1)Find\_S

Code:

import csv

def loadCsv(filename):

  lines = csv.reader(open(filename, "r"))

  dataset = list(lines)

  for i in range(len(dataset)):

    dataset[i] = dataset[i]

  return dataset

attributes = ['Sky','Temp','Humidity','Wind','Water','Forecast']

print('Attributes =',attributes)

num\_attributes = len(attributes)

filename = "ENJOYSPORT.csv"

dataset = loadCsv(filename)

print(dataset)

hypothesis=['0'] \* num\_attributes

print("Intial Hypothesis")

print(hypothesis)

print("The Hypothesis are")

for i in range(1,len(dataset)):

  target = dataset[i][-1]

  if(target == '1'):

    for j in range(num\_attributes):

        if(hypothesis[j]=='0'):

          hypothesis[j] = dataset[i][j]

        if(hypothesis[j]!= dataset[i][j]):

          hypothesis[j]='?'

  print(i+1,'=',hypothesis)

print("Final Hypothesis")

print(hypothesis)

Output:

[['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'EnjoySport'], ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', '1'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', '1'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', '0'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', '1']]

Intial Hypothesis

['0', '0', '0', '0', '0', '0']

The Hypothesis are

2 = ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

3 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

4 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

5 = ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final Hypothesis

['Sunny', 'Warm', '?', 'Strong', '?', '?']

2)Candidate\_elimination:

import numpy as np

import pandas as pd

data = pd.read\_csv('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(l

en(specific\_h))]

    print("\nGeneric Boundary: ",general\_h)

    for i, h in enumerate(concepts):

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

        if target[i] == 1:

            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] == 0:

            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:

Instances are:

[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]

Target Values are: [1 1 0 1]

Initialization of specific\_h and genearal\_h

Specific Boundary: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

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

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Bundaryafter 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']

Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']

Instance is Positive

Specific Bundaryafter 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance 3 is ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']

Instance is Negative

Specific Bundaryafter 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']

Generic Boundary after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']

Instance is Positive

Specific Bundaryafter 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']

Generic Boundary after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific\_h:

['Sunny' 'Warm' '?' 'Strong' '?' '?']

Final General\_h:

[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

**3) 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

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.metrics import accuracy\_score

# Assuming the CSV file is named 'play\_tennis.csv' and it's in the current working directory

data = pd.read\_csv('/content/play\_tennis (1).csv')

# Display the first 5 rows of the data

print("The first 5 values of data is:\n", data.head())

# Prepare the features (X) and target (y)

X = data.iloc[:, :-1] # All rows, all columns except the last one

y = data.iloc[:, -1] # All rows, only the last column

# Display the first 5 rows of features and target

print("\nThe first 5 values of train data is:\n", X.head())

print("\nThe first 5 values of train output is:\n", y.head())

# Use .apply to transform all categorical columns to numeric using LabelEncoder

X\_encoded = X.apply(LabelEncoder().fit\_transform)

y\_encoded = LabelEncoder().fit\_transform(y)

# Display the transformed training data

print("\nNow the train data is:\n", X\_encoded.head())

print("\nNow the train output is:\n", y\_encoded)

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y\_encoded, test\_size=0.20, random\_state=42)

# Fitting Gaussian Naive Bayes to the Training set

classifier = GaussianNB()

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

# Predicting the Test set results

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

# Calculating the accuracy of the model

accuracy = accuracy\_score(y\_test, y\_pred)

print("\nAccuracy is:", accuracy)

**output**

The first 5 values of data is:

day outlook temp humidity wind play

0 D1 Sunny Hot High Weak No

1 D2 Sunny Hot High Strong No

2 D3 Overcast Hot High Weak Yes

3 D4 Rain Mild High Weak Yes

4 D5 Rain Cool Normal Weak Yes

The first 5 values of train data is:

day outlook temp humidity wind

0 D1 Sunny Hot High Weak

1 D2 Sunny Hot High Strong

2 D3 Overcast Hot High Weak

3 D4 Rain Mild High Weak

4 D5 Rain Cool Normal Weak

The first 5 values of train output is:

