# INDEX

|  |  |  |
| --- | --- | --- |
| **Week- No** | **List of Programs** | **Pg Nos.** |
| 1 | Write a python program to import and export data using Pandas library functions |  |
| 2 | Demonstrate various data pre-processing techniques for a given dataset |  |
| 3 | Implement Dimensionality reduction using Principle Component Analysis (PCA) method. |  |
| 4 | Write a Python program to demonstrate various Data  Visualization Techniques. |  |
| 5 | Implement Simple and Multiple Linear Regression Models. |  |
| 6 | Develop Logistic Regression Model for a given dataset. |  |
| 7 | Develop Decision Tree Classification model for a given dataset and use it to classify a new sample. |  |
| 8 | Implement Naïve Bayes Classification in Python |  |
| 9 | Build KNN Classification model for a given dataset. |  |
| 10 | Build Artificial Neural Network model with back propagation on a given dataset. |  |
| 11 | 1. Implement Random forest ensemble method on a given dataset. 2. Implement Boosting ensemble method on a given dataset. |  |
| 12 | Write a python program to implement K-Means clustering Algorithm. |  |

## Write a python program to import and export the data using pandas library

1. **Manual Function**

def load\_csv(filepath): data = []

col = [] checkcol = False

with open(filepath) as f: for val in f.readlines():

val = val.replace("\n","")

val = val.split(',')

if checkcol is False: col = val checkcol = True

else:

data.append(val)

df = pd.DataFrame(data=data, columns=col) return df

## Numpy.loadtxt function

df = np.loadtxt('convertcsv.csv', delimeter = ',') print(df[:5,:])

## Numpy.genfromtxt()

data = np.genfromtxt('100 Sales Records.csv', delimiter=',')

>>> pd.DataFrame(data)

## Pandas.read\_csv()

>>> pdDf = pd.read\_csv('100 Sales Record.csv')

>>> pdDf.head()

## Pickle

with open('test.pkl','wb') as f: pickle.dump(pdDf, f)

Data preprocessing

* 1. **Handling missing values**
     + [isnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
     + [notnull()](https://www.geeksforgeeks.org/python-pandas-isnull-and-notnull/)
     + [dropna()](https://www.geeksforgeeks.org/python-pandas-dataframe-dropna/)
     + [fillna()](https://www.geeksforgeeks.org/python-pandas-dataframe-fillna-to-replace-null-values-in-dataframe/)
     + [replace()](https://www.geeksforgeeks.org/python-pandas-dataframe-replace/)
     + [interpolate()](https://www.geeksforgeeks.org/python-pandas-dataframe-interpolate/)

# importing pandas as pd import pandas as pd

# importing numpy as np import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list df = pd.DataFrame(dict)

# using isnull() function df.isnull()

## # importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv")

# creating bool series True for NaN values bool\_series = pd.isnull(data["Gender"])

# filtering data

# displaying data only with Gender = NaN data[bool\_series]

# importing pandas as pd import pandas as pd

# importing numpy as np import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe using dictionary df = pd.DataFrame(dict)

# using notnull() function df.notnull()

# importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv")

# creating bool series True for NaN values bool\_series = pd.notnull(data["Gender"])

# filtering data

# displaying data only with Gender = Not NaN data[bool\_series]

# importing pandas as pd import pandas as pd

# importing numpy as np import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary df = pd.DataFrame(dict)

# filling missing value using fillna() df.fillna(0)

# importing pandas as pd import pandas as pd

# importing numpy as np import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary df = pd.DataFrame(dict)

# filling a missing value with # previous ones df.fillna(method ='pad')

# importing pandas as pd import pandas as pd

# importing numpy as np import numpy as np

# dictionary of lists

dict = {'First Score':[100, 90, np.nan, 95], 'Second Score': [30, 45, 56, np.nan],

'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from dictionary df = pd.DataFrame(dict)

# filling null value using fillna() function df.fillna(method ='bfill')

Dimensionality Reduction

1. Implementation of PCA import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler #import the breast \_cancer dataset

from sklearn.datasets import load\_cancer data=load \_cancer()

data.keys()

