Malla Reddy College of Engineering

Maisammaguda,Dhulapally(V),Medchal(M),
Hyderabad -500100, Telangana.

COURSE LABORATORY MANUAL

R18 B.TECH CSE III YEAR

CS604PC: MACHINE LEARNING LAB
III Year B.Tech. CSE II-Sem **L T P C**
0 0 3 1.5

Course Objective: The objective of this lab is to get an overview of the various machine learning
techniques and can able to demonstrate them using python.

Course Outcomes: After the completion of the course the student can able to:
1. understand complexity of Machine Learning algorithms and their limitations;
2. understand modern notions in data analysis-oriented computing;
3. be capable of confidently applying common Machine Learning algorithms in practice and implementing their own;
4. Be capable of performing experiments in Machine Learning using real-world data.

List of Experiments :
1. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is theprobability that a student is
absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

2. Extract the data from database using python

3. Implement k-nearest neighbours classification using python

4. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2
predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans
clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

5. The following training examples map descriptions of individuals onto high, medium and low credit-worthiness.
medium skiing design single twenties no ->highRisk
high golf trading married forties yes ->lowRisk
low speedway transport married thirties yes ->medRisk
medium football banking single thirties yes ->lowRisk

high flying media married fifties yes ->highRisk
low football security single twenties no ->medRisk
medium golf media single thirties yes ->medRisk
medium golf transport married forties yes ->lowRisk
high skiing banking single thirties yes ->highRisk
low golf unemployed married forties yes ->highRisk

Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the
unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the
dataset?

6. Implement linear regression using python.

7. Implement Naïve Bayes theorem to classify the English text

8. Implement an algorithm to demonstrate the significance of genetic algorithm

9. Implement the finite words classification system using Back-propagation algorithm

EXPERIMENT NO:1

The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

Aim:

To Find the the probability that a student is absent given that today is Friday from given data with Baye's rule in python.

Theory:

P (Today is Friday)=0.2
P(B)=0.2

(It is Friday∩student is absent)= P(A∩B)=0.03
P(A | B) = P(A and B) / P(B)

it is required to find
P(student is absent |today is Friday)   P(A|B)
The formula for obtaining the conditional probability of event A, given the event B has occurred is as follows:

P(A|B)=P(A∩B)/P(B)
Thus the required probability is as follows:

P(student is absent| today is Friday)=P(It is Friday∩student is absent)/P(Today is Friday)
=0.03/0.2=0.15

The answer is 0.15

PROCEDURE / PROGRAMME

```
# calculate P(A|B) given P(A and B)  and P(B)

def bayes_theorem(p_a_b, p_b):

        # calculate P(A|B) = P(A and B) / P(B)

        p_a_given_b = (p_a_b) / p_b

        return p_a_given_b

 # P(A and B)

p_a_b = 0.03

# P(B)

p_b = 0.20

# calculate P(A|B)

result = bayes_theorem(p_a_b, p_b)

# summarize

print('P(A|B) = %.f%%' % (result * 100))
```

EXPERIMENT NO:2

Extract the data from database using python

Aim:

To extract the data from database using python

Theory:

1. **Connect to MySQL from Python**

   Refer to Python MySQL database connection to connect to MySQL database from Python using MySQL Connector module

2. **Define a SQL SELECT Query**

   Next, prepare a SQL SELECT query to fetch rows from a table. You can select all or limited rows based on your requirement. If the where condition is used, then it decides the number of rows to fetch.
   For example, SELECT col1, col2,…colnN FROM MySQL_table WHERE id = 10;. This will return row number 10.

3. **Get Cursor Object from Connection**

   Next, use a connection.cursor() method to create a cursor object. This method creates a new MySQLCursor object.

4. **Execute the SELECT query using execute() method**

   Execute the select query using the cursor.execute() method.

5. **Extract all rows from a result**

   After successfully executing a Select operation, Use the fetchall() method of a cursor object to get all rows from a query result. it returns a list of rows.

6. **Iterate each row**

   Iterate a row list using a for loop and access each row individually (Access each row's column data using a column name or index number.)

7. **Close the cursor object and database connection object**

   use cursor.clsoe() and connection.clsoe() method to close open connections after your work completes.

PROCEDURE / PROGRAMME

```
import mysql.connector

from mysql.connector import Error

try:

connection=mysql.connector.connect(host='localhost',database='employeeDB',charset='
utf8',user='root',password='root')

    print("connected")

    sql_select_Query = "SELECT * FROM employee"

    cursor = connection.cursor()

    cursor.execute(sql_select_Query)

    records = cursor.fetchall()

    print("Total number of rows in employee is: ", cursor.rowcount)

    print("\nPrinting each employee record")

    for row in records:

        print("Id = ", row[0],"\n" )

        print("Name = ", row[1], "\n")

        print("Address  = ", row[2])

        print("Join date  = ", row[3], "\n")

except Error as e:

    print("Error reading data from MySQL table", e)

    connection.close()

    cursor.close()

    print("MySQL connection is closed")
```

For Insert the value Python program

```
import mysql.connector

from mysql.connector import Error

try:

    mydb                                                                        =
mysql.connector.connect(host='localhost',database='employeeDB',charset='utf8',user='r
oot',password='root')

    mycursor = mydb.cursor()

    sql  = "INSERT INTO employee(id,Name,empaddress,edoj)VALUES (%s,%s,%s,%s)"

    val = [
     (2111,'rubesh','Lowstreet 4','2019-09-12'),
    (2121,'siva','Apple st 652','2019-09-12'),
      ]
    mycursor.executemany(sql, val)
    mydb.commit()
    print(mycursor.rowcount, "was inserted.")
except Error as e:
    print("Error reading data from MySQL table", e)
finally:
     if mydb.is_connected():
      mydb.close()
     #cursor.close()
      print("MySQL connection is closed")
```

