**1.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.**

import csv

with open('tennis.csv', 'r') as f:

reader = csv.reader(f)

your\_list = list(reader)

h = [['0', '0', '0', '0', '0', '0']]

for i in your\_list:

print(i)

if i[-1] == "True":

j = 0

for x in i:

if x != "True":

if x != h[0][j] and h[0][j] == '0':

h[0][j] = x

elif x != h[0][j] and h[0][j] != '0':

h[0][j] = '?'

else:

pass

j = j + 1

print("Most specific hypothesis is")

print(h)

**Output**

**'Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same',True 'Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same',True 'Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change',False 'Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change',True**

Maximally Specific set

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

**2.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 numpy as np

import math

from data\_loader import read\_data

class Node:

def init (self, attribute):

self.attribute = attribute

self.children = []

self.answer = ""

def str (self):

return self.attribute

def subtables(data, col, delete):

dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32)

for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]:

count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]):

if data[y, col] == items[x]:

dict[items[x]][pos] = data[y]

pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1)

return items, dict

def entropy(S):

items = np.unique(S)

if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1))

sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2)

return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1))

intrinsic = np.zeros((items.shape[0], 1))

for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0)

entropies[x] = ratio \* entropy(dict[items[x]][:, -1])

intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1])

iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]):

total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1:

node = Node("")

node.answer = np.unique(data[:, -1])[0]

return node

gains = np.zeros((data.shape[1] - 1, 1))

for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col)

split = np.argmax(gains)

node = Node(metadata[split]

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata)

node.children.append((items[x], child))

return node

def empty(size):

s = ""

for x in range(size):

s += " "

return s

def print\_tree(node, level):

if node.answer != "":

print(empty(level), node.answer)

return

print(empty(level), node.attribute)

for value, n in node.children:

print(empty(level + 1), value)

print\_tree(n, level + 2)

metadata, traindata = read\_data("tennis.csv")

data = np.array(traindata)

node = create\_node(data, metadata)

print\_tree(node, 0)

**Data\_loader.py**

import csv

def read\_data(filename):

with open(filename, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter=',')

headers = next(datareader)

metadata = []

traindata = []

for name in headers:

metadata.append(name)

for row in datareader:

traindata.append(row)

return (metadata, traindata)

**Tennis.csv**

outlook,temperature,humidity,wind,

answer sunny,hot,high,weak,no

sunny,hot,high,strong,no

overcast,hot,high,weak,yes

rain,mild,high,weak,yes

rain,cool,normal,weak,yes

rain,cool,normal,strong,no

overcast,cool,normal,strong,yes

sunny,mild,high,weak,no

sunny,cool,normal,weak,yes

rain,mild,normal,weak,yes

sunny,mild,normal,strong,yes

overcast,mild,high,strong,yes

overcast,hot,normal,weak,yes

rain,mild,high,strong,no

**Output**

outlook

overcast

b'yes'

rain

wind

b'strong'

b'no'

b'weak'

b'yes'

sunny

humidity

b'high'

b'no'

b'normal'

b'yes

**3.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=7000 #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)

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

d\_hiddenlayer = EH \* hiddengrad

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

* bout += np.sum(d\_output, axis=0,keepdims=True) \*lr wh += X.T.dot(d\_hiddenlayer) \*lr

#bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

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

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

print("Predicted Output: \n" ,output)

**output**

Input:

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual Output:

[[ 0.92]

[ 0.86]

[ 0.89]]

Predicted Output:

[[ 0.89559591]

[ 0.88142069]

[ 0.8928407 ]]

**4.Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

import csv

import random

import math

def loadCsv(filename):

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

dataset = list(lines)

for i in range(len(dataset)):

#converting strings into numbers for processing dataset[i] = [float(x) for x in dataset[i]]

return dataset

def splitDataset(dataset, splitRatio):

#67% training size

trainSize = int(len(dataset) \* splitRatio);

trainSet = []

copy = list(dataset);

while len(trainSet) < trainSize:

#generate indices for the dataset list randomly to pick ele for training data index = random.randrange(len(copy));

trainSet.append(copy.pop(index))

return [trainSet, copy]

def separateByClass(dataset):

separated = {}

#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to each class

for i in range(len(dataset)):

vector = dataset[i]

if (vector[-1] not in separated):

separated[vector[-1]] = []

separated[vector[-1]].append(vector)

return separated

def mean(numbers):

return sum(numbers)/float(len(numbers))

def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)

return math.sqrt(variance)

def summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)];

del summaries[-1]

return summaries

def summarizeByClass(dataset):

separated = separateByClass(dataset);

summaries = {}

for classValue, instances in separated.items():