# Check the output classes print(data['target\_names'])

# Check the input attributes print(data['feature\_names'])

# construct a dataframe using pandas df1=pd.DataFrame(data['data'],columns=data['feature\_names'])

# Scale data before applying PCA scaling=StandardScaler()

# Use fit and transform method scaling.fit(df1) Scaled\_data=scaling.transform(df1) # Set the n\_components=3 principal=PCA(n\_components=3) principal.fit(Scaled\_data) x=principal.transform(Scaled\_data)

# Check the dimensions of data after PCA print(x.shape)

# Check the values of eigen vectors # prodeced by principal components principal.components\_ plt.figure(figsize=(10,10))

plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma') plt.xlabel('pc1')

plt.ylabel('pc2')

# import relevant libraries for 3d graph from mpl\_toolkits.mplot3d import Axes3D fig = plt.figure(figsize=(10,10))

# choose projection 3d for creating a 3d graph axis = fig.add\_subplot(111, projection='3d')

# x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3 axis.scatter(x[:,0],x[:,1],x[:,2], c=data['target'],cmap='plasma') axis.set\_xlabel("PC1", fontsize=10)

axis.set\_ylabel("PC2", fontsize=10) axis.set\_zlabel("PC3", fontsize=10)

Write a python program to demonstrate various data visualisation

# importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv") # Printing the first 10 to 24 rows of # the data frame for visualization data[10:25]

# importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv")

# Printing the first 10 to 24 rows of # the data frame for visualization data[10:25]

# importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv")

# will replace Nan value in dataframe with value -99 data.replace(to\_replace = np.nan, value = -99)

# importing pandas as pd import pandas as pd

# Creating the dataframe

df = pd.DataFrame({"A":[12, 4, 5, None, 1],

"B":[None, 2, 54, 3, None],

"C":[20, 16, None, 3, 8],

"D":[14, 3, None, None, 6]})

# Print the dataframe Df

# importing the required module import matplotlib.pyplot as plt

# x axis values x = [1,2,3]

# corresponding y axis values y = [2,4,1]

# plotting the points plt.plot(x, y)

# naming the x axis plt.xlabel('x - axis') # naming the y axis plt.ylabel('y - axis')

# giving a title to my graph plt.title('My first graph!')

# function to show the plot plt.show()

return probabilities

def predict(info, test):

probabilities = calculateClassProbabilities(info, test) bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items(): if bestLabel is None or probability > bestProb:

bestProb = probability bestLabel = classValue

return bestLabel

def getPredictions(info, test): predictions = []

for i in range(len(test)):

result = predict(info, test[i]) predictions.append(result)

return predictions

def accuracy\_rate(test, predictions): correct = 0

for i in range(len(test)):

if test[i][-1] == predictions[i]: correct += 1

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

filename = r'E:\user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv' mydata = csv.reader(open(filename, "rt"))

mydata = list(mydata)

mydata = encode\_class(mydata) for i in range(len(mydata)):

mydata[i] = [float(x) for x in mydata[i]] ratio = 0.7

train\_data, test\_data = splitting(mydata, ratio) print('Total number of examples are: ', len(mydata))

print('Out of these, training examples are: ', len(train\_data)) print("Test examples are: ", len(test\_data))

info = MeanAndStdDevForClass(train\_data) predictions = getPredictions(info, test\_data) accuracy = accuracy\_rate(test\_data, predictions) print("Accuracy of your model is: ", accuracy)

1. Implementation of SVM Classification

# importing scikit learn with make\_blobs

from sklearn.datasets.samples\_generator import make\_blobs # creating datasets X containing n\_samples

# Y containing two classes

X, Y = make\_blobs(n\_samples=500, centers=2,random\_state=0, cluster\_std=0.40) import matplotlib.pyplot as plt

# plotting scatters

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring'); plt.show()

# creating linspace between -1 to 3.5 xfit = np.linspace(-1, 3.5)

# plotting scatter

plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring') # plot a line between the different sets of data

for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:

yfit = m \* xfit + b plt.plot(xfit, yfit, '-k')

plt.fill\_between(xfit, yfit - d, yfit + d, edgecolor='none', color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5); plt.show()

