicted Label for the new data point: ", y\_pred ) # Calculate the test score (accuracy)

test\_score = accuracy\_score(y\_test, y\_pred) print("Test Score: ", test\_score)

# Calculate the confusion matrix

confusion\_mat = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix: \n", confusion\_mat)



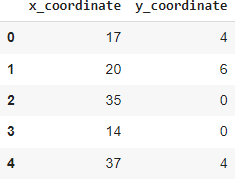
**PRACTICAL: 7**

**7A]Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix**

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

comic\_con = pd.read\_csv('/content/CLUSTERING.csv', index\_col=0) comic\_con.head()



from google.colab import drive drive.mount('/content/drive')



from scipy.cluster.vq import whiten

comic\_con['x\_scaled'] = whiten(comic\_con['x\_coordinate']) comic\_con['y\_scaled'] = whiten(comic\_con['y\_coordinate'])

from scipy.cluster.hierarchy import linkage, fcluster # Use the linkage()

distance\_matrix = linkage(comic\_con[['x\_scaled', 'y\_scaled']], method='ward', metric='euclidean'

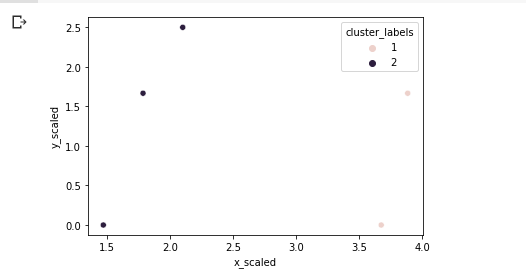
)

# Assign cluster labels

comic\_con['cluster\_labels'] = fcluster(distance\_matrix, 2, criterion='maxclust')

# Plot clusters

sns.scatterplot(x='x\_scaled', y='y\_scaled', hue='cluster\_labels', data=comic\_con);



# Use the linkage()

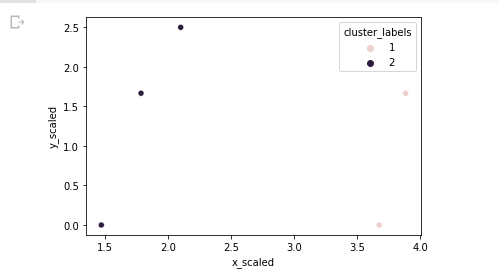
distance\_matrix = linkage(comic\_con[['x\_scaled', 'y\_scaled']], method='single', metric='euclidea n')

# Assign cluster labels

comic\_con['cluster\_labels'] = fcluster(distance\_matrix, 2, criterion='maxclust')

# Plot clusters

sns.scatterplot(x='x\_scaled', y='y\_scaled', hue='cluster\_labels', data=comic\_con);



# Use the linkage()

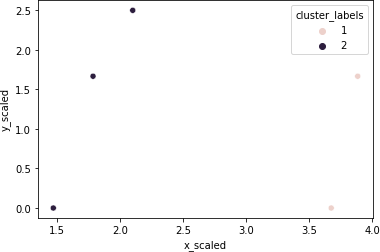
distance\_matrix = linkage(comic\_con[['x\_scaled', 'y\_scaled']], method='complete', metric='euclid ean')

# Assign cluster labels

comic\_con['cluster\_labels'] = fcluster(distance\_matrix, 2, criterion='maxclust')

# Plot clusters

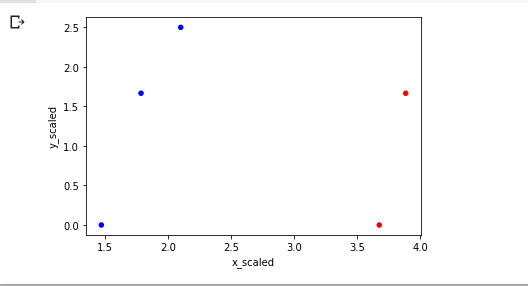
sns.scatterplot(x='x\_scaled', y='y\_scaled', hue='cluster\_labels', data=comic\_con);



# Define a colors dictionary for clusters colors = {1:'red', 2:'blue'}

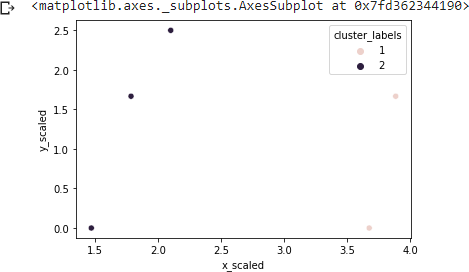
# Plot the scatter plot

comic\_con.plot.scatter(x='x\_scaled', y='y\_scaled', c=comic\_con['cluster\_labels'].apply(lambda x: colors[x]));