#summaries is a dic of tuples(mean,std) for each class value

summaries[classValue] = summarize(instances)

return summaries

def calculateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

def calculateClassProbabilities(summaries, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.items():#class and attribute information as mean and sd

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely

x = inputVector[i] #testvector's first attribute probabilities[classValue] \*= calculateProbability(x, mean, stdev);#use

normal dist

return probabilities

def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():#assigns that class which has he highest prob

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel

def getPredictions(summaries, testSet):

predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i])

predictions.append(result)

return predictions

def getAccuracy(testSet, predictions):

correct = 0

for i in range(len(testSet)):

if testSet[i][-1] == predictions[i]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

filename = '5data.csv'

splitRatio = 0.67

dataset = loadCsv(filename);

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))

# prepare model

summaries = summarizeByClass(trainingSet);

# test model

predictions = getPredictions(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)

print('Accuracy of the classifier is : {0}%'.format(accuracy))

main()

**Output**

confusion matrix is as

follows [[17 0 0]

[0170]

[0011]]

Accuracy metrics

precision recall f1-score support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1.00 | 1.00 | 1.00 | 17 |
| 1 | 1.00 | 1.00 | 1.00 | 17 |
| 2 | 1.00 | 1.00 | 1.00 | 11 |
| avg / total | 1.00 | 1.00 | 1.00 | 45 |

**5.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

msg=pd.read\_csv('naivetext1.csv',names=['message','label']) print('The dimensions of the dataset',msg.shape) msg['labelnum']=msg.label.map({'pos':1,'neg':0}) X=msg.message

y=msg.labelnum

print(X)

print(y)

#splitting the dataset into train and test data

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

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

print(xtest.shape)

print(xtrain.shape)

print(ytest.shape)

print(ytrain.shape)

#output of count vectoriser is a sparse matrix

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

xtrain\_dtm = count\_vect.fit\_transform(xtrain)

xtest\_dtm=count\_vect.transform(xtest)

print(count\_vect.get\_feature\_names())

df=pd.DataFrame(xtrain\_dtm.toarray(),columns=count\_vect.get\_feature\_names()) print(df)#tabular representation

print(xtrain\_dtm) #sparse matrix representation

* Training Naive Bayes (NB) classifier on training data. from sklearn.naive\_bayes import MultinomialNB

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

#printing accuracy metrics

from sklearn import metrics

print('Accuracy metrics')

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

print('Confusion matrix')

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

print('Recall and Precison ')

print(metrics.recall\_score(ytest,predicted))

print(metrics.precision\_score(ytest,predicted))

'''docs\_new = ['I like this place', 'My boss is not my saviour']

X\_new\_counts = count\_vect.transform(docs\_new)

predictednew = clf.predict(X\_new\_counts)

for doc, category in zip(docs\_new, predictednew):

print('%s->%s' % (doc, msg.labelnum[category]))'''

I love this sandwich,pos

This is an amazing place,pos

I feel very good about these beers,pos

This is my best work,pos

What an awesome view,pos

I do not like this restaurant,neg

I am tired of this stuff,neg

I can't deal with this,neg

He is my sworn enemy,neg

My boss is horrible,neg

This is an awesome place,pos

I do not like the taste of this juice,neg

I love to dance,pos

I am sick and tired of this place,neg

What a great holiday,pos

That is a bad locality to stay,neg

We will have good fun tomorrow,pos

I went to my enemy's house today,neg

**OUTPUT**

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'beers', 'best', 'boss', 'can', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'have', 'horrible', 'house', 'is', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant', 'sandwich', 'sick', 'stuff', 'these', 'this', 'tired', 'to', 'today', 'tomorrow', 'very', 'view', 'we', 'went', 'what', 'will', 'with', 'work']