# importing required libraries

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

x = pd.read\_csv("C:\...\cancer.csv") a = np.array(x)

y = a[:,30] # classes having 0 and 1

x = np.column\_stack((x.malignant,x.benign)) x.shape

print (x),(y)

Supervised Learning

**1. Implementation of Linear Regression**

import numpy as np

import matplotlib.pyplot as plt def estimate\_coef(x, y):

n = np.size(x) m\_x = np.mean(x) m\_y = np.mean(y)

SS\_xy = np.sum(y\*x) - n\*m\_y\*m\_x SS\_xx = np.sum(x\*x) - n\*m\_x\*m\_x b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1\*m\_x return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot plt.scatter(x, y, color = "m",marker = "o", s = 30) y\_pred = b[0] + b[1]\*x

plt.plot(x, y\_pred, color = "g") plt.xlabel('x')

plt.ylabel('y') plt.show()

def main():

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

b = estimate\_coef(x, y)

print("Estimated coefficients:\nb\_0 = {}\\nb\_1 = {}".format(b[0], b[1])) plot\_regression\_line(x, y, b)

if name == " main ":

## Implementation of Logistic regression

import numpy as np import pandas as pd

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

warnings.filterwarnings( "ignore" ) class LogitRegression() :

def init ( self, learning\_rate, iterations ) : self.learning\_rate = learning\_rate self.iterations = iterations

def fit( self, X, Y ) : self.m, self.n = X.shape

self.W = np.zeros( self.n ) self.b = 0

self.X = X self.Y = Y

for i in range( self.iterations ) : self.update\_weights()

return self

def update\_weights( self ) :

A = 1 / ( 1 + np.exp( - ( self.X.dot( self.W ) + self.b ) ) ) tmp = ( A - self.Y.T )

tmp = np.reshape( tmp, self.m )

dW = np.dot( self.X.T, tmp ) / self.m db = np.sum( tmp ) / self.m

self.W = self.W - self.learning\_rate \* dW self.b = self.b - self.learning\_rate \* db return self

def predict( self, X ) :

Z = 1 / ( 1 + np.exp( - ( X.dot( self.W ) + self.b ) ) )

Y = np.where( Z > 0.5, 1, 0 )

return Y def main() :

df = pd.read\_csv( "diabetes.csv" ) X = df.iloc[:,:-1].values

Y = df.iloc[:,-1:].values

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split( X, Y, test\_size = 1/3, random\_state = 0 )

model = LogitRegression( learning\_rate = 0.01, iterations = 1000 ) model.fit( X\_train, Y\_train )

model1 = LogisticRegression() model1.fit( X\_train, Y\_train) Y\_pred = model.predict( X\_test ) Y\_pred1 = model1.predict( X\_test ) correctly\_classified = 0

correctly\_classified1 = 0

count = 0

for count in range( np.size( Y\_pred ) ) : if Y\_test[count] == Y\_pred[count] :

correctly\_classified = correctly\_classified + 1 if Y\_test[count] == Y\_pred1[count] :

correctly\_classified1 = correctly\_classified1 + 1 count = count + 1

print( "Accuracy on test set by our model : ", ( correctly\_classified / count ) \* 100 )

print( "Accuracy on test set by sklearn model : ", ( correctly\_classified1 / count ) \* 100 )

if name == " main " : main()

# importing pandas package import pandas as pd

# making data frame from csv file data = pd.read\_csv("employees.csv") # Printing the first 10 to 24 rows of # the data frame for visualization data[10:25]

Supervised Learning

1. Implementation of Decision tree classification import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import train\_test\_split from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report def importdata():

balance\_data = pd.read\_csv(['https://archive.ics.uci.edu/ml/machine-learning-](https://archive.ics.uci.edu/ml/machine-learning-) '+'databases/balance-scale/balance-scale.data',sep= ',', header = None)

print ("Dataset Length: ", len(balance\_data)) print ("Dataset Shape: ", balance\_data.shape) print ("Dataset: ",balance\_data.head())

return balance\_data

def splitdataset(balance\_data):