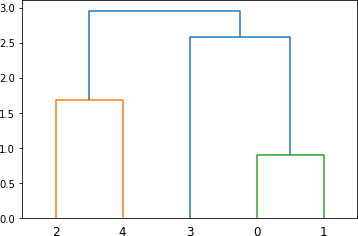
# Plot a scatter plot using seaborn

sns.scatterplot(x='x\_scaled', y='y\_scaled', hue='cluster\_labels', data=comic\_con)



from scipy.cluster.hierarchy import dendrogram # Create a dendrogram

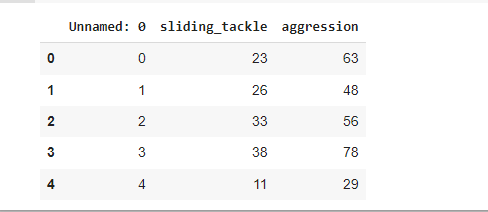
dn = dendrogram(distance\_matrix)



%timeit linkage(comic\_con[['x\_scaled', 'y\_scaled']], method='ward', metric='euclidean')



fifa = pd.read\_csv('/content/drive/MyDrive/data set/fifa18.csv') fifa.head()



fifa['scaled\_sliding\_tackle'] = whiten(fifa['sliding\_tackle']) fifa['scaled\_aggression'] = whiten(fifa['aggression'])

# Fit the data into a hierarchical cluster

distance\_matrix = linkage(fifa[['scaled\_sliding\_tackle', 'scaled\_aggression']], method='ward')

# Assign cluster labels to each row of data

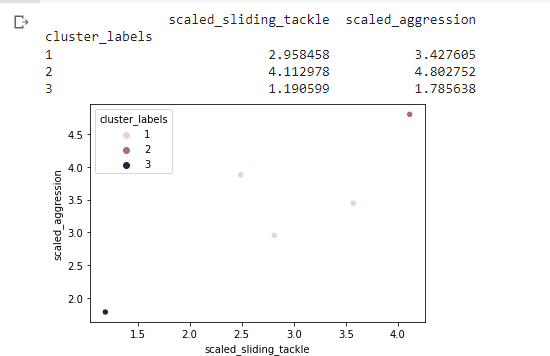
fifa['cluster\_labels'] = fcluster(distance\_matrix, 3, criterion='maxclust')

# Display cluster centers of each cluster

print(fifa[['scaled\_sliding\_tackle', 'scaled\_aggression', 'cluster\_labels']].groupby('cluster\_labels'). mean())

# Create a scatter plot through seaborn

sns.scatterplot(x='scaled\_sliding\_tackle', y='scaled\_aggression', hue='cluster\_labels', data=fifa) plt.savefig('/content/drive/MyDrive/data set/fifa\_cluster.png')



**7B]Implement the Rule based method and test the same.**

from sklearn.metrics import accuracy\_score, confusion\_matrix from sklearn.model\_selection import train\_test\_split

import numpy as np

# Generate some random data for demonstration X = np.random.rand(100, 2)

y = np.random.randint(0, 2, 100)

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Define the rule-based classifier defrule\_based\_classifier(x):

if x[0] > x[1]: return 0

else:

return 1

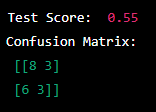
# Make predictions on the test set using the rule-based classifier y\_pred = np.array([rule\_based\_classifier(x) for x in X\_test])

# Calculate the test score (accuracy) test\_score = accuracy\_score(y\_test, y\_pred) print("Test Score: ", test\_score)

# Calculate the confusion matrix

confusion\_mat = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix: \n", confusion\_mat)

**Output**



**PRACTICAL: 8**

**8A]Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease**

**Data Set.**

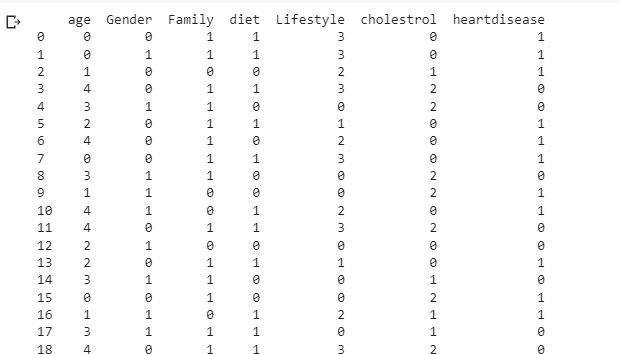
pip install pgmpy import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

data = pd.read\_csv("/content/drive/MyDrive/ml dataset/DS4.csv") heart\_disease = pd.DataFrame(data)

print(heart\_disease)



model = BayesianModel([ ('age', 'Lifestyle'),

('Gender', 'Lifestyle'),

('Family', 'heartdisease'),

('diet', 'cholestrol'),

('Lifestyle', 'diet'), ('cholestrol', 'heartdisease'), ('diet', 'cholestrol')