Output

```
connected
Total number of rows in employee is:  4
Printing each employee record
Id =  111
Name =  siva
Address  =  madurai
Join date  =  2015-12-17
Id =  112
Name =  Ram
Address  =  Theni
Join date  =  2016-12-18
Id =  2111
Name =  rubesh
Address  =  Lowstreet 4
Join date  =  2019-09-12
Id =  2121
Name =  siva
Address  =  Apple st 652
Join date  =  2019-09-12
```

EXPERIMENT NO:3

Implement k-nearest neighbours classification using python

Aim:

To implement k-nearest neighbours classification using python

Theory:

• K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.
• It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data.
• Algorithm

```
Input: Let m be the number of training data samples. Let p be an unknown point.
Method:
1. Store the training samples in an array of data points arr[]. This means each element
of this array represents a tuple (x, y).
2. for i=0 to m
Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to
an already classified data point.
4. Return the majority label among S.
```

PROCEDURE / PROGRAMME :

```
# import the required packages
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))
```

Output

Result-1

```
Iris Data set loaded...
Dataset is split into training and testing samples...
Size of trainng data and its label (135, 4) (135,)
Size of trainng data and its label (15, 4) (15,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
Sample: [4.4 3. 1.3 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.1 2.5 3. 1.1] Actual-label: 1 Predicted-label: 1
Sample: [6.1 2.8 4. 1.3] Actual-label: 1 Predicted-label: 1
Sample: [6. 2.7 5.1 1.6] Actual-label: 1 Predicted-label: 2
Sample: [6.7 2.5 5.8 1.8] Actual-label: 2 Predicted-label: 2
Sample: [5.1 3.8 1.5 0.3] Actual-label: 0 Predicted-label: 0
Sample: [6.7 3.1 4.4 1.4] Actual-label: 1 Predicted-label: 1
Sample: [4.8 3.4 1.6 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.1 3.5 1.4 0.3] Actual-label: 0 Predicted-label: 0
Sample: [5.4 3.7 1.5 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.7 2.8 4.1 1.3] Actual-label: 1 Predicted-label: 1
Sample: [4.5 2.3 1.3 0.3] Actual-label: 0 Predicted-label: 0
Sample: [4.4 2.9 1.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [5.1 3.5 1.4 0.2] Actual-label: 0 Predicted-label: 0
Sample: [6.2 3.4 5.4 2.3] Actual-label: 2 Predicted-label: 2

   Classification Accuracy : 0.93
```

EXPERIMENT NO:4

Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

Aim:
To predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of kmeans clustering with 3 means and given data.

Theory:
Step 1: Python 3 code snippet demonstrates the implementation of a simple K-Means clustering to automatically divide input data into groups based on given features.

Step 2: " , " separated CSV file is loaded first, which contains three corresponding input columns.

Step 3: K-Means clustering model is created from this input data. Afterwards, new data can be classified using the predict() method based on the learned model.

Step 4: The Scikit-learn and the Pandas library to be installed (pip install sklearn, pip install pandas).

Step 5

input_data.txt

```
VAR1,VAR2,cLASS

1.713,1.586,0

0.180,1.786,1

0.353,1.240,1

0.940,1.566,0

1.486,0.759,1

1.266,1.106,0

1.540,0.419,1

0.459,1.799,1

0.773,0.186,1
```

Step 6

PROCEDURE / PROGRAMME

```
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np
import pickle
# read csv input file
input_data = pd.read_csv("input_data.txt", sep=",")
print(input_data.to_string())
# initialize KMeans object specifying the number of desired clusters
kmeans = KMeans(n_clusters=3)
# learning the clustering from the input date
kmeans.fit(input_data.values)
# output the labels for the input data
print(kmeans.labels_)
# predict the classification for given data sample
predicted_class = kmeans.predict([[0.906,0.606,1]])
print(predicted_class)
```

Output

```
   VAR1   VAR2  cLASS
0  1.713  1.586     0
1  0.180  1.786     1
2  0.353  1.240     1
3  0.940  1.566     0
4  1.486  0.759     1
5  1.266  1.106     0
6  1.540  0.419     1
7  0.459  1.799     1
8  0.773  0.186     1
[1 0 0 1 2 1 2 0 2]
[2]
```

EXPERIMENT NO:5

The following training examples map descriptions of individuals onto high, medium and low credit-worthiness.

Aim:

To unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset

medium skiing design single twenties no ->highRisk
high golf trading married forties yes ->lowRisk
low speedway transport married thirties yes ->medRisk
medium football banking single thirties yes ->lowRisk
high flying media married fifties yes ->highRisk
low football security single twenties no ->medRisk
medium golf media single thirties yes ->medRisk
medium golf transport married forties yes ->lowRisk
high skiing banking single thirties yes ->highRisk
low golf unemployed married forties yes ->highRisk

Input attributes are (from left to right) income, recreation, job, status, age-group, home-owner. Find the unconditional probability of `golf' and the conditional probability of `single' given `medRisk' in the dataset?