X = balance\_data.values[:, 1:5] Y = balance\_data.values[:, 0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, Y, test\_size = 0.3, random\_state = 100)

return X, Y, X\_train, X\_test, y\_train, y\_test def train\_using\_gini(X\_train, X\_test, y\_train):

clf\_gini = DecisionTreeClassifier(criterion = "gini",random\_state = 100,max\_depth=3, min\_samples\_leaf=5)

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

def tarin\_using\_entropy(X\_train, X\_test, y\_train):

clf\_entropy = DecisionTreeClassifier(criterion = "entropy", random\_state = 100,max\_depth = 3, min\_samples\_leaf = 5)

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

return clf\_entropy

def prediction(X\_test, clf\_object): y\_pred = clf\_object.predict(X\_test) print("Predicted values:") print(y\_pred)

return y\_pred

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",confusion\_matrix(y\_test, y\_pred))print ("Accuracy : ",accuracy\_score(y\_test,y\_pred)\*100)

print("Report : ", classification\_report(y\_test, y\_pred))

def main():

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data) clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train) clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train) print("Results Using Gini Index:")

y\_pred\_gini = prediction(X\_test, clf\_gini) cal\_accuracy(y\_test, y\_pred\_gini) print("Results Using Entropy:")

y\_pred\_entropy = prediction(X\_test, clf\_entropy) cal\_accuracy(y\_test, y\_pred\_entropy)

if name ==" main ": main()

1. Implementation of K-nearest Neighbor

from sklearn.neighbors import KNeighborsClassifier from sklearn.model\_selection import train\_test\_split from sklearn.datasets import load\_iris

import numpy as np

import matplotlib.pyplot as plt y = irisData.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=42)neighbors

= np.arange(1, 9)

train\_accuracy = np.empty(len(neighbors)) test\_accuracy = np.empty(len(neighbors)) for i, k in enumerate(neighbors):

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

train\_accuracy[i] = knn.score(X\_train, y\_train) test\_accuracy[i] = knn.score(X\_test, y\_test)

plt.plot(neighbors, test\_accuracy, label = 'Testing dataset Accuracy') plt.plot(neighbors, train\_accuracy, label = 'Training dataset Accuracy') plt.legend()

plt.xlabel('n\_neighbors') plt.ylabel('Accuracy') plt.show()

**Implementation of Naïve Bayes classifier algorithm**

import math import random import csv

def encode\_class(mydata): classes = []

for i in range(len(mydata)):

if mydata[i][-1] not in classes: classes.append(mydata[i][-1])

for i in range(len(classes)): for j in range(len(mydata)):

if mydata[j][-1] == classes[i]: mydata[j][-1] = i

return mydata

def splitting(mydata, ratio):

train\_num = int(len(mydata) \* ratio) train = []

test = list(mydata)

while len(train) < train\_num:

index = random.randrange(len(test)) train.append(test.pop(index))

return train, test

def groupUnderClass(mydata): dict = {}

for i in range(len(mydata)):

if (mydata[i][-1] not in dict): dict[mydata[i][-1]] = []

dict[mydata[i][-1]].append(mydata[i]) return dict

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

def std\_dev(numbers): avg = mean(numbers)

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

def MeanAndStdDev(mydata):

info = [(mean(attribute), std\_dev(attribute)) for attribute in zip(\*mydata)] del info[-1]

return info

def MeanAndStdDevForClass(mydata): info = {}

dict = groupUnderClass(mydata)

for classValue, instances in dict.items(): info[classValue] = MeanAndStdDev(instances)

return info

def calculateGaussianProbability(x, mean, stdev):

expo = math.exp(-(math.pow(x - mean, 2) / (2 \* math.pow(stdev, 2)))) return (1 / (math.sqrt(2 \* math.pi) \* stdev)) \* expo

def calculateClassProbabilities(info, test): probabilities = {}

for classValue, classSummaries in info.items(): probabilities[classValue] = 1

for i in range(len(classSummaries)): mean, std\_dev = classSummaries[i] x = test[i]

probabilities[classValue] \*= calculateGaussianProbability(x, mean, std\_dev)

