PROGRAM 1

from heuristicsearch.a\_star\_search import AStar

graph\_nodes = {

'A': [('B', 6), ('F', 3)],

'B': [('C', 3), ('D', 2)],

'C': [('D', 1), ('E', 5)],

'D': [('C', 1), ('E', 8)],

'E': [('I', 5), ('J', 5)],

'F': [('G', 1),('H', 7)] ,

'G': [('I', 3)],

'H': [('I', 2)],

'I': [('E', 5), ('J', 3)],

}

heuristics = {

'A': 10,

'B': 8,

'C': 5,

'D': 7,

'E': 3,

'F': 6,

'G': 5,

'H': 3,

'I': 1,

'J': 0

}

graph= AStar(graph\_nodes,heuristics)

graph.apply\_a\_star(start='A', stop='J')

PROGRAM 2

from heuristicsearch.ao\_star import AOStar

print("Graph-1")

heuristic = {'S': 1, 'A': 7, 'B': 12, 'C': 13, 'D': 5, 'E': 6, 'F': 5, 'G': 7, 'H': 2,}

adjacency\_matrix = {

'S': [[('A', 1), ('B', 1)], [('C', 1)]],

'A': [[('D', 1)], [('E', 1)]],

'C': [[('F', 1), ('G', 1)]],

'D': [('H', 1)]

}

graph=AOStar(adjacency\_matrix,heuristic,'S')

graph.applyAOStar()

PROGRAM 3

import csv

a=[]

with open("enjoysport.csv","r") as csvfile:

fdata=csv.reader(csvfile)

for row in fdata:

a.append(row)

print(row)

num\_att=len(a[0])-1

S=['0']\*num\_att

G=['?']\*num\_att

print(S)

print(G)

temp=[]

for i in range(0,num\_att):

S[i]=a[0][i]

print("................................................")

for i in range(0,len(a)):

if a[i][num\_att]=="Yes":

for j in range(0,num\_att):

if S[j]!=a[i][j]:

S[j]='?'

for j in range(0,num\_att):

for k in range(0,len(temp)):

if temp[k][j]!=S[j] and temp[k][j]!='?':

del temp[k]

if a[i][num\_att]=='No':

for j in range(0,num\_att):

if a[i][j]!=S[j] and S[j]!='?':

G[j]=S[j]

temp.append(G)

G=['?']\*num\_att

print(S)

if len(temp)==0:

print(G)

else:

print(temp)

print("......................................................................")

PROGRAM 4

import pandas as pd

import math

df = pd.read\_csv('/Users/Documents/Python Scripts/PlayTennis.csv')

print("\n Input Data Set is:\n", df)

t = df.keys()[-1]

print('Target Attribute is: ', t)

attribute\_names = list(df.keys())

attribute\_names.remove(t)

print('Predicting Attributes: ', attribute\_names)

def entropy(probs):

return sum( [-prob\*math.log(prob, 2) for prob in probs])

def entropy\_of\_list(ls,value):

from collections import Counter

cnt = Counter(x for x in ls)# Counter calculates the propotion of class

print('Target attribute class count(Yes/No)=',dict(cnt))

total\_instances = len(ls)

print("Total no of instances/records associated with {0} is: {1}".format(value,total\_instances ))

probs = [x / total\_instances for x in cnt.values()] # x means no of YES/NO

print("Probability of Class {0} is: {1:.4f}".format(min(cnt),min(probs)))

print("Probability of Class {0} is: {1:.4f}".format(max(cnt),max(probs)))

return entropy(probs) # Call Entropy

def information\_gain(df, split\_attribute, target\_attribute,battr):

print("\n\n-----Information Gain Calculation of ",split\_attribute, " --------")

df\_split = df.groupby(split\_attribute) # group the data based on attribute values

glist=[]

for gname,group in df\_split:

print('Grouped Attribute Values \n',group)

glist.append(gname)

glist.reverse()

nobs = len(df.index) \* 1.0

df\_agg1=df\_split.agg({target\_attribute:lambda x:entropy\_of\_list(x, glist.pop())})