Theory:

Calculations of parts:

P(A) = (2+1) / (4+2+ 3+1) =0 .3
P(B) = (3+1) / (4+2+ 3+1) = 0.4
P(A∩B) = (.1) / (4+2+ 3+1) =0 .1
And per the formula, P(A|B) = P(A ∩ B) / P(B), put it together.

P(A|B) =0.1 / 0.4= 0.25

unconditional probability of `golf' and given `medRisk' in the dataset is 25%.

PROCEDURE / PROGRAMME

```
import pandas as pd

import numpy as np

df = pd.read_csv('pd.csv')

df.head(10)

print(len(df))

print(df.to_string())

df['Arecreation'] = np.where(df['recreation']=='golf', 1, 0)

df['Arisk'] = np.where(df['risk']=='medRisk', 1, 0)
```

```
df['count'] = 1

df = df[['Arecreation','Arisk','count']]

df.head()

print(df.to_string())

table=pd.pivot_table(

    df,

    values='count',

    index=['Arecreation'],

    columns=['Arisk'],

    aggfunc=np.size,

    fill_value=0

   )

print(table)

a0=table.at[0,0]

a1=table.at[0,1]

a2=table.at[1,0]

a3=table.at[1,1]

pa=(a1+a3)/(a0+a1+a2+a3)

pb=(a2+a3)/(a0+a1+a2+a3)

p_a_and_b=(a3/(a0+a1+a2+a3))

p_a_gives_b=p_a_and_b/pb

print(p_a_gives_b)


print('P(A|B) = %.f%%' % (p_a_gives_b * 100))
```

output

```
10
   income recreation        job   status  age-group home-owner      risk
0  medium     skiing     design   single   twenties         no  highRisk
1    high       golf    trading  married    forties        yes   lowRisk
2     low   speedway  transport  married   thirties        yes   medRisk
3  medium   football    banking   single   thirties        yes   lowRisk
4    high     flying      media  married    fifties        yes  highRisk
5     low   football   security   single   twenties         no   medRisk
6  medium       golf      media   single   thirties        yes   medRisk
7  medium       golf  transport  married    forties        yes   lowRisk
8    high     skiing    banking   single   thirties        yes  highRisk
9     low       golf unemployed  married    forties        yes  highRisk
   Arecreation  Arisk  count
0            0      0      1
1            1      0      1
2            0      1      1
3            0      0      1
4            0      0      1
5            0      1      1
6            1      1      1
7            1      0      1
8            0      0      1
9            1      0      1
Arisk        0  1
Arecreation
0            4  2
1            3  1
0.25
P(A|B) = 25%
```

EXPERIMENT NO:6

Implement linear regression using python.

Aim:

To implement linear regression using python

Theory:

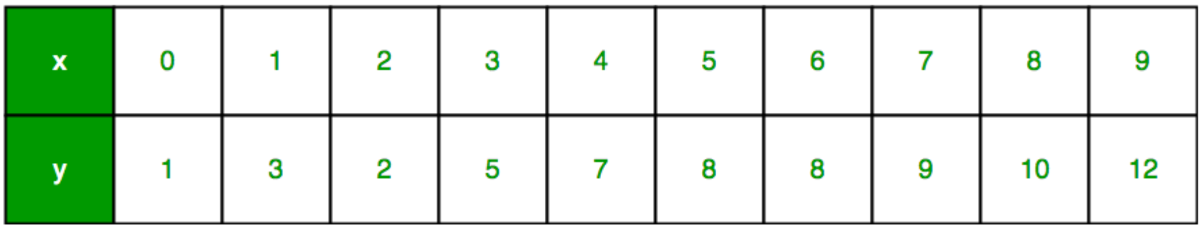
Linear Regression (Python Implementation)

Linear regression is a statistical method for modelling relationship between a dependent variable with a given set of independent variables.

In order to provide a basic understanding of linear regression, we start with the most basic version of linear regression, i.e. **Simple linear regression**.

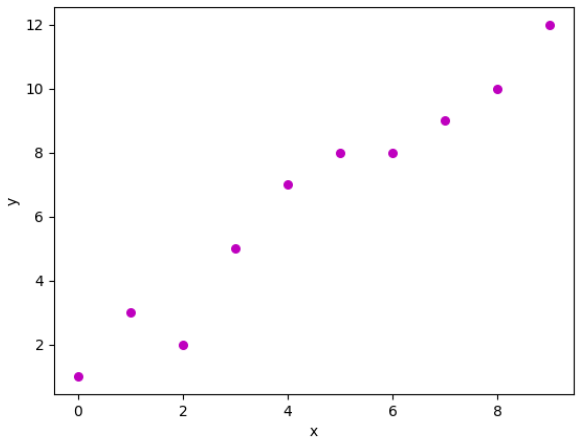
Simple Linear Regression

Simple linear regression is an approach for predicting a **response** using a **single feature**. It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).
Let us consider a dataset where we have a value of response y for every feature x:



x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

For generality, we define:
x as **feature vector**, i.e x = [x_1, x_2, …., x_n],
y as **response vector**, i.e y = [y_1, y_2, …., y_n]
for **n** observations (in above example, n=10).
A scatter plot of above dataset looks like:-



Now, the task is to find a **line which fits best** in above scatter plot so that we can predict the response for any new feature values. (i.e a value of x not present in dataset)
This line is called **regression line**.
The equation of regression line is represented as:

Here,

- h(x_i) represents the **predicted response value** for i^{th} observation.
- b_0 and b_1 are regression coefficients and represent **y-intercept** and **slope** of regression line respectively.

