1.Implement and demonstratethe **FIND-Salgorithm** for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

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

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

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

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

6.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, pre cision, and recall for your data set.

7.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.

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.

9. 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.

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

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

def loadCsv(filename):

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

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = dataset[i]

return dataset

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

print(attributes)

num\_attributes = len(attributes)

filename = "Weather.csv"

dataset = loadCsv(filename)

print(dataset)

hypothesis=['0'] \* num\_attributes

print("Intial Hypothesis")

print(hypothesis)

print("The Hypothesis are")

for i in range(len(dataset)):

target = dataset[i][-1]

if(target == 'Yes'):

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**

runfile('F:/Machine Learning/ML Lab/01-finds.py', wdir='F:/Machine Learning/ML Lab')

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

[['Sunny ', 'Warm', 'Normal', 'Strong ', 'Warm', 'Same', 'Yes'], ['Sunny ', 'Warm', 'High', 'Strong ', 'Warm', 'Same', 'Yes'], ['Rainy', 'Cold', 'High', 'Strong ', 'Warm', 'Change', 'No'], ['Sunny ', 'Warm', 'High', 'Strong ', 'Cool', 'Change', 'Yes']]

Intial Hypothesis

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

The Hypothesis are

(1, '=', ['Sunny ', 'Warm', 'Normal', 'Strong ', 'Warm', 'Same'])

(2, '=', ['Sunny ', 'Warm', '?', 'Strong ', 'Warm', 'Same'])

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

(4, '=', ['Sunny ', 'Warm', '?', 'Strong ', '?', '?'])

Final Hypothesis

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

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

import csv

def get\_domains(examples):

d = [set() for i in examples[0]]

for x in examples:

for i, xi in enumerate(x):

d[i].add(xi)

return [list(sorted(x)) for x in d]

def more\_general(h1, h2):

more\_general\_parts = []

for x, y in zip(h1, h2):

mg = x == "?" or (x != "0" and (x == y or y == "0"))

more\_general\_parts.append(mg)

return all(more\_general\_parts)

def fulfills(example, hypothesis):

# the implementation is the same as for hypotheses:

return more\_general(hypothesis, example)

def min\_generalizations(h, x):

h\_new = list(h)

for i in range(len(h)):

if not fulfills(x[i:i+1], h[i:i+1]):

h\_new[i] = '?' if h[i] != '0' else x[i]

return [tuple(h\_new)]

def min\_specializations(h, domains, x):

results = []

for i in range(len(h)):

if h[i] == "?":

for val in domains[i]:

if x[i] != val:

h\_new = h[:i] + (val,) + h[i+1:]

results.append(h\_new)

elif h[i] != "0":

h\_new = h[:i] + ('0',) + h[i+1:]

results.append(h\_new)

return results

def generalize\_S(x, G, S):

S\_prev = list(S)

for s in S\_prev:

if s not in S:

continue

if not fulfills(x, s):

S.remove(s)

Splus = min\_generalizations(s, x)

## keep only generalizations that have a counterpart in G

S.update([h for h in Splus if any([more\_general(g,h)

for g in G])])

## remove hypotheses less specific than any other in S

S.difference\_update([h for h in S if

any([more\_general(h, h1)

for h1 in S if h != h1])])

return S

def specialize\_G(x, domains, G, S):

G\_prev = list(G)

for g in G\_prev:

if g not in G:

continue

if fulfills(x, g):

G.remove(g)

Gminus = min\_specializations(g, domains, x)

## keep only specializations that have a conuterpart in S

G.update([h for h in Gminus if any([more\_general(h, s)

for s in S])])

## remove hypotheses less general than any other in G

G.difference\_update([h for h in G if

any([more\_general(g1, h)

for g1 in G if h != g1])])

return G

def candidate\_elimination(examples):

domains = get\_domains(examples)[:-1]

n = len(domains)

G = set([("?",)\*n])

S = set([("0",)\*n])

print("Maximally specific hypotheses - S ")

print("Maximally general hypotheses - G ")

i=0

print("\nS[0]:",str(S),"\nG[0]:",str(G))

for xcx in examples:

i=i+1

x, cx = xcx[:-1], xcx[-1] # Splitting data into attributes and decisions

if cx=='Y': # x is positive example

G = {g for g in G if fulfills(x, g)}

S = generalize\_S(x, G, S)

else: # x is negative example

S = {s for s in S if not fulfills(x, s)}

G = specialize\_G(x, domains, G, S)

print("\nS[{0}]:".format(i),S)

print("G[{0}]:".format(i),G)

return

with open('tennis1.csv') as csvFile:

examples = [tuple(line) for line in csv.reader(csvFile)]

candidate\_elimination(examples)

Maximally specific hypotheses - S

Maximally general hypotheses - G

S[0]: {('0', '0', '0', '0', '0', '0')}

G[0]: {('?', '?', '?', '?', '?', '?')}

S[1]: {(' Sunny', 'Warm', 'Normal', 'Strong', ' Warm', 'Same')}

G[1]: {('?', '?', '?', '?', '?', '?')}

S[2]: {(' Sunny', 'Warm', '?', 'Strong', '?', 'Same')}

G[2]: {('?', '?', '?', '?', '?', '?')}

S[3]: {(' Sunny', 'Warm', '?', 'Strong', '?', 'Same')}

G[3]: {(' Sunny', '?', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same'), ('?', 'Warm', '?', '?', '?', '?')}

S[4]: {(' Sunny', 'Warm', '?', 'Strong', '?', '?')}

G[4]: {(' Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?')}

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

import math

import csv

def load\_csv(filename):

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

dataset = list(lines)

headers = dataset.pop(0)

return dataset, headers

class Node:

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

self.attribute = attribute

self.children = []

self.answer = "" # NULL indicates children exists. # Not Null indicates this is a Leaf Node

def subtables(data, col, delete):

dic = {}

coldata = [ row[col] for row in data]

attr = list(set(coldata)) # All values of attribute retrived

for k in attr:

dic[k] = []

for y in range(len(data)):

key = data[y][col]

if delete:

del data[y][col]

dic[key].append(data[y])

return attr, dic

def entropy(S):

attr = list(set(S))

if len(attr) == 1: #if all are +ve/-ve then entropy = 0

return 0

counts = [0,0] # Only two values possible 'yes' or 'no'

for i in range(2):

counts[i] = sum( [1 for x in S if attr[i] == x] ) / (len(S) \* 1.0)

sums = 0

for cnt in counts:

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

return sums

def compute\_gain(data, col):

attValues, dic = subtables(data, col, delete=False)

total\_entropy = entropy([row[-1] for row in data])

for x in range(len(attValues)):

ratio = len(dic[attValues[x]]) / ( len(data) \* 1.0)

entro = entropy([row[-1] for row in dic[attValues[x]]])

total\_entropy -= ratio\*entro

return total\_entropy

def build\_tree(data, features):

lastcol = [row[-1] for row in data]

if (len(set(lastcol))) == 1: # If all samples have same labels return that label

node=Node("")

node.answer = lastcol[0]

return node

n = len(data[0])-1

gains = [compute\_gain(data, col) for col in range(n) ]

split = gains.index(max(gains)) # Find max gains and returns index

node = Node(features[split]) # 'node' stores attribute selected

#del (features[split])

fea = features[:split]+features[split+1:]

attr, dic = subtables(data, split, delete=True) # Data will be spilt in subtables

for x in range(len(attr)):

child = build\_tree(dic[attr[x]], fea)

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

return node

def print\_tree(node, level):

if node.answer != "":

print(" "\*level, node.answer) # Displays leaf node yes/no

return

print(" "\*level, node.attribute) # Displays attribute Name

for value, n in node.children:

print(" "\*(level+1), value)

print\_tree(n, level + 2)

def classify(node,x\_test,features):

if node.answer != "":

print(node.answer)

return

pos = features.index(node.attribute)

for value, n in node.children:

if x\_test[pos]==value:

classify(n,x\_test,features)

''' Main program '''

dataset, features = load\_csv("data3.csv") # Read Tennis data

node = build\_tree(dataset, features) # Build decision tree

print("The decision tree for the dataset using ID3 algorithm is ")

print\_tree(node, 0)

testdata, features = load\_csv("data3.csv")

for xtest in testdata:

print("The test instance : ",xtest)

print("The predicted label : ",end="")

classify(node,xtest,features)

**OUTPUT**

The decision tree for the dataset using ID3 algorithm is

('', 'Outlook')

('\t', 'overcast')

(' ', 'yes')

('\t', 'sunny')

(' ', 'Humidity')

('\t\t\t', 'high')

(' ', 'no')

('\t\t\t', 'normal')

(' ', 'yes')

('\t', 'rain')

(' ', 'Wind')

('\t\t\t', 'strong')

(' ', 'no')

('\t\t\t', 'weak')

(' ', 'yes')

('The test instance : ', ['sunny', 'hot', 'high', 'weak', 'no'])

no

('The test instance : ', ['sunny', 'hot', 'high', 'strong', 'no'])

no

('The test instance : ', ['overcast', 'hot', 'high', 'weak', 'yes'])

yes

('The test instance : ', ['rain', 'mild', 'high', 'weak', 'yes'])

yes

('The test instance : ', ['rain', 'cool', 'normal', 'weak', 'yes'])

yes

('The test instance : ', ['rain', 'cool', 'normal', 'strong', 'no'])

no

('The test instance : ', ['overcast', 'cool', 'normal', 'strong', 'yes'])

yes

('The test instance : ', ['sunny', 'mild', 'high', 'weak', 'no'])

no

('The test instance : ', ['sunny', 'cool', 'normal', 'weak', 'yes'])

yes

('The test instance : ', ['rain', 'mild', 'normal', 'weak', 'yes'])

yes

('The test instance : ', ['sunny', 'mild', 'normal', 'strong', 'yes'])

yes

('The test instance : ', ['overcast', 'mild', 'high', 'strong', 'yes'])

yes

('The test instance : ', ['overcast', 'hot', 'normal', 'weak', 'yes'])

yes

('The test instance : ', ['rain', 'mild', 'high', 'strong', 'no'])

no

4. Build an Artificial Neural Network by implementing the **Back propagation algorithm** and test the same using appropriate data sets from math import exp from random import seed from random import random

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6],[5,4]), dtype=float)

Y = np.array(([92], [86], [89],[90]), dtype=float)

y = Y/100

m,n=np.shape(Y)

print(m)

print(n)

print(X)

print(y)

#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=100000 #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))

print(wh)

print(bh)

print(wout)

print(bout)

for i in range(epoch):

#Forward Propogation # Dot product + bias

h\_ip=np.dot(X,wh) + bh

h\_act = sigmoid(h\_ip) # Activation function

o\_ip=np.dot(h\_act,wout) + bout

output = sigmoid(o\_ip)

#Backpropagation

# Error at Output layer # Error at o/p

Eo = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = Eo\* outgrad # Errj=Oj(1-Oj)(Tj-Oj)

# Error at Hidden later # .T means transpose

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

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

d\_hidden = Eh \* hiddengrad # Dotproduct of nextlayererror and currentlayerop

wout += h\_act.T.dot(d\_output) \*lr

wh += X.T.dot(d\_hidden) \*lr

print(wh)

print(wout)

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

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

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

**OUTPUT**

Normalized Input:

[[2. 9.]

[1. 5.]

[3. 6.]

[5. 4.]]

Actual Output:

[[0.92]

[0.86]

[0.89]

[0.9 ]]

('Predicted Output: \n', array([[0.90395176],

[0.87023216],

[0.89541639],

[0.89816954]]))

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

# Naive Bayes implementation in Python

import csv

import random

import math

#1.Load Data

def loadCsv(filename):

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

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [float(x) for x in dataset[i]]

return dataset

#Split the data into Training and Testing randomly

def splitDataset(dataset, splitRatio):

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

trainSet = []

copy = list(dataset)

while len(trainSet) < trainSize:

index = random.randrange(len(copy))

trainSet.append(copy.pop(index))

return [trainSet, copy]

#Seperatedata by Class

def separateByClass(dataset):

separated = {}

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

#Calculate Mean

def mean(numbers):

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

#Calculate Standard Deviation

def stdev(numbers):

avg = mean(numbers)

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

return math.sqrt(variance)

#Summarize the data

def summarize(dataset):

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

del summaries[-1]

return summaries

#Summarize Attributes by Class

def summarizeByClass(dataset):

separated = separateByClass(dataset)

print(len(separated))

summaries = {}

for classValue, instances in separated.items():

summaries[classValue] = summarize(instances)

print(summaries)

return summaries

#Calculate Gaussian Probability Density Function

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

#Calculate Class Probabilities

def calculateClassProbabilities(summaries, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.items():

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i]

x = inputVector[i]

probabilities[classValue] \*= calculateProbability(x, mean, stdev)

return probabilities

#Make a Prediction

def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel

#return a list of predictions for each test instance.

def getPredictions(summaries, testSet):

predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i])

predictions.append(result)

return predictions

#calculate accuracy ratio.