df\_agg2=df\_split.agg({target\_attribute :lambda x:len(x)/nobs})

df\_agg1.columns=['Entropy']

df\_agg2.columns=['Proportion']

new\_entropy = sum( df\_agg1['Entropy'] \* df\_agg2['Proportion'])

if battr !='S':

old\_entropy = entropy\_of\_list(df[target\_attribute],'S-'+df.iloc[0][df.columns.get\_loc(battr)])

else:

old\_entropy = entropy\_of\_list(df[target\_attribute],battr)

return old\_entropy - new\_entropy

def id3(df, target\_attribute, attribute\_names, default\_class=None,default\_attr='S'):

from collections import Counter

cnt = Counter(x for x in df[target\_attribute])# class of YES /NO

if len(cnt) == 1:

return next(iter(cnt)) # next input data set, or raises StopIteration when EOF is hit.

elif df.empty or (not attribute\_names):

return default\_class # Return None for Empty Data Set

else:

default\_class = max(cnt.keys()) #No of YES and NO Class

gainz=[]

for attr in attribute\_names:

ig= information\_gain(df, attr, target\_attribute,default\_attr)

gainz.append(ig)

print('Information gain of ',attr,' is : ',ig)

index\_of\_max = gainz.index(max(gainz))

best\_attr = attribute\_names[index\_of\_max

print("\nAttribute with the maximum gain is: ", best\_attr)

tree = {best\_attr:{}} # Initiate the tree with best attribute as a node

remaining\_attribute\_names =[i for i in attribute\_names if i != best\_attr]

subtrees, which

for attr\_val, data\_subset in df.groupby(best\_attr):

subtree = id3(data\_subset,target\_attribute,

remaining\_attribute\_names,default\_class,best\_attr)

tree[best\_attr][attr\_val] = subtree

return tree

from pprint import pprint

tree = id3(df,t,attribute\_names)

print("\nThe Resultant Decision Tree is:")

print(tree)

def classify(instance, tree,default=None): # Instance of Play Tennis with Predicted

attribute = next(iter(tree)) # Outlook/Humidity/Wind

if instance[attribute] in tree[attribute].keys(): # Value of the attributs in set of Tree keys

result = tree[attribute][instance[attribute]]

if isinstance(result, dict): # this is a tree, delve deeper

return classify(instance, result)

else:

return result # this is a label

else:

return default

df\_new=pd.read\_csv('/Users/Documents/Python Scripts/PlayTennisTest.csv')

df\_new['predicted'] = df\_new.apply(classify, axis=1, args=(tree,'?'))

print(df\_new)

PROGRAM 5

import numpy as np

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

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

y = y/100

def sigmoid(x):

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

def derivatives\_sigmoid(x):

return x \* (1 - x)

epoch=10000

lr=0.1

inputlayer\_neurons = 2

hiddenlayer\_neurons = 3

output\_neurons = 1

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bias\_hidden=np.random.uniform(size=(1,hiddenlayer\_neurons))

weight\_hidden=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bias\_output=np.random.uniform(size=(1,output\_neurons))

for i in range(epoch):

hinp1=np.dot(X,wh)

hinp= hinp1 + bias\_hidden

hlayer\_activation = sigmoid(hinp)

outinp1=np.dot(hlayer\_activation,weight\_hidden)

outinp= outinp1+ bias\_output

output = sigmoid(outinp)

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(weight\_hidden.T)

hiddengrad = derivatives\_sigmoid(hlayer\_activation)

d\_hiddenlayer = EH \* hiddengrad

weight\_hidden += hlayer\_activation.T.dot(d\_output) \*lr

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

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

bias\_output += np.sum(d\_output, axis=0,keepdims=True) \*lr

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

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

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

PROGRAM 6

import numpy as np

import math

import csv

import pdb

def read\_data(filename):

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

datareader = csv.reader(csvfile)

metadata = next(datareader)

traindata=[]

for row in datareader:

traindata.append(row)

return (metadata, traindata)

def splitDataset(dataset, splitRatio):