To create our model, we must "learn" or estimate the values of regression coefficients b_0 and b_1. And once we've estimated these coefficients, we can use the model to predict responses!
In this article, we are going to use the principle of **Least Squares** .
Now consider:

Here, e_i is **residual error** in ith observation.
So, our aim is to minimize the total residual error.
We define the squared error or cost function, J as:

and our task is to find the value of b_0 and b_1 for which J(b_0,b_1) is minimum!
Without going into the mathematical details, we present the result here:

where SS_xy is the sum of cross-deviations of y and x:

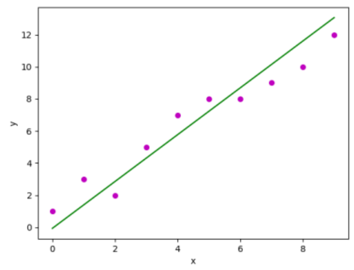
and SS_xx is the sum of squared deviations of x:

PROCEDURE / PROGRAMME

```
import numpy as np

import matplotlib.pyplot as plt

def estimate_coef(x, y):

        # number of observations/points

        n = np.size(x)

        # mean of x and y vector

        m_x = np.mean(x)

        m_y = np.mean(y)

        # calculating cross-deviation and deviation about x

        SS_xy = np.sum(y*x) - n*m_y*m_x

        SS_xx = np.sum(x*x) - n*m_x*m_x

        # calculating regression coefficients

        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)
    # predicted response vector
    y_pred = b[0] + b[1]*x
    # plotting the regression line
    plt.plot(x, y_pred, color = "g")
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()
def main():
    # observations / data
    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])
    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \
        \nb_1 = {}".format(b[0], b[1]))
    # plotting regression line
    plot_regression_line(x, y, b)
if __name__ == "__main__":
    main()
```

Output

```
Estimated coefficients:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



EXPERIMENT NO:7

Implement Naïve Bayes theorem to classify the English text

Aim:

To implement Naïve Bayes theorem to classify the English text

Theory:

<u>Naive Bayes algorithms for learning and classifying text</u>

LEARN_NAIVE_BAYES_TEXT (Examples, V)

Examples is a set of text documents along with their target values. V is the set of all possible target values. This function learns the probability terms $P(w_k | v_j,)$, describing the probability that a randomly drawn word from a document in class v_j will be the English word w_k. It also learns the class prior probabilities $P(v_j)$.

```
1. collect all words, punctuation, and other tokens that occur in Examples
    • Vocabulary ← c the set of all distinct words and other tokens occurring in any
      textdocument from Examples
2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms
    • For each target value $v_j$ in V do
        • $docs_j$ ← the subset of documents from Examples for which the target value
          is $v_j$
        • $P(v_j) ← | docs_j | / |Examples|$
        • $Text_j$ ← a single document created by concatenating all members of $docs_j$
        • n ← total number of distinct word positions in $Text_j$
        • for each word $w_k$ in Vocabulary
            • $n_k$ ← number of times word $w_k$ occurs in $Text_j$
            • $P(w_k|v_j) ← ( n_k + 1) / (n + | Vocabulary| )$
```

CLASSIFY_NAIVE_BAYES_TEXT (Doc)

Return the estimated target value for the document Doc. a_i denotes the word found in the i^{th} position within Doc.

```
• positions ← all word positions in Doc that contain tokens found in Vocabulary
• Return $V_{NB}$, where
```

Data set:

	Text Documents	Label
1	I love this sandwich	pos
2	This is an amazing place	pos
3	I feel very good about these beers	pos
4	This is my best work	pos
5	What an awesome view	pos
6	I do not like this restaurant	neg
7	I am tired of this stuff	neg
8	I can't deal with this	neg
9	He is my sworn enemy	neg
10	My boss is horrible	neg
11	This is an awesome place	pos
12	I do not like the taste of this juice	neg
13	I love to dance	pos
14	I am sick and tired of this place	neg
15	What a great holiday	pos
16	That is a bad locality to stay	neg
17	We will have good fun tomorrow	pos
18	I went to my enemy's house today	neg

PROCEDURE / PROGRAMME

```
import pandas as pd

msg=pd.read_csv('naivetext.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.message

y=msg.labelnum

print(X)

print(y)

#splitting the dataset into train and test data
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)
#output of count vectoriser is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
predicted = clf.predict(xtest_dtm)
#printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))
print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision' , metrics.precision_score(ytest,predicted))
print('\n The value of Recall' , metrics.recall_score(ytest,predicted))
```

Output

```
The dimensions of the dataset (18, 2)
0                 I love this sandwich
1              This is an amazing place
2         I feel very good about these beers
3                 This is my best work
4                 What an awesome view
5            I do not like this restaurant
6               I am tired of this stuff
```

```
7                  I can't deal with this
8                    He is my sworn enemy
9                     My boss is horrible
10                This is an awesome place
11    I do not like the taste of this juice
12                         I love to dance
13       I am sick and tired of this place
14                    What a great holiday
15           That is a bad locality to stay
16           We will have good fun tomorrow
17         I went to my enemy's house today
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64
 The total number of Training Data : (13,)
 The total number of Test Data : (5,)
 The words or Tokens in the text documents

['am', 'amazing', 'an', 'awesome', 'best', 'boss', 'can', 'dance', 'deal', 'do', 'enemy', 'great',
'he', 'holiday', 'horrible', 'is', 'juice', 'like', 'love', 'my', 'not', 'of', 'place', 'restaurant',
'sandwich', 'stuff', 'sworn', 'taste', 'the', 'this', 'tired', 'to', 'view', 'what', 'with', 'work']