**Implementation of K-nearest Neighbor** from sklearn.neighbors import KNeighborsClassifier from sklearn.model\_selection import train\_test\_split from sklearn.datasets import load\_iris

import numpy as np

import matplotlib.pyplot as plt

y = irisData.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state=42) neighbors = np.arange(1, 9)

train\_accuracy = np.empty(len(neighbors)) test\_accuracy = np.empty(len(neighbors)) for i, k in enumerate(neighbors):

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

train\_accuracy[i] = knn.score(X\_train, y\_train) test\_accuracy[i] = knn.score(X\_test, y\_test)

plt.plot(neighbors, test\_accuracy, label = 'Testing dataset Accuracy') plt.plot(neighbors, train\_accuracy, label = 'Training dataset Accuracy') plt.legend()

plt.xlabel('n\_neighbors') plt.ylabel('Accuracy') plt.show()

## Build Artificial Neural Network model with back propagation

Let’s first understand the term neural networks. In a neural network, where neurons are fed inputs which then neurons consider the weighted sum over them and pass it by an activation function and passes out the output to next neuron.

Python: To run our script

Pip: Necessary to install Python packages pip install tensorflow

pip install keras

# Importing libraries

from keras.datasets import imdb from keras.models import Sequential from keras.layers import Dense

from keras.layers import Flatten

from keras.layers.convolutional import Conv1D

from keras.layers.convolutional import MaxPooling1D from keras.layers.embeddings import Embedding

from keras.preprocessing import sequence# Our dictionary will contain only of the top 7000 words appearing most frequently

top\_words = 7000# Now we split our data-set into training and test data (X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=top\_words)# Looking at the nature of training data

print(X\_train[0])

print(y\_train[0])print('Shape of training data: ') print(X\_train.shape) print(y\_train.shape)print('Shape of test data: ') print(X\_test.shape)

print(y\_test.shape)

Output :

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36,

256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172,

112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192,

50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16,

43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62,

386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12,

16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28,

77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766,

5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,

381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134,

476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65,

16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19,

178, 32]

1

Shape of training data:

(25000,)

(25000,)

Shape of test data:

(25000,)

(25000,)

# Padding the data samples to a maximum review length in words max\_words = 450X\_train = sequence.pad\_sequences(X\_train, maxlen=max\_words)

X\_test = sequence.pad\_sequences(X\_test, maxlen=max\_words)# Building the CNN Model

model = Sequential() # initilaizing the Sequential nature for CNN model# Adding the embedding layer which will take in maximum of 450

words as input and provide a 32 dimensional output of those words which

belong in the top\_words dictionary model.add(Embedding(top\_words, 32, input\_length=max\_words)) model.add(Conv1D(32, 3, padding='same', activation='relu')) model.add(MaxPooling1D())

model.add(Flatten()) model.add(Dense(250, activation='relu')) model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()

Implementing Random Forest # Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

data = pd.read\_csv('Salaries.csv') print(data)

# Fitting Random Forest Regression to the dataset # import the regressor

from sklearn.ensemble import RandomForestRegressor # create regressor object

regressor = RandomForestRegressor(n\_estimators = 100, random\_state = 0) # fit the regressor with x and y data

regressor.fit(x, y)

Y\_pred = regressor.predict(np.array([6.5]).reshape(1, 1)) # test the output by changing values # Visualising the Random Forest Regression results

# arrange for creating a range of values # from min value of x to max

# value of x with a difference of 0.01 # between two consecutive values

X\_grid = np.arrange(min(x), max(x), 0.01)

# reshape for reshaping the data into a len(X\_grid)\*1 array, # i.e. to make a column out of the X\_grid value

X\_grid = X\_grid.reshape((len(X\_grid), 1)) # Scatter plot for original data plt.scatter(x, y, color = 'blue')

# plot predicted data

plt.plot(X\_grid, regressor.predict(X\_grid),color = 'green') plt.title('Random Forest Regression')

plt.xlabel('Position level') plt.ylabel('Salary')