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

filename = 'DBetes.csv'

splitRatio = 0.70

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: {0}%'.format(accuracy))

**OUTPUT**

runfile('F:/Machine Learning/ML Lab/05-NaiveBayesianclassifier.py', wdir='F:/Machine Learning/ML Lab')

Split 250 rows into train=175 and test=75 rows

2

{0.0: [(3.5714285714285716, 2.9184541314494767), (110.01785714285714, 30.97193793599199), (68.5, 18.1976139412054), (18.633928571428573, 15.631451049239223), (61.125, 111.53160784974611)], 1.0: [(4.9523809523809526, 3.607467932607666), (145.23809523809524, 31.34187875758726), (72.92063492063492, 19.56486448128427), (23.61904761904762, 15.990348471488398), (114.7936507936508, 161.2343674573134)]}

Accuracy: 70.6666666667%

6. 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, pre cision, and recall for your data set.

#1 Loading the data set

from sklearn.datasets import fetch\_20newsgroups

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True)

print("lenth of the twenty\_train--------->", len(twenty\_train))

#print(twenty\_train.target\_names) #prints all the categories

print("\*\*\*First Line of the First Data File\*\*\*")

#print("\n".join(twenty\_train.data[0].split("\n")[:5]))#prints first line of the first data file

#2 Extracting features from text files

from sklearn.feature\_extraction.text import CountVectorizer

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform(twenty\_train.data)

print('dim=',X\_train\_counts.shape)

#3 TF-IDF

from sklearn.feature\_extraction.text import TfidfTransformer

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts)

print(X\_train\_tfidf.shape)

# Machine Learning

#4 Training Naive Bayes (NB) classifier on training data.

from sklearn.naive\_bayes import MultinomialNB

clf = MultinomialNB().fit(X\_train\_tfidf, twenty\_train.target)

# Building a pipeline: We can write less code and do all of the above, by building a pipeline as follows:

# The names ‘vect’ , ‘tfidf’ and ‘clf’ are arbitrary but will be used later.

# We will be using the 'text\_clf' going forward.

from sklearn.pipeline import Pipeline

text\_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])

text\_clf = text\_clf.fit(twenty\_train.data, twenty\_train.target)

# Performance of NB Classifier

import numpy as np

twenty\_test = fetch\_20newsgroups(subset='test', shuffle=True)

predicted = text\_clf.predict(twenty\_test.data)

accuracy=np.mean(predicted == twenty\_test.target)

print("Predicted Accuracy = ",accuracy)

#To Calculate Accuracy,Precision,Recall

from sklearn import metrics

print("Accuracy= ",metrics.accuracy\_score(twenty\_test.target,predicted))

print("Precision=",metrics.precision\_score(twenty\_test.target,predicted,average=None))

print("Recall=",metrics.recall\_score(twenty\_test.target,predicted,average=None))

print(metrics.classification\_report(twenty\_test.target, predicted,target\_names=twenty\_test.target\_names))

**OUTPUT**

runfile('F:/Machine Learning/ML Lab/06-NaiveBayesianclassifier.py', wdir='F:/Machine Learning/ML Lab')

('lenth of the twenty\_train--------->', 6)

\*\*\*First Line of the First Data File\*\*\*

('dim=', (11314, 130107))

(11314, 130107)

('Predicted Accuracy = ', 0.7738980350504514)

('Accuracy= ', 0.7738980350504514)

('Precision=', array([0.80193237, 0.81028939, 0.81904762, 0.67180617, 0.85632184,

0.88955224, 0.93127148, 0.84651163, 0.93686869, 0.92248062,

0.89170507, 0.59379845, 0.83629893, 0.92113565, 0.84172662,

0.43896976, 0.64339623, 0.92972973, 0.95555556, 0.97222222]))

('Recall=', array([0.52037618, 0.64781491, 0.65482234, 0.77806122, 0.77402597,

0.75443038, 0.69487179, 0.91919192, 0.9321608 , 0.89924433,

0.96992481, 0.96717172, 0.59796438, 0.73737374, 0.89086294,

0.98492462, 0.93681319, 0.91489362, 0.41612903, 0.13944223]))

precision recall f1-score support

alt.atheism 0.80 0.52 0.63 319

comp.graphics 0.81 0.65 0.72 389

comp.os.ms-windows.misc 0.82 0.65 0.73 394

comp.sys.ibm.pc.hardware 0.67 0.78 0.72 392

comp.sys.mac.hardware 0.86 0.77 0.81 385

comp.windows.x 0.89 0.75 0.82 395

misc.forsale 0.93 0.69 0.80 390

rec.autos 0.85 0.92 0.88 396

rec.motorcycles 0.94 0.93 0.93 398

rec.sport.baseball 0.92 0.90 0.91 397

rec.sport.hockey 0.89 0.97 0.93 399

sci.crypt 0.59 0.97 0.74 396

sci.electronics 0.84 0.60 0.70 393

sci.med 0.92 0.74 0.82 396

sci.space 0.84 0.89 0.87 394

soc.religion.christian 0.44 0.98 0.61 398

talk.politics.guns 0.64 0.94 0.76 364

talk.politics.mideast 0.93 0.91 0.92 376

talk.politics.misc 0.96 0.42 0.58 310

talk.religion.misc 0.97 0.14 0.24 251

avg / total 0.82 0.77 0.77 7532

7.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.