0 No

1 No

2 Yes

3 Yes

4 Yes

Name: play, dtype: object

Now the train data is:

day outlook temp humidity wind

0 0 2 1 0 1

1 6 2 1 0 0

2 7 0 1 0 1

3 8 1 2 0 1

4 9 1 0 1 1

Now the train output is:

[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

Accuracy is: 0.6666666666666666

**4. Assuming a set of documents that need to be classified, use the naïve Bayesian classifier model to perform this task. Built-in Java classes /API can be used to write the program. Calculate the accuracy precision and recall for your data set.**

import pandas as pd

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

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn import metrics

data = pd.read\_csv('text\_classification.csv', names=['message', 'label'])

print("The dimensions of the dataset:", data.shape)

data['labelnum'] = data.label.map({'pos': 1, 'neg': 0})

X = data.message

Y = data.labelnum

xtrain, xtest, ytrain, ytest = train\_test\_split(X, Y)

print('\nThe Total number of Training data:', ytrain.shape)

print('\nThe Total number of Test data:', ytest.shape)

cv = CountVectorizer()

xtrain\_dtm = cv.fit\_transform(xtrain)

xtest\_dtm = cv.transform(xtest)

print('\nThe words or Tokens in the text documents:\n')

print(cv.get\_feature\_names\_out())

df = pd.DataFrame(xtrain\_dtm.toarray(), columns=cv.get\_feature\_names\_out())

clf = MultinomialNB().fit(xtrain\_dtm, ytrain)

predicted = clf.predict(xtest\_dtm)

print('\nAccuracy of the classifier is:', metrics.accuracy\_score(ytest, predicted))

print('\nConfusion Matrix:')

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

print('\nThe Value of precision:', metrics.precision\_score(ytest, predicted))

print('\nThe Value of the recall:', metrics.recall\_score(ytest, predicted))

**Output**

The dimensions of the dataset: (18, 2)

The Total number of Training data: (13,)

The Total number of Test data: (5,)

The words or Tokens in the text documents:

['about' 'am' 'amazing' 'an' 'beers' 'best' 'boss' 'can' 'deal' 'do'

'donot' 'enemy' 'feel' 'fun' 'good' 'great' 'have' 'he' 'holiday'

'horrible' 'house' 'is' 'juice' 'like' 'love' 'my' 'not' 'of' 'place'

'restaurant' 'sandwich' 'stuff' 'sworn' 'taste' 'the' 'these' 'this'

'tired' 'to' 'today' 'tomorrow' 'very' 'we' 'went' 'what' 'will' 'with'

'work']

Accuracy of the classifier is: 1.0

Confusion Matrix:

[[2 0]

[0 3]]

The Value of precision: 1.0

The Value of the recall: 1.0

**5. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis the of heart patients using standard heart disease data set. You can use Java or Python ML Library classes /API.**

pip install pgmpy

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianNetwork

from pgmpy.inference import VariableElimination

data=pd.read\_csv('heart.csv')

print(data)

data=data.replace('?',np.nan)

print('Sample instance from the Dataset are given below')

print(data.head())

print('\n Attributes and datatypes')

print(data.dtypes)

model=BayesianNetwork([('age','target'),('sex','target'),('exang','target'),('cp','target'),('target','restecg'),('target','chol')])

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(data,estimator=MaximumLikelihoodEstimator)

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

HeartDiseasetest\_infer=VariableElimination(model)

print('\n 1.Probability of Heart Disease given evidence = restecg :1')

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

print(q1)

print('\n 2.Probability of Heart Disease given evidence = cp:2')

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

print(q2)

**Output**

age sex cp trestbps chol fbs restecg thalach exang oldpeak \

0 63 1 3 145 233 1 0 150 0 2.3

1 37 1 2 130 250 0 1 187 0 3.5

2 41 0 1 130 204 0 0 172 0 1.4

3 56 1 1 120 236 0 1 178 0 0.8

4 57 0 0 120 354 0 1 163 1 0.6

.. ... ... .. ... ... ... ... ... ... ...