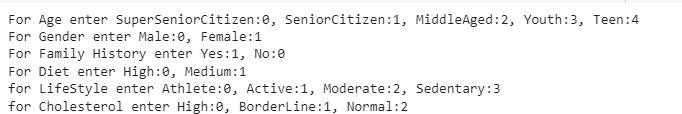
])

model.fit(heart\_disease, estimator=MaximumLikelihoodEstimator) HeartDisease\_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4') print('For Gender enter Male:0, Female:1')

print('For Family History enter Yes:1, No:0') print('For Diet enter High:0, Medium:1')

print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3') print('for Cholesterol enter High:0, BorderLine:1, Normal:2')



q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={ 'age': int(input('Enter Age: ')),

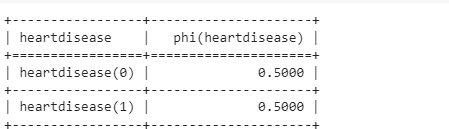
'Gender': int(input('Enter Gender: ')), 'Family': int(input('Enter Family History: ')), 'diet': int(input('Enter Diet: ')),

'Lifestyle': int(input('Enter Lifestyle: ')), 'cholestrol': int(input('Enter Cholestrol: '))

})



print(q)



**8B]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 sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error from sklearn.model\_selection import train\_test\_split import numpy as np

import matplotlib.pyplot as plt

# Generate some random data for demonstration np.random.seed(0)

X = np.random.rand(100, 1)

y = np.sin(2 \* np.pi \* X) + np.random.randn(100, 1) \* 0.1

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Initialize the LWR model

lwr = LinearRegression(fit\_intercept=True)

# Define the bandwidth parameter bandwidth = 0.2

# Fit the model to the training data predictions = []

for x in X\_test:

weights = np.exp(-((X\_train - x) \*\* 2) / (2 \* bandwidth \*\* 2)) lwr.fit(X\_train, y\_train, sample\_weight=weights) predictions.append(lwr.predict([x]))

# Calculate the test MSE

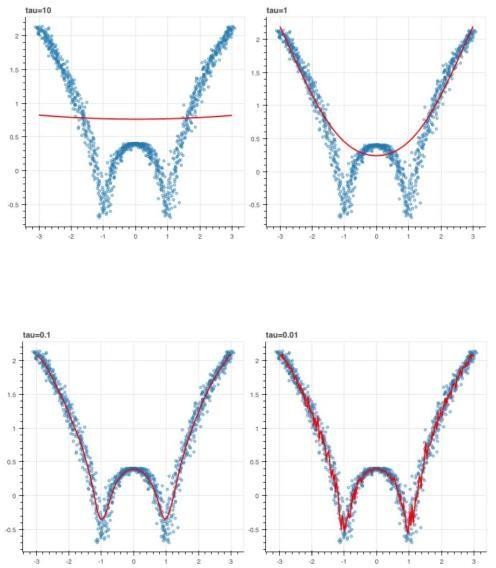
test\_mse = mean\_squared\_error(y\_test, predictions) print("Test MSE: ", test\_mse)

# Plot the data points and the LWR model plt.scatter(X\_train, y\_train, label='Training Data') plt.scatter(X\_test, y\_test, label='Test Data') plt.plot(X\_test, predictions, color='r', label='LWR Model') plt.legend()

plt.show()

**Output**





**PRACTICAL: 9**

**9A]Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function def sigmoid (x):

return 1/(1 + np.exp(-x)) #Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x) #Variable initialization

epoch=5000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer #weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y for i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp) #Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

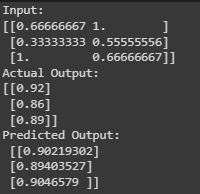
EH = d\_output.dot(wout.T)

#how much hidden layer wts contributed to error hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop

wout += hlayer\_act.T.dot(d\_output) \*lr wh += X.T.dot(d\_hiddenlayer) \*lr print("Input: \n" + str(X)) print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