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

#Read the attributes

lines = list(csv.reader(open('data7\_names.csv', 'r')));

attributes = lines[0]

#Read Cleveland Heart dicease data

print(lines)

print(lines[0])

heartDisease = pd.read\_csv('data7\_heart.csv', names = attributes)

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

print('Few examples from the dataset are given below')

print(heartDisease.head())

print('\nAttributes and datatypes')

print(heartDisease.dtypes)

# Model Baysian Network

model = BayesianModel([('age', 'trestbps'), ('age', 'fbs'), ('sex', 'trestbps'),

('exang', 'trestbps'),('trestbps','heartdisease'),('fbs','heartdisease'),

('heartdisease','restecg'),('heartdisease','thal'),('heartdisease','chol')])

import networkx as nx

import pylab as plt

nx.draw(model, with\_labels=True)

plt.show()

model.local\_independencies('trestbps')

model.get\_independencies()

# Learning CPDs using Maximum Likelihood Estimators

print('\nLearning CPDs using Maximum Likelihood Estimators...');

print(model.fit(heartDisease))

print(model.get\_cpds('age'))

print(model.get\_cpds('thal'))

print(model.get\_cpds('chol'))

print(model.get\_cpds('sex'))

# Inferencing with Bayesian Network

print('\nInferencing with Bayesian Network:')

HeartDisease\_infer = VariableElimination(model)

# Computing the probability

print('\n1.Probability of HeartDisease given Age=40')

q = HeartDisease\_infer.query(variables=['heartdisease'],evidence={'age':40,'chol':15})

print(q['heartdisease'])

print('\n2. Probability of HeartDisease given chol (Cholestoral) =100')

q = HeartDisease\_infer.query(variables=['heartdisease'],evidence={'chol':0})

print(q['heartdisease'])

**OUTPUT**

1.Probability of HeartDisease given Age=40

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 0.4887 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.0000 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.1984 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.3128 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0000 │

╘════════════════╧═════════════════════╛

2. Probability of HeartDisease given chol (Cholestoral) =100

╒════════════════╤═════════════════════╕

│ heartdisease │ phi(heartdisease) │

╞════════════════╪═════════════════════╡

│ heartdisease\_0 │ 1.0000 │

├────────────────┼─────────────────────┤

│ heartdisease\_1 │ 0.0000 │

├────────────────┼─────────────────────┤

│ heartdisease\_2 │ 0.0000 │

├────────────────┼─────────────────────┤

│ heartdisease\_3 │ 0.0000 │

├────────────────┼─────────────────────┤

│ heartdisease\_4 │ 0.0000 │

╘════════════════╧═════════════════════╛

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 csv

#import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

#1.Load Data

def loadCsv(filename):

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

dataset = list(lines)

for i in range(len(dataset)):

dataset[i] = [float(x) for x in dataset[i]]

return dataset

filename = "sample\_stocks.csv"

X = loadCsv(filename)

print(X)

kmeans = KMeans(n\_clusters=3)

kmeans.fit(X)

centroids = kmeans.cluster\_centers\_

labels = kmeans.labels\_

print(centroids)

print(labels)

colors = ["g.","r.","y."]

for i in range(len(X)):

#print("coordinate:",X[i],"label:",labels[i])

plt.plot(X[i][0],X[i][1],colors[labels[i]],markersize = 20)

plt.scatter(centroids[:,0],centroids[:,1] ,marker = 'x', s=100,linewidth = 2,zorder=10)

plt.show()