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

trainSet = []

testset = list(dataset)

i=0

while len(trainSet) < trainSize:

trainSet.append(testset.pop(i))

return [trainSet, testset]

def classify(data,test):

total\_size = data.shape[0]

print("\n")

print("training data size=",total\_size)

print("test data size=",test.shape[0])

countYes = 0

countNo = 0

probYes = 0

probNo = 0

print("\n")

print("target count probability")

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

if data[x,data.shape[1]-1] == 'yes':

countYes +=1

if data[x,data.shape[1]-1] == 'no':

countNo +=1

probYes=countYes/total\_size

probNo= countNo / total\_size

print('Yes',"\t",countYes,"\t",probYes)

print('No',"\t",countNo,"\t",probNo)

prob0 =np.zeros((test.shape[1]-1))

prob1 =np.zeros((test.shape[1]-1))

accuracy=0

print("\n")

print("instance prediction target")

for t in range(test.shape[0]):

for k in range (test.shape[1]-1):

count1=count0=0

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

if test[t,k] == data[j,k] and data[j,data.shape[1]-1]=='no':

count0+=1

if test[t,k]==data[j,k] and data[j,data.shape[1]-1]=='yes':

count1+=1

prob0[k]=count0/countNo

prob1[k]=count1/countYes

probno=probNo

probyes=probYes

for i in range(test.shape[1]-1):

probno=probno\*prob0[i]

probyes=probyes\*prob1[i]

if probno>probyes:

predict='no'

else:

predict='yes'

print(t+1,"\t",predict,"\t ",test[t,test.shape[1]-1])

if predict == test[t,test.shape[1]-1]:

accuracy+=1

final\_accuracy=(accuracy/test.shape[0])\*100

print("accuracy",final\_accuracy,"%")

return

metadata,traindata= read\_data("/Users/Chachu/Documents/Python Scripts/tennis.csv")

splitRatio=0.6

trainingset, testset=splitDataset(traindata, splitRatio)

training=np.array(trainingset)

print("\n The Training data set are:")

for x in trainingset:

print(x)

testing=np.array(testset)

print("\n The Test data set are:")

for x in testing:

print(x)

classify(training,testing)

PROGRAM 7

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

iris = datasets.load\_iris()

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

model = KMeans(n\_clusters=3)

model.fit(X) # model.labels\_ : Gives cluster no for which samples belongs to

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

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

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')

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')

from sklearn import preprocessing

deviation of 1.

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

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

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

gmm\_y = gmm.predict(xs)

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.')

PROGRAM 8

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

# Load dataset

iris=datasets.load\_iris()

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

# Split the data into train and test samples

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

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

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

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

# Prints Label no. and their names

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)

# Perform Training

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));

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

print('Confusion Matrix')

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

print('Accuracy Metrics')

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

PROGRAM 9

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

def kernel(point,xmat, k):

m,n = np.shape(xmat)

weights = np.mat(np.eye((m))) # eye - identity matrix

for j in range(m):

diff = point - X[j]

weights[j,j] = np.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 = np.shape(xmat)

ypred = np.zeros(m)

for i in range(m):

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

return ypred

def graphPlot(X,ypred):

sortindex = X[:,1].argsort(0) #argsort - index of the smallest

xsort = X[sortindex][:,0]

fig = plt.figure()

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

ax.scatter(bill,tip, color='green')

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

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();

# load data points

data = pd.read\_csv('/Users/Chachu/Documents/Python Scripts/data10\_tips.csv')

bill = np.array(data.total\_bill) # We use only Bill amount and Tips data

tip = np.array(data.tip)

mbill = np.mat(bill) # .mat will convert nd array is converted in 2D array

mtip = np.mat(tip)

m= np.shape(mbill)[1]

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

X = np.hstack((one.T,mbill.T)) # 244 rows, 2 cols

ypred = localWeightRegression(X,mtip,2) # increase k to get smooth curves

graphPlot(X,ypred)