298 57 0 0 140 241 0 1 123 1 0.2

299 45 1 3 110 264 0 1 132 0 1.2

300 68 1 0 144 193 1 1 141 0 3.4

301 57 1 0 130 131 0 1 115 1 1.2

302 57 0 1 130 236 0 0 174 0 0.0

slope ca thal target

0 0 0 1 1

1 0 0 2 1

2 2 0 2 1

3 2 0 2 1

4 2 0 2 1

.. ... .. ... ...

298 1 0 3 0

299 1 0 3 0

300 1 2 3 0

301 1 1 3 0

302 1 1 2 0

[303 rows x 14 columns]

Sample instance from the Dataset are given below

age sex cp trestbps chol fbs restecg thalach exang oldpeak slope \

0 63 1 3 145 233 1 0 150 0 2.3 0

1 37 1 2 130 250 0 1 187 0 3.5 0

2 41 0 1 130 204 0 0 172 0 1.4 2

3 56 1 1 120 236 0 1 178 0 0.8 2

4 57 0 0 120 354 0 1 163 1 0.6 2

ca thal target

0 0 1 1

1 0 2 1

2 0 2 1

3 0 2 1

4 0 2 1

Attributes and datatypes

age int64

sex int64

cp int64

trestbps int64

chol int64

fbs int64

restecg int64

thalach int64

exang int64

oldpeak float64

slope int64

ca int64

thal int64

target int64

dtype: object

Learning CPD using Maximum likelihood estimators

Interfering with Bayesian Network:

1.Probability of Heart Disease given evidence = restecg :1

+-----------+---------------+

| target | phi(target) |

+===========+===============+

| target(0) | 0.4242 |

+-----------+---------------+

| target(1) | 0.5758 |

+-----------+---------------+

2.Probability of Heart Disease given evidence = cp:2

+-----------+---------------+

| target | phi(target) |

+===========+===============+

| target(0) | 0.3755 |

+-----------+---------------+

| target(1) | 0.6245 |

+-----------+---------------+

**6. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

import pandas as pd

from collections import Counter

def ID3(examples, target\_attribute, attributes):

root = {}

if all(examples[target\_attribute] == 'Yes'):

return {'label': 'yes'}

elif all(examples[target\_attribute] == 'No'):

return {'label': 'no'}

if not attributes:

most\_common\_value = Counter(examples[target\_attribute]).most\_common(1)[0][0]

return {'label': most\_common\_value}

best\_attribute = None

best\_info\_gain = float('-inf')

for attribute in attributes:

info\_gain = calculate\_information\_gain(examples, target\_attribute, attribute)

if info\_gain > best\_info\_gain:

best\_info\_gain = info\_gain

best\_attribute = attribute

root['decision\_attribute'] = best\_attribute

for value in df[best\_attribute].unique():

subset\_examples = examples[examples[best\_attribute] == value]

if subset\_examples.empty:

most\_common\_value = Counter(examples[target\_attribute]).most\_common(1)[0][0]

root[value] = {'label': most\_common\_value}

else:

root[value] = ID3(subset\_examples, target\_attribute, [attr for attr in attributes if attr != best\_attribute])

return root

def gain(examples, target\_attribute, attribute):

entropy\_parent = calculate\_entropy(examples[target\_attribute])

entropy\_children = 0

for value in examples[attribute].unique():

subset\_examples = examples[examples[attribute] == value]

entropy\_children += len(subset\_examples) / len(examples) \* calculate\_entropy(subset\_examples[target\_attribute])

information\_gain = entropy\_parent - entropy\_children

return information\_gain

def calentropy(attribute\_values):

entropy = 0

total\_count = len(attribute\_values)

value\_counts = attribute\_values.value\_counts()

for count in value\_counts:

probability = count / total\_count

entropy -= probability \* np.log2(probability)

return entropy

dataset = pd.read\_csv('play\_tennis.csv')

target\_attribute = 'play'

attributes = ['outlook', 'temp', 'humidity', 'wind']

decision\_tree = ID3(dataset, target\_attribute, attributes)

def print\_decision\_tree(tree, indent=' '):

if 'label' in tree:

print(indent + str(tree['label']))

else:

decision\_attribute = tree['decision\_attribute']

print(indent + decision\_attribute)

for value, subtree in tree.items():

if value != 'decision\_attribute':

print(indent + '| ' + value)

print\_decision\_tree(subtree, indent + '| ')

print\_decision\_tree(decision\_tree)