 Accuracy of the classifer is 0.8

 Confusion matrix

[[2 1]

 [0 2]]

 The value of Precision 0.6666666666666666

 The value of Recall 1.0
```

Basic knowledge

Confusion Matrix

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True positives: data points labelled as positive that are actually positive **False positives:** data points labelled as positive that are actually negative **True negatives:** data points labelled as negative that are actually negative **False negatives:** data points labelled as negative that are actually positive

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Predicted Positive}}$$

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Example:

		Actual	
		Positive	Negative
Predicted	Positive	1 TP	3 FP
	Negative	0 FN	1 TN

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{1}{1+3} = 0.25$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{1}{1+0} = 1$$

Accuracy: how often is the classifier correct?

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{Total}} = \frac{1+1}{5} = 0.4$$

Example: Movie Review

Doc	Text	Class
1	I loved the movie	+
2	I hated the movie	-
3	a great movie. good movie	+
4	poor acting	-
5	great acting. good movie	+

Unique word

< I, loved, the, movie, hated, a, great, good, poor, acting>

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
2	1		1	1	1						-
3				2		1	1	1			+
4									1	1	-
5				1			1	1		1	+

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
1	1	1	1	1							+
3				2		1	1	1			+
5				1			1	1		1	+

$$P(+) = \frac{3}{5} = 0.6$$

$$P(I\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(loved\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(the\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(movie\ |+) = \frac{4+1}{14+10} = 0.2083$$

$$P(hated\ |+) = \frac{0+1}{14+10} = 0.0416$$

$$P(a\ |+) = \frac{1+1}{14+10} = 0.0833$$

$$P(great\ |+) = \frac{2+1}{14+10} = 0.125$$

$$P(good\ |+) = \frac{2+1}{14+10} = 0.125$$

$$P(poor\ |+) = \frac{0+1}{14+10} = 0.0416$$

$$P(acting\ |+) = \frac{1+1}{14+10} = 0.0833$$

Doc	I	loved	the	movie	hated	a	great	good	poor	acting	Class
2	1		1	1	1						-
4									1	1	-

$$P(-) = \frac{2}{5} = 0.4$$

$$P(I \mid -) = \frac{1+1}{6+10} = 0.125 \qquad P(a \mid -) = \frac{0+1}{6+10} = 0.0625$$

$$P(loved \mid -) = \frac{0+1}{6+10} = 0.0625 \qquad P(great \mid -) = \frac{0+1}{6+10} = 0.0625$$

$$P(the \mid -) = \frac{1+1}{6+10} = 0.125 \qquad P(good \mid -) = \frac{0+1}{6+10} = 0.0625$$

$$P(movie \mid -) = \frac{1+1}{6+10} = 0.125 \qquad P(poor \mid -) = \frac{1+1}{6+10} = 0.125$$

$$P(hated \mid -) = \frac{1+1}{6+10} = 0.125 \qquad P(acting \mid -) = \frac{1+1}{6+10} = 0.125$$

Let's classify the new document

I hated the poor acting

If $V_j = +$

then,

$= P(+)\ P(I \mid +)\ P(hated \mid +)\ P(the \mid +)\ P(poor \mid +)\ P(acting \mid +)$

$= 0.6 * 0.0833 * 0.0416 * 0.0833 * 0.0416 * 0.0833$

$= 6.03 \times 10^{-2}$

If $V_j = -$

then,

$= P(-)\ P(I \mid -)\ P(hated \mid -)\ P(the \mid -)\ P(poor \mid -)\ P(acting \mid -)$

$= 0.4 * 0.125 * 0.125 * 0.125 * 0.125 * 0.125$

$= 1.22 \times 10^{-5}$

$= 1.22 \times 10^{-5} > 6.03 \times 10^{-2}$

So, the new document belongs to $(-)$ class

EXPERIMENT NO:8

Implement an algorithm to demonstrate the significance of genetic algorithm

Aim:

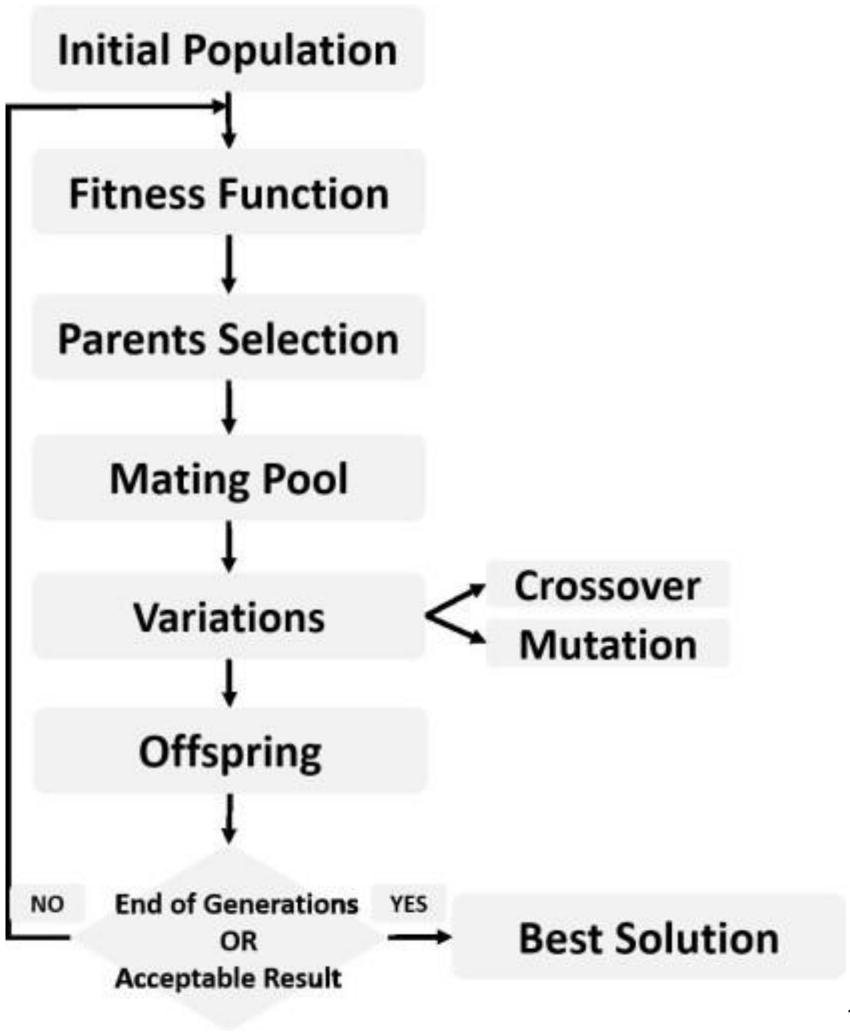
To implement an algorithm to demonstrate the significance of genetic algorithm

