MACHINE LEARNING

LABORATORY MANUAL

[R20A0590]

B.TECH III YEAR – II

SEM [A.Y:2022-2023]

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC
ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

Department of Computer Science and Engineering

Vision

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students in to competent and confident engineers.

- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes,and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. Fundamentals and critical knowledge of the Computer System:- Able toUnderstand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. Applications of Computing Domain & Research: Able to use the professional,managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineeringfundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyzecomplex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions:** Design solutions for complex engineeringproblems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge andresearch methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, andmodern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge toassess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professionalengineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilitiesand norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member orleader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities withthe engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding ofthe engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning:** Recognize the need for, and have the preparation and ability toengage in independent and life-long learning in the broadest context of technological change.

1. Lab Objectives:

- Learning basic concepts of Python through illustrative examples and small exercises
- To prepare students to become Familiarity with the Python programming in AI environment.
- To provide student with an academic environment aware of various AI Algorithms.
- To train Students with python programming as to comprehend, analyze, design and create AI platforms and solutions for the real life problems.

2. Lab Outcomes:

Upon completion of the course, students will be able to

- Apply various AI search algorithms (uninformed, informed, heuristic, constraint satisfaction,)
- Understand the fundamentals of knowledge representation, inference.
- Understand the fundamentals of theorem proving using AI tools.
- Demonstrate working knowledge of reasoning in the presence of incomplete and/or uncertain information

3. Introduction about lab

Minimum System requirements:

- Processors: Intel Atom® processor or Intel® Core™ i3 processor.
- Disk space: 1 GB.
- Operating systems: Windows* 7 or later, macOS, and Linux.
- **Python*** versions: 2.7.X, 3.6.X.,3.8.X

About lab:

Python is a general purpose, high-level programming language; other high level languages you might have heard of C++, PHP, Java and Python. Virtually all modern programming languages make us of an Integrated Development Environment (IDE), which allows the creation, editing, testing, and saving of programs and modules. In Python, the IDE is called IDLE (like many items in the language, this is a reference to the British comedy group Monty Python,and in this case, one of its members, Eric Idle).

Many modern languages use both processes. They are first compiled into a lower level language, called byte code, and then interpreted by a program called a virtual machine. Python uses both processes, but because of the way

programmers interact with it, it is usually considered an interpreted language.

Practical aspects are the key to understanding and conceptual visualization of Theoretical aspects covered in the books. Also, this course is designed to review the concepts of Data Structure , studied in previous semester and implement the various algorithms related to different data structures.

4. Guidelines to students

A. Standard operating procedure

a) Explanation on today's experiment by the concerned faculty using PPT covering the following aspects:

1) Name of the experiment
2) Aim

b) Writing the python programs by the students

c) Commands for executing programs

Writing of the experiment in the Observation Book

The students will write the today's experiment in the Observation book as per the following format:

a) Name of the experiment
b) Aim
c) Writing the program
d) Viva-Voce Questions and Answers
e) Errors observed (if any) during compilation/execution
f) Signature of the Faculty

B. Guide Lines to Students in Lab

Disciplinary to be maintained by the students in the Lab

- Students are required to carry their lab observation book and record book with completed experiments while entering the lab.
- Students must use the equipment with care. Any damage is caused student is punishable
- Students are not allowed to use their cell phones/pen drives/ CDs in labs.
- Students need to be maintain proper dress code along with ID Card
- Students are supposed to occupy the computers allotted to them and are not supposed to talk or make noise in the lab. Students, after completion of each experiment they need to be updated in observation notes and same to be updated in the record.
- Lab records need to be submitted after completion of experiment and get it corrected with the concerned lab faculty.
- If a student is absent for any lab, they need to be completed the same experiment in the free time before attending next lab.

Instructions to maintain the record

- Before start of the first lab they have to buy the record and bring the record to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.
- In case the record is lost inform the same day to the faculty in charge and get the new record within 2 days the record has to be submitted and get it corrected by the faculty.
- If record is not submitted in time or record is not written properly, the evaluation marks (5M) will be deducted.

C. General laboratory instructions

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab. c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

INDEX

Week1: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
```

2. Numpy.loadtxt function

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

3. Numpy.genfromtxt()

```
data = np.genfromtxt('100 Sales Records.csv', delimiter=',')
>>> pd.DataFrame(data)
```

4. Pandas.read_csv()

```
>>> pdDf = pd.read_csv('100 Sales Record.csv')
>>> pdDf.head()
```

5. Pickle

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

WEEK-2: Data preprocessing

1. **Handling missing values**
   - isnull()
   - notnull()
   - dropna()
   - fillna()
   - replace()
   - 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')
```

WEEK-3: 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_breast_cancer
data=load_breast_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)
```

WEEK-4: 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)
```

WEEK-5: 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__":
```

WEEK-6 : 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]
```

WEEK-7: 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-
'+'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()
```

WEEK-8

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)
```

Week-9: 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()
```

WEEK-10: 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()
```

WEEK-11

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)
```

2. 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))
```

WEEK-12: 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;
```