**OUTPUT**

[[-19.0, 0.0], [-13.0, 0.0], [-14.0, 0.0], [-9.0, 0.0], [-19.0, 0.0], [-10.0, 0.0], [-20.0, 0.0], [-11.0, 0.0], [-12.0, 0.0], [-10.0, 0.0], [-13.0, 0.0], [-14.0, 0.0], [-14.0, 0.0], [-12.0, 0.0], [-19.0, 0.0], [-10.0, 0.0], [-11.0, 0.0], [-11.0, 0.0], [-13.0, 0.0], [-11.0, 0.0], [-9.0, 0.0], [-16.0, 0.0], [-9.0, 0.0], [-18.0, 0.0], [-19.0, 0.0], [-19.0, 0.1], [-13.0, 0.1], [-12.0, 0.1], [-8.0, 0.1], [-17.0, 0.1], [-13.0, 0.1], [-9.0, 0.1], [-17.0, 0.1], [-8.0, 0.1], [-20.0, 0.1], [-17.0, 0.1], [-19.0, 0.1], [-11.0, 0.1], [-8.0, 0.2], [-13.0, 0.2], [-17.0, 0.2], [-8.0, 0.2], [-8.0, 0.2], [-20.0, 0.2], [-13.0, 0.2], [-18.0, 0.2], [-16.0, 0.2], [-11.0, 0.2], [-11.0, 0.2], [-10.0, 0.2], [-8.0, 0.2], [-14.0, 0.2], [-20.0, 0.2], [-11.0, 0.2], [-10.0, 0.2], [-20.0, 0.2], [-10.0, 0.3], [-9.0, 0.3], [-10.0, 0.3], [-13.0, 0.3], [-17.0, 0.3], [-10.0, 0.3], [-15.0, 0.3], [-10.0, 0.3], [-13.0, 0.3], [-8.0, 0.3], [-11.0, 0.3], [-9.0, 0.3], [-14.0, 0.3], [-17.0, 0.3], [-16.0, 0.3], [-18.0, 0.3], [-10.0, 0.3], [-12.0, 0.3], [-11.0, 0.3], [-11.0, 0.4], [-17.0, 0.4], [-10.0, 0.4], [-11.0, 0.4], [-8.0, 0.4], [-11.0, 0.4], [-11.0, 0.4], [-17.0, 0.4], [-15.0, 0.4], [-20.0, 0.4], [-16.0, 0.4], [-13.0, 0.4], [-13.0, 0.4], [-17.0, 0.4], [-11.0, 0.4], [-11.0, 0.4], [-19.0, 0.4], [-20.0, 0.4], [-13.0, 0.4], [-17.0, 0.4], [-10.0, 0.4], [-13.0, 0.4], [-13.0, 0.4], [-13.0, 0.5], [-10.0, 0.5], [-16.0, 0.5], [-19.0, 0.5], [-15.0, 0.5], [-13.0, 0.5], [-18.0, 0.5], [-20.0, 0.5], [-9.0, 0.5], [-10.0, 0.5], [-10.0, 0.5], [-19.0, 0.5], [-16.0, 0.5], [-15.0, 0.5], [-12.0, 0.5], [-9.0, 0.5], [-13.0, 0.5], [-20.0, 0.5], [-9.0, 0.6], [-19.0, 0.6], [-18.0, 0.6], [-18.0, 0.6], [-17.0, 0.6], [-16.0, 0.6], [-11.0, 0.6], [-16.0, 0.6], [-17.0, 0.6], [-10.0, 0.6], [-16.0, 0.6], [-19.0, 0.6], [-8.0, 0.6], [-14.0, 0.6], [-18.0, 0.6], [-16.0, 0.6], [-14.0, 0.6], [-16.0, 0.6], [-20.0, 0.6], [-15.0, 0.7], [-20.0, 0.7], [-12.0, 0.7], [-19.0, 0.7], [-13.0, 0.7], [-15.0, 0.7], [-14.0, 0.7], [-11.0, 0.7], [-14.0, 0.7], [-19.0, 0.7], [-14.0, 0.7], [-20.0, 0.7], [-14.0, 0.7], [-16.0, 0.7], [-19.0, 0.7], [-13.0, 0.7], [-19.0, 0.7], [-13.0, 0.7], [-8.0, 0.7], [-20.0, 0.7], [-20.0, 0.7], [-14.0, 0.7], [-15.0, 0.7], [-19.0, 0.7], [-19.0, 0.7], [-19.0, 0.8], [-9.0, 0.8], [-17.0, 0.8], [-15.0, 0.8], [-13.0, 0.8], [-15.0, 0.8], [-12.0, 0.8], [-20.0, 0.8], [-13.0, 0.8], [-13.0, 0.8], [-17.0, 0.8], [-18.0, 0.8], [-19.0, 0.8], [-12.0, 0.8], [-15.0, 0.8], [-10.0, 0.8], [-12.0, 0.8], [-19.0, 0.8], [-20.0, 0.8], [-10.0, 0.9], [-16.0, 0.9], [-11.0, 0.9], [-15.0, 0.9], [-19.0, 0.9], [-10.0, 0.9], [-11.0, 0.9], [-13.0, 0.9], [-12.0, 0.9], [-10.0, 0.9], [-12.0, 0.9], [-10.0, 0.9], [-15.0, 0.9], [-12.0, 0.9], [-15.0, 0.9], [-16.0, 0.9], [-13.0, 0.9], [-15.0, 0.9], [-15.0, 0.9], [-12.0, 0.9], [14.0, 2.0], [10.0, 2.0], [7.0, 2.0], [10.0, 2.0], [14.0, 2.0], [14.0, 2.0], [12.0, 2.0], [7.0, 2.0], [14.0, 2.0], [6.0, 2.0], [10.0, 2.0], [12.0, 2.0], [9.0, 2.0], [8.0, 2.0], [7.0, 2.1], [11.0, 2.1], [8.0, 2.1], [8.0, 2.1], [11.0, 2.1], [7.0, 2.1], [6.0, 2.1], [11.0, 2.1], [6.0, 2.1], [10.0, 2.1], [16.0, 2.1], [11.0, 2.1], [10.0, 2.1], [15.0, 2.1], [16.0, 2.1], [10.0, 2.1], [13.0, 2.1], [7.0, 2.1], [15.0, 2.1], [7.0, 2.1], [14.0, 2.1], [14.0, 2.1], [7.0, 2.1], [6.0, 2.1], [7.0, 2.2], [5.0, 2.2], [9.0, 2.2], [5.0, 2.2], [10.0, 2.2], [14.0, 2.2], [10.0, 2.2], [10.0, 2.2], [5.0, 2.2], [16.0, 2.2], [16.0, 2.2], [13.0, 2.2], [16.0, 2.2], [7.0, 2.2], [14.0, 2.2], [16.0, 2.2], [5.0, 2.2], [12.0, 2.2], [16.0, 2.2], [14.0, 2.2], [11.0, 2.2], [11.0, 2.2], [9.0, 2.2], [10.0, 2.3], [16.0, 2.3], [7.0, 2.3], [6.0, 2.3], [15.0, 2.3], [9.0, 2.3], [13.0, 2.3], [5.0, 2.3], [15.0, 2.3], [6.0, 2.3], [7.0, 2.3], [13.0, 2.3], [15.0, 2.3], [6.0, 2.3], [11.0, 2.3], [7.0, 2.3], [6.0, 2.4], [16.0, 2.4], [12.0, 2.4], [8.0, 2.4], [7.0, 2.4], [10.0, 2.4], [9.0, 2.4], [9.0, 2.4], [16.0, 2.4], [7.0, 2.4], [12.0, 2.4], [15.0, 2.4], [11.0, 2.4], [11.0, 2.4], [13.0, 2.4], [12.0, 2.4], [12.0, 2.4], [8.0, 2.4], [10.0, 2.4], [6.0, 2.5], [14.0, 2.5], [12.0, 2.5], [14.0, 2.5], [12.0, 2.5], [6.0, 2.5], [7.0, 2.5], [15.0, 2.5], [6.0, 2.5], [15.0, 2.5], [7.0, 2.5], [15.0, 2.5], [9.0, 2.5], [10.0, 2.5], [6.0, 2.5], [6.0, 2.5], [15.0, 2.5], [10.0, 2.5], [7.0, 2.5], [14.0, 2.5], [14.0, 2.5], [11.0, 