Theory:

Genetic algorithm

The genetic algorithm is a population-based evolutionary algorithm, where a group of solutions works together to find the optimal parameters for a problem. The below figure, from this book, summarizes all the steps in the genetic algorithm.

The population of solutions is initialized randomly, where each solution consists of a number of genes. The quality of solutions is assessed using a fitness function, which returns a numeric value representing how fit the solution is.



This figure is copyrighted material in this book and should not be used without permission.

The high-quality (high-fitness) solutions survive longer than the ones with low fitness. The higher the fitness, the higher probability of selecting the solution as a parent to produce new offspring. To produce the offspring, pairs of parents mate using the crossover operation, where a new solution is generated that carries genes from its parents.

After crossover, mutation is applied to add some random changes over the solution. The evolution continues through a number of generations to reach the highest-quality solution.

For more information about the genetic algorithm, read this article: Introduction to Optimization with Genetic Algorithm.

Even though the same steps are applied to all types of problems, you still need to select appropriate parameters to fit different problems. Some of these parameters include:

The number of solutions in the population,

Parent selection type,

Crossover operator type,

Mutation operator type,

Crossover probability,

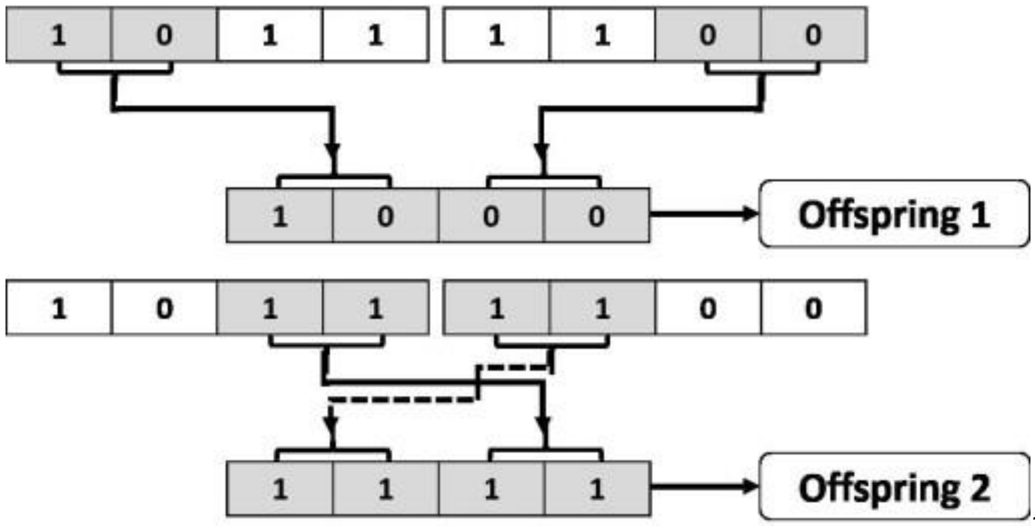
Mutation probability,

Fitness function.
For example, there are different types of parent selection, like rank and roulette wheel, and you should know which one to use when designing the algorithm for a specific problem.

The parameter we'll be focusing on is mutation probability. So, let's review the mutation operation, and whether high or low mutation probability is better.

How mutation works

Given two parents to mate, the first operation in the mating process is the crossover. The produced child just transfers some genes from its two parents. There's nothing new in the child, as all of its genes are already existing in its parents. The next figure shows how crossover works.

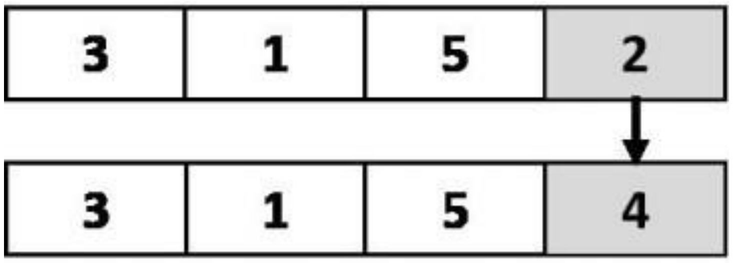


This figure is copyrighted material in this book and should not be used without permission.

If there are some bad genes within the parents, they will definitely be transferred to their children after crossover. The mutation operation plays a crucial role in fixing this issue.

During mutation, some genes are randomly selected from each child where some random changes are applied. Genes are selected based on a random probability for each gene. If the probability of mutating a gene is smaller than or equal to a predefined threshold, then this gene will be selected for mutation. Otherwise, it will be skipped. We'll discuss mutation probability later on.

Let’s assume there are 4 genes in the solution, as in the next figure, where only the last gene is selected for mutation. A random change is applied to change its old value **2** and the new value is **4**.



After briefly reviewing how random mutation works, next we’ll solve a problem using the genetic algorithm with random mutation.

PROCEDURE / PROGRAMME