**Output**

outlook

| Sunny

| humidity

| | High

| | no

| | Normal

| | yes

| Overcast

| yes

| Rain

| wind

| | Weak

| | yes

| | Strong

| | no

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

import pandas as pd

import numpy as np

df = pd.read\_csv('NN.csv')

X = df[['x', 'y']].values

y = df[['target']].values

X = X/np.amax(X, axis=0)

y = y/100

class Neural\_Network(object):

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

self.inputSize = 2

self.outputSize = 1

self.hiddenSize = 3

self.W1 = np.random.randn(self.inputSize, self.hiddenSize)

self.W2 = np.random.randn(self.hiddenSize, self.outputSize)

def forward(self, X):

self.z = np.dot(X, self.W1)

self.z2 = self.sigmoid(self.z)

self.z3 = np.dot(self.z2, self.W2)

o = self.sigmoid(self.z3)

return o

def sigmoid(self, s):

return 1/(1+np.exp(-s))

def sigmoidPrime(self, s):

return s \* (1 - s)

def backward(self, X, y, o):

self.o\_error = y - o

self.o\_delta = self.o\_error\*self.sigmoidPrime(o)

self.z2\_error = self.o\_delta.dot(self.W2.T)

self.z2\_delta = self.z2\_error\*self.sigmoidPrime(self.z2)

self.W1 += X.T.dot(self.z2\_delta)

self.W2 += self.z2.T.dot(self.o\_delta)

def train (self, X, y):

o = self.forward(X)

self.backward(X, y, o)

NN = Neural\_Network()

print ("\nInput: \n" + str(X))

print ("\nActual Output: \n" + str(y))

print ("\nPredicted Output: \n" + str(NN.forward(X)))

print ("\nLoss: \n" + str(np.mean(np.square(y - NN.forward(X)))))

NN.train(X, y)

**Output**

Input:

[[0.66666667 1. ]

[0.33333333 0.55555556]

[1. 0.66666667]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

Predicted Output:

[[0.58546269]

[0.58278121]

[0.57204801]]

Loss:

0.09661964449696675

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

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

from sklearn import preprocessing

from sklearn.mixture import GaussianMixture

# Load the iris dataset

iris = datasets.load\_iris()

X = pd.DataFrame(iris.data, columns=['Sepal\_Length', 'Sepal\_Width', 'Petal\_Length', 'Petal\_Width'])

y = pd.DataFrame(iris.target, columns=['Targets'])

# K-Means Clustering

model = KMeans(n\_clusters=3)

model.fit(X)

# Plotting

plt.figure(figsize=(14, 14))

colormap = np.array(['red', 'lime', 'black'])

# Real Clusters

plt.subplot(2, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Clusters')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# K-Means Clustering

plt.subplot(2, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K-Means Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# Standardize data

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns=X.columns)

# Gaussian Mixture Model Clustering

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

gmm\_y = gmm.predict(xs)

# GMM Clustering

plt.subplot(2, 2, 3)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[gmm\_y], s=40)

plt.title('GMM Clustering')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

print('Observation: The GMM using EM algorithm based clustering matched the true labels more closely than the Kmeans.')

plt.show()

**9. Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.**

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

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

iris = datasets.load\_iris()

print("Iris Data set loaded...")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.1, random\_state=42)

print("Dataset is split into training and testing...")

print("Size of training data and its label", x\_train.shape, y\_train.shape)

print("Size of testing data and its label", x\_test.shape, y\_test.shape)

for i in range(len(iris.target\_names)):

print("Label", i, "-", str(iris.target\_names[i])) # Create object of KNN classifier

Classifier = KNeighborsClassifier(n\_neighbors=1)

Classifier.fit(x\_train, y\_train) # Perform testing

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

# Display the results

print("Results of Classification using K-nn with K=1")

for r in range(0, len(x\_test)):

print("Sample:", str(x\_test[r]), "Actual-label:", str(y\_test[r]), "Predicted-label:", str(y\_pred[r]))

print("Classification Accuracy", Classifier.score(x\_test, y\_test))

# Evaluate the model

print("Confusion Matrix")

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

print("Accuracy Metrics")

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

**Output:**

Iris Data set loaded...