2.5], [9.0, 2.5], [15.0, 2.5], [10.0, 2.6], [12.0, 2.6], [5.0, 2.6], [12.0, 2.6], [8.0, 2.6], [13.0, 2.6], [10.0, 2.6], [12.0, 2.6], [16.0, 2.6], [6.0, 2.6], [6.0, 2.6], [10.0, 2.6], [7.0, 2.6], [12.0, 2.6], [13.0, 2.6], [6.0, 2.7], [10.0, 2.7], [7.0, 2.7], [7.0, 2.7], [11.0, 2.7], [12.0, 2.7], [5.0, 2.7], [9.0, 2.7], [7.0, 2.7], [14.0, 2.7], [16.0, 2.7], [11.0, 2.7], [8.0, 2.7], [16.0, 2.7], [14.0, 2.7], [7.0, 2.7], [8.0, 2.7], [6.0, 2.7], [6.0, 2.7], [9.0, 2.7], [6.0, 2.7], [15.0, 2.7], [14.0, 2.8], [9.0, 2.8], [11.0, 2.8], [8.0, 2.8], [7.0, 2.8], [9.0, 2.8], [10.0, 2.8], [10.0, 2.8], [7.0, 2.8], [9.0, 2.8], [12.0, 2.8], [12.0, 2.8], [13.0, 2.8], [11.0, 2.8], [8.0, 2.8], [10.0, 2.8], [5.0, 2.8], [12.0, 2.8], [15.0, 2.8], [6.0, 2.8], [7.0, 2.8], [15.0, 2.8], [14.0, 2.8], [15.0, 2.8], [10.0, 2.9], [14.0, 2.9], [15.0, 2.9], [6.0, 2.9], [5.0, 2.9], [16.0, 2.9], [5.0, 2.9], [8.0, 2.9], [14.0, 2.9], [12.0, 2.9], [9.0, 2.9], [13.0, 2.9], [10.0, 2.9], [15.0, 2.9], [8.0, 2.9], [13.0, 2.9], [13.0, 2.9], [7.0, 2.9], [12.0, 2.9], [33.0, 4.0], [37.0, 4.0], [33.0, 4.0], [30.0, 4.0], [40.0, 4.0], [36.0, 4.0], [31.0, 4.0], [31.0, 4.0], [39.0, 4.0], [38.0, 4.0], [30.0, 4.0], [33.0, 4.0], [32.0, 4.0], [33.0, 4.0], [33.0, 4.0], [31.0, 4.0], [38.0, 4.0], [32.0, 4.1], [29.0, 4.1], [40.0, 4.1], [30.0, 4.1], [37.0, 4.1], [35.0, 4.1], [35.0, 4.1], [36.0, 4.1], [35.0, 4.1], [33.0, 4.1], [28.0, 4.1], [36.0, 4.1], [35.0, 4.1], [30.0, 4.1], [32.0, 4.1], [33.0, 4.1], [29.0, 4.1], [40.0, 4.1], [39.0, 4.1], [37.0, 4.1], [34.0, 4.1], [31.0, 4.1], [31.0, 4.1], [32.0, 4.1], [29.0, 4.2], [32.0, 4.2], [32.0, 4.2], [31.0, 4.2], [33.0, 4.2], [33.0, 4.2], [33.0, 4.2], [36.0, 4.2], [36.0, 4.2], [29.0, 4.2], [32.0, 4.2], [30.0, 4.2], [31.0, 4.2], [31.0, 4.2], [28.0, 4.2], [32.0, 4.2], [32.0, 4.2], [35.0, 4.2], [28.0, 4.2], [39.0, 4.2], [36.0, 4.2], [39.0, 4.2], [32.0, 4.3], [37.0, 4.3], [35.0, 4.3], [39.0, 4.3], [38.0, 4.3], [38.0, 4.3], [30.0, 4.3], [33.0, 4.3], [38.0, 4.3], [40.0, 4.3], [30.0, 4.3], [34.0, 4.3], [35.0, 4.3], [40.0, 4.3], [40.0, 4.3], [36.0, 4.3], [38.0, 4.3], [40.0, 4.3], [31.0, 4.3], [36.0, 4.3], [28.0, 4.3], [34.0, 4.3], [35.0, 4.3], [32.0, 4.4], [37.0, 4.4], [35.0, 4.4], [38.0, 4.4], [31.0, 4.4], [30.0, 4.4], [37.0, 4.4], [28.0, 4.4], [29.0, 4.4], [38.0, 4.5], [37.0, 4.5], [37.0, 4.5], [31.0, 4.5], [31.0, 4.5], [40.0, 4.5], [40.0, 4.5], [38.0, 4.5], [29.0, 4.5], [37.0, 4.5], [32.0, 4.5], [32.0, 4.5], [40.0, 4.5], [36.0, 4.5], [33.0, 4.5], [38.0, 4.5], [37.0, 4.5], [40.0, 4.5], [33.0, 4.5], [28.0, 4.5], [32.0, 4.5], [37.0, 4.5], [29.0, 4.5], [37.0, 4.6], [39.0, 4.6], [29.0, 4.6], [37.0, 4.6], [39.0, 4.6], [31.0, 4.6], [31.0, 4.6], [30.0, 4.6], [36.0, 4.6], [35.0, 4.6], [40.0, 4.6], [34.0, 4.6], [31.0, 4.6], [28.0, 4.6], [34.0, 4.6], [29.0, 4.6], [28.0, 4.6], [33.0, 4.6], [37.0, 4.6], [37.0, 4.6], [30.0, 4.6], [31.0, 4.6], [32.0, 4.7], [34.0, 4.7], [34.0, 4.7], [32.0, 4.7], [30.0, 4.7], [33.0, 4.7], [35.0, 4.7], [32.0, 4.7], [31.0, 4.7], [35.0, 4.7], [40.0, 4.7], [40.0, 4.7], [32.0, 4.7], [30.0, 4.7], [29.0, 4.7], [33.0, 4.7], [31.0, 4.8], [31.0, 4.8], [37.0, 4.8], [40.0, 4.8], [34.0, 4.8], [40.0, 4.8], [31.0, 4.8], [35.0, 4.8], [38.0, 4.8], [40.0, 4.8], [29.0, 4.8], [29.0, 4.8], [31.0, 4.8], [33.0, 4.8], [36.0, 4.8], [28.0, 4.8], [30.0, 4.8], [36.0, 4.8], [28.0, 4.8], [36.0, 4.8], [34.0, 4.8], [39.0, 4.9], [37.0, 4.9], [31.0, 4.9], [33.0, 4.9], [29.0, 4.9], [30.0, 4.9], [32.0, 4.9], [29.0, 4.9], [32.0, 4.9], [39.0, 4.9], [33.0, 4.9], [35.0, 4.9], [32.0, 4.9], [34.0, 4.9], [40.0, 4.9], [31.0, 4.9], [39.0, 4.9], [37.0, 4.9], [35.0, 4.9], [35.0, 4.9], [35.0, 4.9], [40.0, 4.9], [28.0, 4.9], [39.0, 4.9], [1.0, 0.0], [10.0, 0.0], [5.0, 0.0], [6.0, 0.0], [5.0, 0.0], [17.0, 0.0], [16.0, 0.0], [2.0, 0.0], [12.0, 0.0], [26.0, 1.0], [-1.0, 1.0], [19.0, 1.0], [14.0, 1.0], [-12.0, 1.0], [13.0, 1.0], [26.0, 1.0], [16.0, 1.0], [19.0, 1.0], [-5.0, 2.0], [-1.0, 2.0], [38.0, 2.0], [22.0, 2.0], [18.0, 2.0], [12.0, 2.0], [7.0, 2.0], [17.0, 2.0], [-19.0, 2.0], [18.0, 2.0], [-4.0, 3.0], [-9.0, 3.0], [13.0, 3.0], [33.0, 3.0], [5.0, 3.0], [40.0, 3.0], [11.0, 3.0], [14.0, 3.0], [31.0, 3.0], [26.0, 3.0], [22.0, 4.0], [33.0, 4.0], [39.0, 4.0], [25.0, 4.0], [3.0, 4.0], [4.0, 4.0], [13.0, 4.0], [1.0, 4.0], [22.0, 4.0], [-16.0, 4.0]]