```
# genetic algorithm search for continuous function optimization
from numpy.random import randint
from numpy.random import rand

# objective function
# fitness function
def objective(x):
    """
    # the function to optimize
    >> x**2 - y * y**0.5 + 14
    """
    return (x[0] ** 2.0) - (x[1] * (x[1] ** 0.5)) + 14


# decode bitstring to numbers
def decode(bounds, n_bits, bitstring):
    """
    assume bitstring is in binary, decode into number(s) for the function input(s)
    """
    decoded = list()
    largest = 2 ** n_bits
    for i in range(len(bounds)):
        # extract the substring
        start, end = i * n_bits, (i * n_bits) + n_bits
        substring = bitstring[start:end]
        # convert bitstring to a string of chars
        chars = "".join([str(s) for s in substring])
        # convert string to integer
        integer = int(chars, 2)
        # scale integer to desired range
        value = bounds[i][0] + (integer / largest) * (bounds[i][1] - bounds[i][0])
        # store
        decoded.append(value)
    # will return array of length same as length of bounds
    # ie: decoded values for all variables
    return decoded


# tournament selection
def selection(pop, scores, k=3):
    """
    pick k tournaments and pick the best parent in each call
    best is determined by scores array (fitness scores (based on objective function))
    """
    # first random selection
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k - 1):
        # check if better (e.g. perform a tournament)
        if scores[ix] > scores[selection_ix]:
            selection_ix = ix
    return pop[selection_ix]


# crossover two parents to create two children
def crossover(p1, p2, r_cross):
    """
    r_cross (crossover rate) is a hyperparameter that determines whether crossover
is performed or not,
    and if not, the parents are copied into the next generation (default behaviour in
this case)

    It is a probability and typically has a large value close to 1.0.
    """
    # children are copies of parents by default
    c1, c2 = p1.copy(), p2.copy()
    # check for recombination
    if rand() < r_cross:
        # select crossover point that is not on the end of the string
        pt = randint(1, len(p1) - 2)
        # perform crossover
        """
        one-point crossover
        """
        c1 = p1[:pt] + p2[pt:]
        c2 = p2[:pt] + p1[pt:]
    return [c1, c2]
```

```
# mutation operator
def mutation(bitstring, r_mut):
    """
    mutate bitstring itself, NOT the copy
    """
    for i in range(len(bitstring)):
        # check for a mutation
        if rand() < r_mut:
            # flip the bit
            """
            bit-flip mutation
                1 - (1) => 0
                1 - (0) => 1
            """
            bitstring[i] = 1 - bitstring[i]


# genetic algorithm
def genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut):
    # initial population of random bitstring
    pop = [randint(0, 2, n_bits * len(bounds)).tolist() for _ in range(n_pop)]
    """
    pop =>
    [
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
1, 0, 0]
        ...
        ...
        ...
        (n_pop)th array
    ]
    """

    # keep track of best solution
    best, best_eval = 0, objective(decode(bounds, n_bits, pop[0]))

    # enumerate generations
    for gen in range(n_iter):
        # decode population
        decoded = [decode(bounds, n_bits, p) for p in pop]

        # evaluate all candidates in the population
        scores = [objective(d) for d in decoded]

        # check for new best solution
        for i in range(n_pop):
            if scores[i] > best_eval:
                best, best_eval = pop[i], scores[i]
                print(">%d, new best f(%s) = %f" % (gen, decoded[i], scores[i]))
        # select parents
        selected = [selection(pop, scores) for _ in range(n_pop)]
        # `selected` is a length 100 array of each element being 32 length arrays

        # create the next generation
        children = list()
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i + 1]
            # crossover and mutation
            for c in crossover(p1, p2, r_cross):
                # mutation
                mutation(c, r_mut)
                # store for next generation
                children.append(c)

        # replace population
        pop = children

    return [best, best_eval]


# define range for each input variable
bounds = [[0.0, 15.0], [12.0, 17.0]]
# define the total iterations
n_iter = 100
# bits per variable
n_bits = 16
# define the population size
n_pop = 100
# crossover rate: high probability value
r_cross = 0.9
# mutation rate: low probability value
r_mut = 1.0 / (float(n_bits) * len(bounds))
# perform the genetic algorithm search
best, score = genetic_algorithm(
    objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut
)
# getting the best population and the best score possible
print("Done!")
decoded = decode(bounds, n_bits, best)
# decoding best population to value
print("f(%s) = %f" % (decoded, score))
```

Output