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | about am amazing an and awesome beers best boss can ... today \ | | | |
| 0 | 1 0 | 00001 | 0 0 | 0...0 |
| 1 | 0 0 | 00000 | 1 0 | 0...0 |
| 2 | 0 0 | 1100 0000...0 | | |
| 3 | 0 0 | 00000 | 0 0 | 0...1 |
| 4 | 0 0 | 00000 | 0 0 | 0...0 |
| 5 | 0 1 | 001 00000...0 | | |
| 6 | 0 0 | 00000 | 0 0 | 1...0 |
| 7 | 0 0 | 00000 | 0 0 | 0...0 |
| 8 | 0 1 | 00000 | 0 0 | 0...0 |
| 9 | 0 0 | 01010 | 0 0 | 0...0 |
| 1000 | | 00000000 | | ... 0 |
| 1100 | | 000 00010...0 | | |
| 1200 | | 01010000 | | ... 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | |  | |
|  |  |  |  |  |  |  | |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

**6.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. You can use Java/Python ML library classes/API.**

From pomegranate import\*

Asia=DiscreteDistribution({ „True‟:0.5, „False‟:0.5 })

Tuberculosis=ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.2],

[„True‟, „False‟, 0.8],

[ „False‟, „True‟, 0.01],

[ „False‟, „False‟, 0.98]], [asia])

Smoking = DiscreteDistribution({ „True‟:0.5, „False‟:0.5 })

Lung = ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.75],

[„True‟, „False‟,0.25].

[ „False‟, „True‟, 0.02],

[ „False‟, „False‟, 0.98]], [ smoking])

Bronchitis = ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.92],

[„True‟, „False‟,0.08].

[ „False‟, „True‟,0.03],

[ „False‟, „False‟, 0.98]], [ smoking])

Tuberculosis\_or\_cancer = ConditionalProbabilityTable(

[[ „True‟, „True‟, „True‟, 1.0],

[„True‟, „True‟, „False‟, 0.0],

[„True‟, „False‟, „True‟, 1.0],

[„True‟, „False‟, „False‟, 0.0],

[„False‟, „True‟, „True‟, 1.0],

[„False‟, „True‟, „False‟, 0.0],

[„False‟, „False‟ „True‟, 1.0],

[„False‟, „False‟, „False‟, 0.0]], [tuberculosis, lung])

Xray = ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.885],

[„True‟, „False‟, 0.115],

[ „False‟, „True‟, 0.04],

[ „False‟, „False‟, 0.96]], [tuberculosis\_or\_cancer])

dyspnea = ConditionalProbabilityTable(

[[ „True‟, „True‟, „True‟, 0.96],

[„True‟, „True‟, „False‟, 0.04],

[„True‟, „False‟, „True‟, 0.89],

[„True‟, „False‟, „False‟, 0.11],

[„False‟, „True‟, „True‟, 0.96],

[„False‟, „True‟, „False‟, 0.04],

[„False‟, „False‟ „True‟, 0.89],

[„False‟, „False‟, „False‟, 0.11 ]], [tuberculosis\_or\_cancer, bronchitis])

s0 = State(asia, name=”asia”)

s1 = State(tuberculosis, name=” tuberculosis”)

s2 = State(smoking, name=” smoker”)

network = BayesianNetwork(“asia”)

network.add\_nodes(s0,s1,s2)

network.add\_edge(s0,s1)

network.add\_edge(s1.s2)

network.bake()

print(network.predict\_probal({„tuberculosis‟: „True‟}))

**7.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 numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets.samples\_generator import make\_blobs X, y\_true = make\_blobs(n\_samples=100, centers = 4,Cluster\_std=0.60,random\_state=0) X = X[:, ::-1]

**#flip axes for better plotting**

from sklearn.mixture import GaussianMixture gmm = GaussianMixture (n\_components = 4).fit(X) lables = gmm.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap=‟viridis‟); probs = gmm.predict\_proba(X) print(probs[:5].round(3))

size = 50 \* probs.max(1) \*\* 2 # square emphasizes differences plt.scatter(X[:, 0], X[:, 1], c=labels, cmap=‟viridis‟, s=size);

from matplotlib.patches import Ellipse

def draw\_ellipse(position, covariance, ax=None, \*\*kwargs); “””Draw an ellipse with a given position and covariance”””

Ax = ax or plt.gca()

* **Convert covariance to principal axes** if covariance.shape ==(2,2):

U, s, Vt = np.linalg.svd(covariance)

Angle = np.degrees(np.arctan2(U[1, 0], U[0,0]))

Width, height = 2 \* np.sqrt(s)

else:

angle = 0

width, height = 2 \* np.sqrt(covariance)