[[ 10.49568966 2.3625 ]

[-13.97073171 0.51658537]

[ 33.83412322 4.37725118]]

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

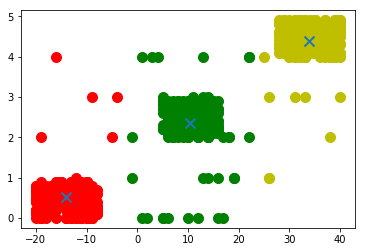
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 2 0 0 0 1 0 2 0 0 1 0 2 0 0 0 0 0 1 0 1

1 0 2 0 2 0 0 2 2 0 2 2 2 0 0 0 0 0 1]



9. 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.

#1.Import Data

from sklearn.datasets import load\_iris

iris = load\_iris()

print("Feature Names:",iris.feature\_names,"Iris Data:",iris.data,"Target Names:",iris.target\_names,"Target:",iris.target)

#2. Split the data into Test and Data

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

iris.data, iris.target, test\_size = .25)

#neighbors\_settings = range(1, 11)

#for n\_neighbors in neighbors\_settings:

#3.Build The Model

from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()

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

#4.Calculate Accuracy of the Test data with the trained data

print(" Accuracy=",clf.score(X\_test, y\_test))

#5 Calculate the prediction with the labels of the test data

print("Predicted Data")

print(clf.predict(X\_test))

prediction=clf.predict(X\_test)

print("Test data :")

print(y\_test)

#6 To identify the miss classification

diff=prediction-y\_test

print("Result is ")

print(diff)

print('Total no of samples misclassied =', sum(abs(diff)))

**OUTPUT**

('Feature Names:', ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'Iris Data:', array([[5.1, 3.5, 1.4, 0.2],