Dataset is split into training and testing...

Size of training data and its label (135, 4) (135,)

Size of testing data and its label (15, 4) (15,)

Label 0 - setosa

Label 1 - versicolor

Label 2 - virginica

Results of Classification using K-nn with K=1

Sample: [6.1 2.8 4.7 1.2] Actual-label: 1 Predicted-label: 1

Sample: [5.7 3.8 1.7 0.3] Actual-label: 0 Predicted-label: 0

Sample: [7.7 2.6 6.9 2.3] Actual-label: 2 Predicted-label: 2

Sample: [6. 2.9 4.5 1.5] Actual-label: 1 Predicted-label: 1

Sample: [6.8 2.8 4.8 1.4] Actual-label: 1 Predicted-label: 1

Sample: [5.4 3.4 1.5 0.4] Actual-label: 0 Predicted-label: 0

Sample: [5.6 2.9 3.6 1.3] Actual-label: 1 Predicted-label: 1

Sample: [6.9 3.1 5.1 2.3] Actual-label: 2 Predicted-label: 2

Sample: [6.2 2.2 4.5 1.5] Actual-label: 1 Predicted-label: 1

Sample: [5.8 2.7 3.9 1.2] Actual-label: 1 Predicted-label: 1

Sample: [6.5 3.2 5.1 2. ] Actual-label: 2 Predicted-label: 2

Sample: [4.8 3. 1.4 0.1] Actual-label: 0 Predicted-label: 0

Sample: [5.5 3.5 1.3 0.2] Actual-label: 0 Predicted-label: 0

Sample: [4.9 3.1 1.5 0.1] Actual-label: 0 Predicted-label: 0

Sample: [5.1 3.8 1.5 0.3] Actual-label: 0 Predicted-label: 0

Classification Accuracy 1.0

Confusion Matrix

[[6 0 0]

[0 6 0]

[0 0 3]]

Accuracy Metrics

precision recall f1-score support

0 1.00 1.00 1.00 6

1 1.00 1.00 1.00 6

2 1.00 1.00 1.00 3

accuracy 1.00 15

macro avg 1.00 1.00 1.00 15

weighted avg 1.00 1.00 1.00 15

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

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Kernel function to calculate weights

def kernel(point, xmat, k):

m, n = np.shape(xmat)

weights = np.mat(np.eye((m)))

for j in range(m):

diff = point - xmat[j]

weights[j, j] = np.exp(diff \* diff.T / (-2.0 \* k\*\*2))

return weights

# Function to return local weight of each training example

def localWeight(point, xmat, ymat, k):

wt = kernel(point, xmat, k)

W = (xmat.T \* (wt \* xmat)).I \* (xmat.T \* wt \* ymat.T)

return W

# Root function that drives the algorithm

def localWeightRegression(xmat, ymat, k):

m, n = np.shape(xmat)

ypred = np.zeros(m)

for i in range(m):

ypred[i] = xmat[i] \* localWeight(xmat[i], xmat, ymat, k)

return ypred

# Load data

data = pd.read\_csv('10-dataset.csv')

# Convert data to suitable data types

colA = np.array(data['total\_bill'])

colB = np.array(data['tip'])

mcola = np.mat(colA)

mcolB = np.mat(colB)

m = np.shape(mcolB)[1]

one = np.ones((1, m), dtype=int)

# Horizontal stacking

X = np.hstack((one.T, mcola.T))

print(X.shape)

# Predicting values using LWLR

ypred = localWeightRegression(X, mcolB, 0.8)

# Plotting the predicted graph

xsort = X.copy()

xsort.sort(axis=0)

plt.scatter(colA, colB, color='blue')

plt.plot(xsort[:, 1], ypred[X[:, 1].argsort(0)], color='yellow', linewidth=5)

plt.xlabel('Total Bill')

plt.ylabel('Tip')

plt.show()

**Output**