**#Draw the Ellipse**

for nsig in range(1,4):

ax.add\_patch(Ellipse(position, nsig \* width, nsig \*height,

angle, \*\*kwargs))

def plot\_gmm(gmm, X, label=True, ax=None):

ax = ax or plt.gca()

labels = gmm.fit(X).predict(X)

if label:

ax.scatter(X[:, 0], x[:, 1], c=labels, s=40, cmap=‟viridis‟, zorder=2)

else:

ax.scatter(X[:, 0], x[:, 1], s=40, zorder=2)

ax.axis(„equal‟)

w\_factor = 0.2 / gmm.weights\_.max()

for pos, covar, w in zip(gmm.means\_, gmm.covariances\_, gmm.weights\_):

draw\_ellipse(pos, covar, alpha=w \* w\_factor)

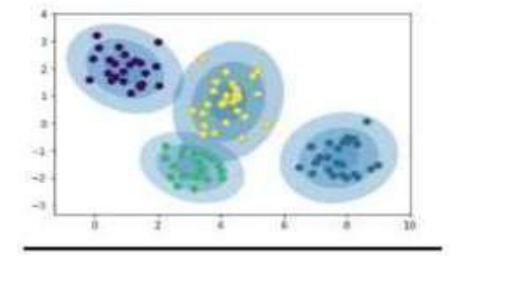
gmm = GaussianMixture(n\_components=4, random\_state=42)

plot\_gmm(gmm, X)

gmm = GaussianMixture(n\_components=4, covariance\_type=‟full‟, random\_state=42)

plot\_gmm(gmm, X)

**Output**

****

**[[1 ,0, 0, 0]**

**[0 ,0, 1, 0]**

**[1 ,0, 0, 0]**

**[1 ,0, 0, 0]**

**[1 ,0, 0, 0]]**

**K-means**

from sklearn.cluster import KMeans

#from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

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

df1=pd.DataFrame(data)

print(df1)

f1 = df1['Distance\_Feature'].values

f2 = df1['Speeding\_Feature'].values

X=np.matrix(list(zip(f1,f2)))

plt.plot()

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.title('Dataset')

plt.ylabel('speeding\_feature')

plt.xlabel('Distance\_Feature')

plt.scatter(f1,f2)

plt.show()

* create new plot and data plt.plot()

colors = ['b', 'g', 'r'] markers = ['o', 'v', 's']

* KMeans algorithm

#K=3

kmeans\_model = KMeans(n\_clusters=3).fit(X)

plt.plot()

for i, l in enumerate(kmeans\_model.labels\_):

plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')

plt.xlim([0, 100])

plt.ylim([0, 50])

plt.show()

**Driver\_ID,Distance\_Feature,Speeding\_Feature** 3423311935,71.24,28

3423313212,52.53,25

3423313724,64.54,27

3423311373,55.69,22

3423310999,54.58,25

3423313857,41.91,10

3423312432,58.64,20

3423311434,52.02,8

3423311328,31.25,34

3423312488,44.31,19

3423311254,49.35,40

3423312943,58.07,45

3423312536,44.22,22

3423311542,55.73,19

3423312176,46.63,43

3423314176,52.97,32

3423314202,46.25,35

3423311346,51.55,27

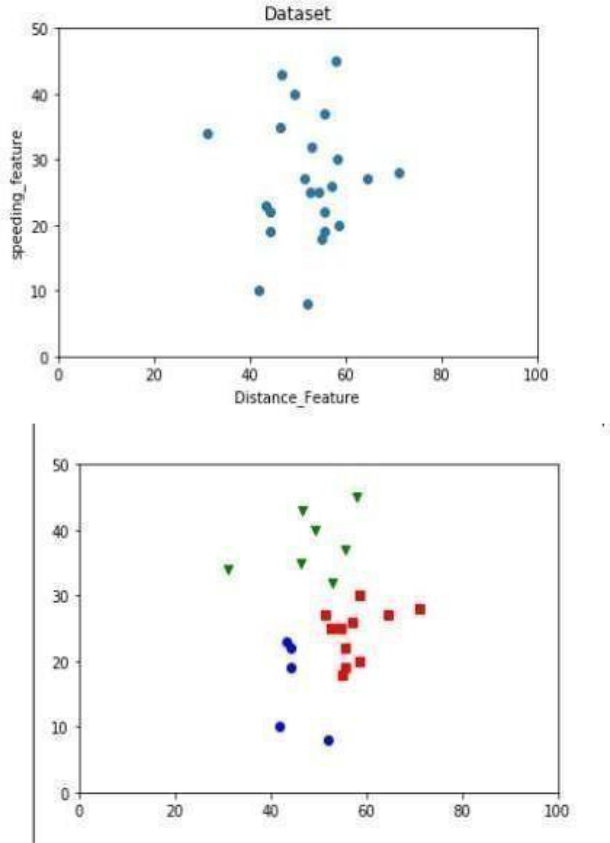
3423310666,57.05,26

3423313527,58.45,30

3423312182,43.42,23

3423313590,55.68,37

3423312268,55.15,18



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

import csv

import random

import math

import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]):