**Output**



**9B] Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.**

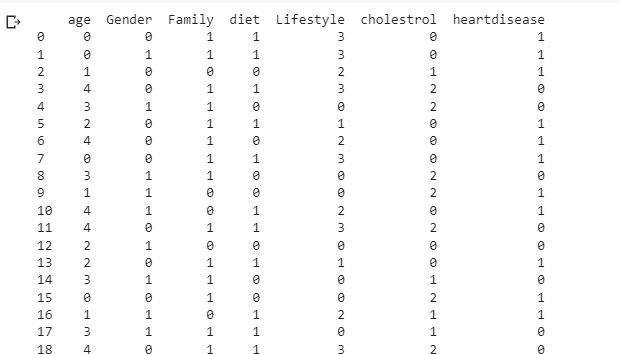
pip install pgmpy import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

data = pd.read\_csv("/content/drive/MyDrive/ml dataset/DS4.csv") heart\_disease = pd.DataFrame(data)

print(heart\_disease)



model = BayesianModel([ ('age', 'Lifestyle'),

('Gender', 'Lifestyle'),

('Family', 'heartdisease'),

('diet', 'cholestrol'),

('Lifestyle', 'diet'), ('cholestrol', 'heartdisease'), ('diet', 'cholestrol')

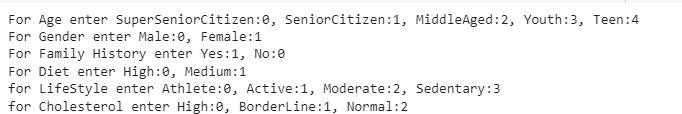
])

model.fit(heart\_disease, estimator=MaximumLikelihoodEstimator) HeartDisease\_infer = VariableElimination(model)

print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4') print('For Gender enter Male:0, Female:1')

print('For Family History enter Yes:1, No:0') print('For Diet enter High:0, Medium:1')

print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3') print('for Cholesterol enter High:0, BorderLine:1, Normal:2')



q = HeartDisease\_infer.query(variables=['heartdisease'], evidence={ 'age': int(input('Enter Age: ')),

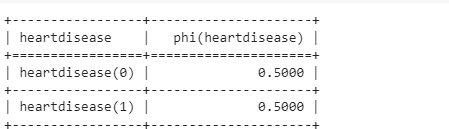
'Gender': int(input('Enter Gender: ')), 'Family': int(input('Enter Family History: ')), 'diet': int(input('Enter Diet: ')),

'Lifestyle': int(input('Enter Lifestyle: ')), 'cholestrol': int(input('Enter Cholestrol: '))

})



print(q)



**PRACTICAL: 10**

**10A] 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 import math

import numpy as np

data = pd.read\_csv("/content/drive/MyDrive/dataset/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)

def classify(root: Node, new): for child in root.children:

if child.value == new[root.value]: if child.isLeaf:

print ("Predicted Label for new example", new," is:", child.pred) exit

else:

classify (child.children[0], new) root = ID3(data, features)

print("Decision Tree is:") printTree(root)

print (" ")

new = {"outlook":"sunny", "temperature":"hot", "humidity":"normal", "wind":"strong"} classify (root, new)

**Output**



**10B] Build An Artificial Neural Network By Implementing The Backpropagation Algorithm And Test The Same Using Appropriate Data Sets.**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) # maximum of X array longitudinally y = y/100

#Sigmoid Function def sigmoid (x):

return 1/(1 + np.exp(-x)) #Derivative of Sigmoid Function def derivatives\_sigmoid(x):

return x \* (1 - x) #Variable initialization

epoch=5000 #Setting training iterations lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set hiddenlayer\_neurons = 3 #number of hidden layers neurons output\_neurons = 1 #number of neurons at output layer #weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons)) bh=np.random.uniform(size=(1,hiddenlayer\_neurons)) wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons)) bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y for i in range(epoch):

#Forward Propogation hinp1=np.dot(X,wh) hinp=hinp1 + bh hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout) outinp= outinp1+ bout

output = sigmoid(outinp) #Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output) d\_output = EO\* outgrad

EH = d\_output.dot(wout.T)

#how much hidden layer wts contributed to error hiddengrad = derivatives\_sigmoid(hlayer\_act) d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop wout += hlayer\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hiddenlayer) \*lr print("Input: \n" + str(X)) print("Actual Output: \n" + str(y)) print("Predicted Output: \n" ,output)

**Output**