```
>0, new best f([7.9882049560546875, 12.405349731445312]) = 34.118252
>0, new best f([11.887664794921875, 15.736419677734375]) = 92.891525
>0, new best f([13.221588134765625, 14.141799926757812]) = 135.629327
>0, new best f([12.9400634765625, 12.668716430664062]) = 136.353303
>0, new best f([13.5699462890625, 14.339630126953125]) = 143.842555
>0, new best f([14.693527221679688, 12.138702392578125]) = 187.607725
```

```
>0, new best f([14.81231689453125, 12.377120971679688]) = 189.860618
>1, new best f([14.81964111328125, 12.37796020507812 5]) = 190.073220
>1, new best f([14.81048583984375, 12.104522705078125]) = 191.236975
>3, new best f([14.81231689453125, 12.104827880859375]) = 191.289623
>3, new best f([14.811630249023438, 12.054473876953125]) = 191.531796
>4, new best f([14.93499755859375, 12.525588989257812]) = 192.724203
>5, new best f([14.99542236328125, 12.737686157226562]) = 193.402024
>5, new best f([14.925155639648438, 12.127792358398438]) = 194.525258
>6, new best f([14.925155639648438, 12.127639770507812]) = 194.526055
>6, new best f([14.994735717773438, 12.018768310546875]) = 197.175319
>11, new best f([14.996109008789062, 12.0262451171875]) = 197.177618
>12, new best f([14.99908447265625, 12.0360107421875]) = 197.216058
>13, new best f([14.9981689453125, 12.0274658203125]) = 197.233054
>13, new best f([14.998397827148438, 12.0262451171875]) = 197.246270
>14, new best f([14.995651245117188, 12.009384155273438]) = 197.251566
>15, new best f([14.998397827148438, 12.0164794921875]) = 197.297059
>16, new best f([14.9981689453125, 12.0079345703125]) = 197.334616
>17, new best f([14.999771118164062, 12.008544921875]) = 197.379506
>19, new best f([14.999771118164062, 12.003662109375]) = 197.404884
>23, new best f([14.999771118164062, 12.00244140625]) = 197.411228
>25, new best f([14.999771118164062, 12.001220703125]) = 197.417571
>26, new best f([14.999771118164062, 12.0]) = 197.423914
Done!
f([14.999771118164062, 12.0]) = 197.423914
```

EXPERIMENT NO:9

Implement the finite words classification system using Back-propagation algorithm

Aim:

To implement the finite words classification system using Back-propagation algorithm

Theory:

• Artificial neural networks (ANNs) provide a general, practical method for learning realvalued,
discrete-valued, and vector-valued functions from examples.
• Algorithms such as BACKPROPAGATION gradient descent to tune network parameters to
best fit a training set of input-output pairs.
• ANN learning is robust to errors in the training data and has been successfully applied to
problems such as interpreting visual scenes, speech recognition, and learning robot control
strategies.
Backpropogation algorithm

```
1. Create a feed-forward network with ni inputs, nhidden hidden units, and nout output
units.
2. Initialize each wi to some small random value (e.g., between -.05 and .05).
3. Until the termination condition is met, do
For each training example <(x1,…xn),t>, do
// Propagate the input forward through the network:
a. Input the instance (x1, ..,xn) to the n/w & compute the n/w outputs ok for every
unit
// Propagate the errors backward through the network:
b. For each output unit k, calculate its error term ☐k ; ☐k = ok(1-ok)(tk-ok)
c. For each hidden unit h, calculate its error term ☐h; ☐h=oh(1-oh) ☐k wh,k ☐k
d. For each network weight wi,j do; wi,j=wi,j+☐wi,j where ☐wi,j= ☐ ☐j xi,j
```

PROCEDURE / PROGRAMME

```
import numpy as np # numpy is commonly used to process number array

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # Features ( Hrs Slept, Hrs Studied)

y = np.array(([92], [86], [89]), dtype=float) # Labels(Marks obtained)

X = X/np.amax(X,axis=0) # Normalize

y = y/100

def sigmoid(x):

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

def sigmoid_grad(x):

 return x * (1 - x)

# Variable initialization

epoch=1000 #Setting training iterations

eta =0.2 #Setting learning rate (eta)

input_neurons = 2 #number of features in data set

hidden_neurons = 3 #number of hidden layers neurons

output_neurons = 1 #number of neurons at output layer

# Weight and bias - Random initialization

wh=np.random.uniform(size=(input_neurons,hidden_neurons)) # 2x3

bh=np.random.uniform(size=(1,hidden_neurons)) # 1x3

wout=np.random.uniform(size=(hidden_neurons,output_neurons)) # 1x1
```

```
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation

 h_ip=np.dot(X,wh) + bh # Dot product + bias

 h_act = sigmoid(h_ip) # Activation function

 o_ip=np.dot(h_act,wout) + bout

 output = sigmoid(o_ip)

#Backpropagation

# Error at Output layer

Eo = y-output # Error at o/p

outgrad = sigmoid_grad(output)

d_output = Eo* outgrad # Errj=Oj(1-Oj)(Tj-Oj)

# Error at Hidden later

Eh = d_output.dot(wout.T) # .T means transpose

hiddengrad = sigmoid_grad(h_act) # How much hidden layer wts contributed to error

d_hidden = Eh * hiddengrad

wout += h_act.T.dot(d_output) *eta # Dotproduct of nextlayererror and
currentlayerop

wh += X.T.dot(d_hidden) *eta

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

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

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

Output

```
Normalized Input:

[[0.66666667 1.        ]

 [0.33333333 0.55555556]

 [1.         0.66666667]]
```

```
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.77295493]
 [0.75973893]
 [0.78167013]]
```