[4.9, 3. , 1.4, 0.2],

[4.7, 3.2, 1.3, 0.2],

[4.6, 3.1, 1.5, 0.2],

[5. , 3.6, 1.4, 0.2],

[5.4, 3.9, 1.7, 0.4],

[4.6, 3.4, 1.4, 0.3],

[5. , 3.4, 1.5, 0.2],

[4.4, 2.9, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.1],

[5.4, 3.7, 1.5, 0.2],

[4.8, 3.4, 1.6, 0.2],

[4.8, 3. , 1.4, 0.1],

[4.3, 3. , 1.1, 0.1],

[5.8, 4. , 1.2, 0.2],

[5.7, 4.4, 1.5, 0.4],

[5.4, 3.9, 1.3, 0.4],

[5.1, 3.5, 1.4, 0.3],

[5.7, 3.8, 1.7, 0.3],

[5.1, 3.8, 1.5, 0.3],

[5.4, 3.4, 1.7, 0.2],

[5.1, 3.7, 1.5, 0.4],

[4.6, 3.6, 1. , 0.2],

[5.1, 3.3, 1.7, 0.5],

[4.8, 3.4, 1.9, 0.2],

[5. , 3. , 1.6, 0.2],

[5. , 3.4, 1.6, 0.4],

[5.2, 3.5, 1.5, 0.2],

[5.2, 3.4, 1.4, 0.2],

[4.7, 3.2, 1.6, 0.2],

[4.8, 3.1, 1.6, 0.2],

[5.4, 3.4, 1.5, 0.4],

[5.2, 4.1, 1.5, 0.1],

[5.5, 4.2, 1.4, 0.2],

[4.9, 3.1, 1.5, 0.1],

[5. , 3.2, 1.2, 0.2],

[5.5, 3.5, 1.3, 0.2],

[4.9, 3.1, 1.5, 0.1],

[4.4, 3. , 1.3, 0.2],

[5.1, 3.4, 1.5, 0.2],

[5. , 3.5, 1.3, 0.3],

[4.5, 2.3, 1.3, 0.3],

[4.4, 3.2, 1.3, 0.2],

[5. , 3.5, 1.6, 0.6],

[5.1, 3.8, 1.9, 0.4],

[4.8, 3. , 1.4, 0.3],

[5.1, 3.8, 1.6, 0.2],

[4.6, 3.2, 1.4, 0.2],

[5.3, 3.7, 1.5, 0.2],

[5. , 3.3, 1.4, 0.2],

[7. , 3.2, 4.7, 1.4],

[6.4, 3.2, 4.5, 1.5],

[6.9, 3.1, 4.9, 1.5],

[5.5, 2.3, 4. , 1.3],

[6.5, 2.8, 4.6, 1.5],

[5.7, 2.8, 4.5, 1.3],

[6.3, 3.3, 4.7, 1.6],

[4.9, 2.4, 3.3, 1. ],

[6.6, 2.9, 4.6, 1.3],

[5.2, 2.7, 3.9, 1.4],

[5. , 2. , 3.5, 1. ],

[5.9, 3. , 4.2, 1.5],

[6. , 2.2, 4. , 1. ],

[6.1, 2.9, 4.7, 1.4],

[5.6, 2.9, 3.6, 1.3],

[6.7, 3.1, 4.4, 1.4],

[5.6, 3. , 4.5, 1.5],

[5.8, 2.7, 4.1, 1. ],

[6.2, 2.2, 4.5, 1.5],

[5.6, 2.5, 3.9, 1.1],

[5.9, 3.2, 4.8, 1.8],

[6.1, 2.8, 4. , 1.3],

[6.3, 2.5, 4.9, 1.5],

[6.1, 2.8, 4.7, 1.2],

[6.4, 2.9, 4.3, 1.3],

[6.6, 3. , 4.4, 1.4],

[6.8, 2.8, 4.8, 1.4],

[6.7, 3. , 5. , 1.7],

[6. , 2.9, 4.5, 1.5],

[5.7, 2.6, 3.5, 1. ],

[5.5, 2.4, 3.8, 1.1],

[5.5, 2.4, 3.7, 1. ],

[5.8, 2.7, 3.9, 1.2],

[6. , 2.7, 5.1, 1.6],

[5.4, 3. , 4.5, 1.5],

[6. , 3.4, 4.5, 1.6],

[6.7, 3.1, 4.7, 1.5],

[6.3, 2.3, 4.4, 1.3],

[5.6, 3. , 4.1, 1.3],

[5.5, 2.5, 4. , 1.3],

[5.5, 2.6, 4.4, 1.2],

[6.1, 3. , 4.6, 1.4],

[5.8, 2.6, 4. , 1.2],

[5. , 2.3, 3.3, 1. ],

[5.6, 2.7, 4.2, 1.3],

[5.7, 3. , 4.2, 1.2],

[5.7, 2.9, 4.2, 1.3],

[6.2, 2.9, 4.3, 1.3],

[5.1, 2.5, 3. , 1.1],

[5.7, 2.8, 4.1, 1.3],

[6.3, 3.3, 6. , 2.5],

[5.8, 2.7, 5.1, 1.9],

[7.1, 3. , 5.9, 2.1],

[6.3, 2.9, 5.6, 1.8],

[6.5, 3. , 5.8, 2.2],

[7.6, 3. , 6.6, 2.1],

[4.9, 2.5, 4.5, 1.7],

[7.3, 2.9, 6.3, 1.8],

[6.7, 2.5, 5.8, 1.8],

[7.2, 3.6, 6.1, 2.5],

[6.5, 3.2, 5.1, 2. ],

[6.4, 2.7, 5.3, 1.9],

[6.8, 3. , 5.5, 2.1],

[5.7, 2.5, 5. , 2. ],

[5.8, 2.8, 5.1, 2.4],

[6.4, 3.2, 5.3, 2.3],

[6.5, 3. , 5.5, 1.8],

[7.7, 3.8, 6.7, 2.2],

[7.7, 2.6, 6.9, 2.3],

[6. , 2.2, 5. , 1.5],

[6.9, 3.2, 5.7, 2.3],

[5.6, 2.8, 4.9, 2. ],

[7.7, 2.8, 6.7, 2. ],

[6.3, 2.7, 4.9, 1.8],

[6.7, 3.3, 5.7, 2.1],

[7.2, 3.2, 6. , 1.8],

[6.2, 2.8, 4.8, 1.8],

[6.1, 3. , 4.9, 1.8],

[6.4, 2.8, 5.6, 2.1],

[7.2, 3. , 5.8, 1.6],

[7.4, 2.8, 6.1, 1.9],

[7.9, 3.8, 6.4, 2. ],

[6.4, 2.8, 5.6, 2.2],

[6.3, 2.8, 5.1, 1.5],

[6.1, 2.6, 5.6, 1.4],

[7.7, 3. , 6.1, 2.3],

[6.3, 3.4, 5.6, 2.4],

[6.4, 3.1, 5.5, 1.8],

[6. , 3. , 4.8, 1.8],

[6.9, 3.1, 5.4, 2.1],

[6.7, 3.1, 5.6, 2.4],

[6.9, 3.1, 5.1, 2.3],

[5.8, 2.7, 5.1, 1.9],

[6.8, 3.2, 5.9, 2.3],

[6.7, 3.3, 5.7, 2.5],

[6.7, 3. , 5.2, 2.3],

[6.3, 2.5, 5. , 1.9],

[6.5, 3. , 5.2, 2. ],

[6.2, 3.4, 5.4, 2.3],

[5.9, 3. , 5.1, 1.8]]), 'Target Names:', array(['setosa', 'versicolor', 'virginica'], dtype='|S10'), 'Target:', array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]))

(' Accuracy=', 0.9736842105263158)

Predicted Data

[2 1 0 0 2 2 1 2 2 0 1 2 2 2 2 1 1 1 2 2 0 2 2 0 0 0 2 2 2 2 2 0 2 0 0 1 2 0]

Test data :

[2 1 0 0 2 2 1 2 2 0 2 2 2 2 2 1 1 1 2 2 0 2 2 0 0 0 2 2 2 2 2 0 2 0 0 1 2 0]

Result is

[ 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0]

('Total no of samples misclassied =', 1)

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

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#the Gaussian Kernel

def kernel(point,xmat, k):

m,n = np.shape(xmat)

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

for j in range(m):

diff = point - X[j]

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

return weights

#Weigh each point by its distance to the reference point. We are considering

# All points here. If KNN was the topic, we could restrict this to "K"

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 = np.shape(xmat)

ypred = np.zeros(m)

for i in range(m):

# predicted value y = wx. Here w = weights we have computed.

# Remember that both w and x are vectors here (2\*1 and 1\*2 respectively)

# Resultant value of y is a scalar

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

return ypred

# load data points

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

colA = np.array(data.colA)

colB = np.array(data.colB)

#preparing and add 1

#convert to matrix form

mcolA = np.mat(colA)

mcolB = np.mat(colB)

m= np.shape(mcolA)[1]

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

#horizontally stack

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

print(X.shape)

#set k here (0.5)

ypred = localWeightRegression(X,mcolB,0.5)

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

xsort = X[SortIndex][:,0]

fig = plt.figure()

ax = fig.add\_subplot(1,1,1)

ax.scatter(colA,colB, color='green')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

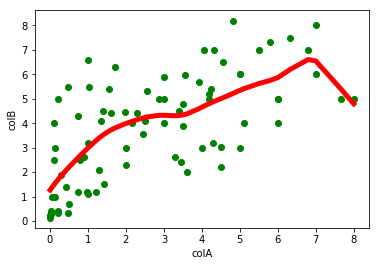
plt.xlabel('colA')

plt.ylabel('colB')

plt.show();

**OUTPUT**

**(80, 2)**

****