## WEEK-11(B) : Model Selection, Bagging and Boosting

* 1. **Cross Validation**

# This code may not be run on GFG IDE # as required packages are not found.

# importing cross-validation from sklearn package.from sklearn import cross\_validation # value of K is 10.

data = cross\_validation.KFold(len(train\_set), n\_folds=10, indices=False)

## Implementing AdaBoost

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import AdaBoostClassifier import warnings

warnings.filterwarnings("ignore")

# Reading the dataset from the csv file

# separator is a vertical line, as seen in the dataset data = pd.read\_csv("Iris.csv")

# Printing the shape of the dataset print(data.shape)

data = data.drop('Id',axis=1) X = data.iloc[:,:-1]

y = data.iloc[:,-1]

print("Shape of X is %s and shape of y is %s"%(X.shape,y.shape)) total\_classes = y.nunique()

print("Number of unique species in dataset are: ",total\_classes) distribution = y.value\_counts()

print(distribution)

X\_train,X\_val,Y\_train,Y\_val = train\_test\_split(X,y,test\_size=0.25,random\_state=28)

print("The accuracy of the model on validation set is", adb\_model.score(X\_val,Y\_val))

**Unsupervised Learning**

Implementing K-means Clustering def ReadData(fileName):

# Read the file, splitting by lines f = open(fileName, 'r');

lines = f.read().splitlines(); f.close();

items = [];

for i in range(1, len(lines)): line = lines[i].split(','); itemFeatures = [];

for j in range(len(line)-1):

# Convert feature value to float v = float(line[j]);

# Add feature value to dict itemFeatures.append(v);

items.append(itemFeatures); shuffle(items);

return items;

def FindColMinMax(items):n

= len(items[0]);

minima = [sys.maxint for i in range(n)]; maxima = [-sys.maxint -1 for i in range(n)]; for item in items:

for f in range(len(item)): if (item[f] < minima[f]):

minima[f] = item[f]; if (item[f] > maxima[f]):

maxima[f] = item[f]; return minima,maxima;

def InitializeMeans(items, k, cMin, cMax):

# Initialize means to random numbers between # the min and max of each column/feature

f = len(items[0]); # number of features

means = [[0 for i in range(f)] for j in range(k)]; for mean in means:

for i in range(len(mean)):

# Set value to a random float

# (adding +-1 to avoid a wide placement of a mean) mean[i] = uniform(cMin[i]+1, cMax[i]-1);

return means;

def EuclideanDistance(x, y):

S = 0; # The sum of the squared differences of the elements for i in range(len(x)):

S += math.pow(x[i]-y[i], 2) #The square root of the sum return math.sqrt(S)

def UpdateMean(n,mean,item): for i in range(len(mean)):

m = mean[i];

m = (m\*(n-1)+item[i])/float(n); mean[i] = round(m, 3);

return mean;

def Classify(means,item):

# Classify item to the mean with minimum distance minimum = sys.maxint;

index = -1;

for i in range(len(means)):

# Find distance from item to mean

dis = EuclideanDistance(item, means[i]); if (dis < minimum):

minimum = dis; index = i;

return index;

def CalculateMeans(k,items,maxIterations=100000): # Find the minima and maxima for columns cMin, cMax = FindColMinMax(items);

# Initialize means at random points

means = InitializeMeans(items,k,cMin,cMax); # Initialize clusters, the array to hold

# the number of items in a class clusterSizes= [0 for i in range(len(means))]; # An array to hold the cluster an item is in belongsTo = [0 for i in range(len(items))]; # Calculate means

for e in range(maxIterations):

# If no change of cluster occurs, halt noChange = True;

for i in range(len(items)): item = items[i];

# Classify item into a cluster and update the # corresponding means.

index = Classify(means,item); clusterSizes[index] += 1; cSize = clusterSizes[index];

means[index] = UpdateMean(cSize,means[index],item); # Item changed cluster

if(index != belongsTo[i]): noChange = False;

belongsTo[i] = index;

# Nothing changed, return if (noChange):

break; return means;

def FindClusters(means,items):

clusters = [[] for i in range(len(means))]; # Init clusters for item in items:

# Classify item into a cluster index = Classify(means,item); # Add item to cluster clusters[index].append(item);

return clusters;