BFS:

graph = {

'5' : ['3','7'],

'3' : ['2','4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = [] #list for visited nodes.

queue = [] #initialize a queue

def bfs(visited, graph, node): #function for bfs

visited.append(node)

queue.append(node)

while queue: #creating loop to visit each node

m = queue.pop(0)

print (m, end = " ")

for neighbour in graph[m]:

if neighbourvnot in visited:

visited.append(neighbour)

queue.append(neighbour)

#driver code

print("following is the Breadth-First Search")

bfs(visited, graph, 's') #function calling

Decision Tree:

#import libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

then enter

df = pd.read\_csv("diabetes.csv")

enter

df.head(10)

enter

df.tail(10)

enter

df.describe()

enter

df.info()

enter

df.isnull().values.any()

enter

df.shape

enter

x = df.iloc[:, 0:-1]

y = df.iloc[:, -1]

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size=0.2, random\_state=2020)

print('shape of x\_train = ', x\_train.shape)

print('shape of x\_test = ', x\_test.shape)

print('shape of y\_train = ', y\_train.shape)

print('shape of y\_test = ', y\_test.shape)

enter

from sklearn.tree import DecisionTreeClassifier

enter

classifier = DecisionTreeClassifier(criterion='gin1')

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

enter

classifier.score(x\_test, y\_test)

enter

classifier\_entropy = DecisionTreeClassifier(criterion='entropy')

classifier\_entropy.fit(x\_train, y\_train)

enter

classifier\_entropy.score(x\_test, y\_test)

enter

from sklearn.preprocessing import standardscaler

sc = standardscaler()

enter

sc.fit(x\_train)

enter

x\_train\_sc = sc.transform(x\_test)

x\_test\_sc = sc.transform(x\_test)

enter

classifer\_sc = DecisionTreeClassifer(criterion='gin1')

classifer\_sc.fit(x\_train\_sc, y\_train)

classifier\_sc.score(x\_test\_sc, y\_test)

uniformed searches (Dfs):

graph = {

'5' : ['3','8'],

'3' : ['2', '4'],

'8' : ['7'],

'2' : [],

'4' : [],

'7' : []

}

visited = set()

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")

dfs(visited, graph '5')

KNN:

import pandas as pd

from sklearn.datasets import load\_iris

iris = load\_iris()

enter

iris.feature\_names

enter

iris.target\_names

enter

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df.head()

enter

df['target'] = iris.target

df.head()

enter

df[df.target==1]

enter

df[df.target==2].head()

enter

df['flower\_name'] = df.target.apply(lambda x: iris.target\_names[x])

df.head()

enter

df[45:55]

enter

df0 = df[:50]

df1 = df[50:100]

df2 = df[100:]

enter

import matplotlib.pyplot as plt

%matplotlib inline

enter

plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'], color="green", marker='+')

ply.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'], color="blue" , marker='0')

same petal

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

enter

x = df.drop(['target', 'flower\_name'], axis='columns')

y = df.target

enter

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x,y, test\_size=0.2, random\_state=1)

enter

len(x\_train)

enter

len(x\_test)

enter

from sklearn.neighbours import KNeighboursClassifier

knn = KNeighbourClassifier(n\_neighbour = 10)

enter

knn.fit(x\_train, y\_train)

enter

knn.score(x\_test, y\_test)

enter

from sklearn.metrics import confusion\_matrix

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

cm = confusion\_matrix(y\_test, y\_pred)

cm

enter

%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(7,5))

sns.heatmap(cm, annot=True)

plt.xlabel('predicted')

plt.ylabel('Truth')

enter

from sklearn.metrics import classification\_report

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

Linear Regression:

#importing Libraries

import numpy as np

import matplot.pyplot as plt

import pandas as pd

import seaborn as sns

enter

companies = pd.read\_csv('etc.csv')

x = companies.iloc[:, :-1].values

y = companies.iloc[:, 4].values

enter

companies.head()

enter

#Encoding categorical data

from sklearn.preprosessing import LabelEncoder, OneHotEncoder

labelencoder = LabelEncoder()

x[:, 3] = labelencoder.fit\_transform(x[:, 3])

# onehotencoder = OneHotEncoder(categorical\_features = [2])

# x = onehotencoder.fit\_transform(x).toarray()

enter

x = x[:, 1:]

enter

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, test\_size = 0.2, random\_state = 0)

enter

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

enter

#predicting the test set results

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

y\_pred

enter

#calculating the coefficients

print(regressor.coef\_)

enter

#calculating the intercept

print(regressor.intercept)

enter

#calculating the R squared Value

from sklearn.metrics import r2\_score

r2\_score(y\_test, y\_pred)

ANN:

# import dependencies

import numpy as np

# x = (hours sleeping, hours studying), y = test score of the student

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

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

enter

# scale units

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

y = y/100

enter

Class NeuralNetwork(object):

def\_init\_(self):

#parameters

self.inputsize = 2

self.outputsize = 1

self.hiddensize = 3

#weights

self.w1 = np.random.randn(self.inputsize, self.hiddensize)

self.w2 = np.random.randn(self.hiddensize, self.outputsize)

def feedforward(self, x):

#forward propogation through the network

self.z = np.dot(x, self.w1)

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

self.z3 = np.dot(self.z2, self.w2)

output = self.sigmoid(self.z3)

return output

def sigmoid(self, s, deriv=False):

if(deriv==True):

return s\*(1 - 5)

return 1/(1+np.exp(-5))

def backward(self, x, y, output):

#backward propogate through the network

self.output\_error = y-output

self.output\_delta = self.output\_error\*self.sigmoid(output, deriv=True)

self.z2\_error = self.output\_delta.dot(self.w2.T)

self.z2\_error = self.z2\_error\*self.sigmoid(self.z2, deriv=True)

self.w1 += x.T.dot(self.z2\_delta)

self.w2 += self.z2.T.dot(self.output\_delta)

def train(self, x, y):

output = self.feedforward(x)

self.backward(x, y, output)

print("Input: "+ str(x))

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

print("Loss: "+ str(np.mean(np.square(y - NN.feedForward(x)))))

print("\n")

print("predicted output: "+ str(NN.feedForward(x)))