with open(filename, 'rb') as csvfile:

lines = csv.reader(csvfile)

dataset = list(lines)

for x in range(len(dataset)-1):

for y in range(4):

dataset[x][y] = float(dataset[x][y])

if random.random() < split:

trainingSet.append(dataset[x])

else:

testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length):

distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2)

return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):

distances = []

length = len(testInstance)-1

for x in range(len(trainingSet)):

dist = euclideanDistance(testInstance, trainingSet[x], length)

distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1))

neighbors = []

for x in range(k):

neighbors.append(distances[x][0])

return neighbors

def getResponse(neighbors):

classVotes = {}

for x in range(len(neighbors)):

response = neighbors[x][-1]

if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes =

sorted(classVotes.iteritems(),

reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet,

predictions): correct = 0

for x in

range(len(testSet)):

key=operator.itemgetter(1

),

if testSet[x][-1] == predictions[x]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

* prepare data trainingSet= [] testSet=[] split = 0.67

loadDataset('knndat.data', split, trainingSet, testSet) print('Train set: ' + repr(len(trainingSet))) print('Test set: ' + repr(len(testSet)))

* generate

predictions

predictions=[]

k=3

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x],

1. result = getResponse(neighbors) predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-

1])) accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) +

'%') main()

**OUTPUT**

**Confusion matrix is as follows**

**[[11 0 0]**

**[091]**

**[0 1 8]]**

**Accuracy metrics**

**0 1.00 1.00 1.00 11**

**1 0.90 0.90 0.90 10**

**2 0.89 0.89 0,89 9**

**Avg/Total 0.93 0.93 0.93 30**

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

from numpy import \*

import operator

from os import listdir

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

import numpy.linalg as np

from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):

m,n = np1.shape(xmat)

weights = np1.mat(np1.eye((m)))

for j in range(m):

diff = point - X[j]

weights[j,j] = np1.exp(diff\*diff.T/(-2.0\*k\*\*2))

return weights

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

wei = kernel(point,xmat,k)

W=(X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T))

return W

def localWeightRegression(xmat,ymat,k):

m,n = np1.shape(xmat)

ypred = np1.zeros(m)

for i in range(m):

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

return ypred

# load data points

data = pd.read\_csv('data10.csv')

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

tip = np1.array(data.tip)

#preparing and add 1 in bill

mbill = np1.mat(bill)

mtip = np1.mat(tip)

m= np1.shape(mbill)[1]

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

X= np1.hstack((one.T,mbill.T))

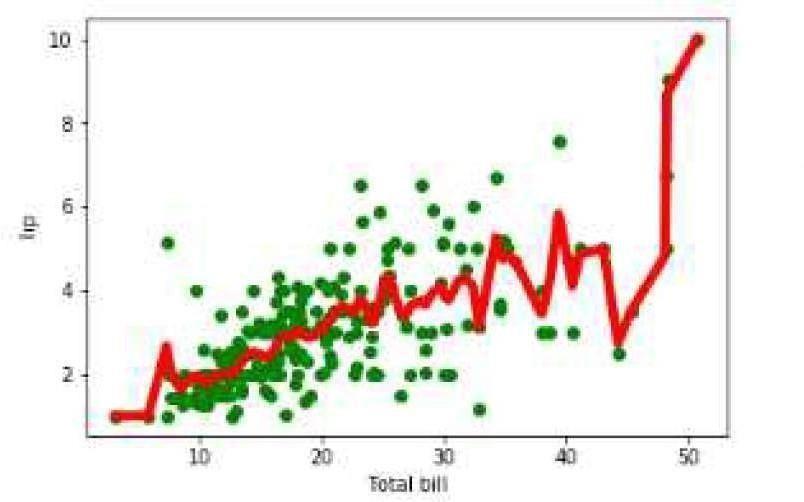
#set k here

ypred = localWeightRegression(X,mtip,2)

SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]

**Output**

****