RAJALAKSHMI ENGINEERING COLLEGE RAJALAKSHMI NAGAR, THANDALAM – 602 105



CS23331 FUNDAMENTAL OF MACHINE LEARNING LAB

Laboratory Observation Note Book

Name : Giriyaşanth V
Year / Branch / Section : 2 nd Year / AIML / A
Register No.: 231501048
Semester: 3 rd Semester
Academic Year : 2024-2025

INDEX

Reg. No: 231501048

Year: 2nd Year

Name: Girivasanth V

Branch: AIML Sec: A

Date	Title	Page No.	Teacher's Signature / Remarks
/ /24	A PYTHON PROGRAM TO IMPLEMENT		
/ /24	A PYTHON PROGRAM TO IMPLEMENT SIMPLE		
	LINEAR REGRESSION USING LEAST SQUARE		
	METHOD		
/ /24	A PYTHON PROGRAM TO IMPLEMENT LOGISTIC		
	MODEL		
/ /24	A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER		
	PERCEPTRON		
/ /24	A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER		
	PERCEPTRON WITH BACK PROPOGATION		
/ /24	A PYTHON PROGRAM TO IMPLEMENT SVM		
	CLASSIFIER MODEL		
/ /24	A PYTHON PROGRAM TO IMPLEMENT DECISION TREE		
/ /24	A PYTHON PROGRAM TO IMPLEMENT ADA		
	BOOSTING		
/ /24	A PYTHON PROGRAM TO IMPLEMENT KNN MODEL		
/ /24	A PYTHON PROGRAM TO IMPLEMENT		
	DIMENSIONALITY REDUCTION USING		
	/ /24 / /24 / /24 / /24 / /24 / /24 / /24	/ /24 A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESSION A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD / /24 A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL / /24 A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON / /24 A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER PERCEPTRON WITH BACK PROPOGATION / /24 A PYTHON PROGRAM TO IMPLEMENT SVM CLASSIFIER MODEL / /24 A PYTHON PROGRAM TO IMPLEMENT DECISION TREE / /24 A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING / /24 A PYTHON PROGRAM TO IMPLEMENT KNN MODEL	No. No.

Ex No: 1 Date:

A PYTHON PROGRAM TO IMPLEMENT UNIVARIATE, BIVARIATE AND MULTIVARIATE REGRESION

Aim:

To implement a python program using univariate, bivariate and multivariate regression features for a given iris dataset.

Algorithm:

Step 1: Import necessary libraries:

 pandas for data manipulation, numpy for numerical operations, and matplotlib.pyplot for plotting.

Step 2: Read the dataset:

- Use the pandas 'read csv' function to read the dataset.
- Store the dataset in a variable (e.g., `data`).

Step 3: Prepare the data:

- Extract the independent variable(s) (X) and dependent variable (y) from the dataset.
- Reshape X and y to be 2D arrays if needed.

Step 4:Univariate Regression:

- For univariate regression, use only one independent variable.
- Fit a linear regression model to the data using numpy's polyfit function or sklearn's LinearRegression class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 5: Bivariate Regression:

- For bivariate regression, use two independent variables.
- Fit a linear regression model to the data using numpy's `polyfit` function or sklearn's `LinearRegression` class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 6: Multivariate Regression:

- For multivariate regression, use more than two independent variables.
- Fit a linear regression model to the data using sklearn's `LinearRegression` class.
- Make predictions using the model.
- Calculate the R-squared value to evaluate the model's performance.

Step 7: Plot the results:

- For univariate regression, plot the original data points (X, y) as a scatter plot and the regression line as a line plot.
- For bivariate regression, plot the original data points (X1, X2, y) as a 3D scatter plot and the regression plane.
- For multivariate regression, plot the predicted values against the actual values.

Step 8: Display the results:

- Print the coefficients (slope) and intercept for each regression model.
- Print the R-squared value for each regression model.

Step 9: Complete the program:

- Combine all the steps into a Python program.
- Run the program to perform univariate, bivariate, and multivariate regression on the dataset.

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df=pd.read_csv('/content/drive/MyDrive/Datasets/iris.csv')
df.head(150)
df.shape
df
df_Setosa=df.loc[df['species']=='setosa']
```

```
df_Virginica=df.loc[df['species']=='virginica']
df Versicolor=df.loc[df['species']=='versicolor']
df Setosa
#univariate for sepal width
plt.scatter(df_Setosa['sepal_width'],np.zeros_like(df_Setosa['sepal_width']))
plt.scatter(df_Virginica['sepal_width'],np.zeros_like(df_Virginica['sepal_width']
))
plt.scatter(df_Versicolor['sepal_width'],np.zeros_like(df_Versicolor['sepal_widt
h']))
plt.xlabel('sepal width')
plt.show()
#univariate for sepal length
plt.scatter(df Setosa['sepal length'],np.zeros like(df Setosa['sepal length']))
plt.scatter(df_Virginica['sepal_length'],np.zeros_like(df_Virginica['sepal_length']
']))
plt.scatter(df_Versicolor['sepal_length'],np.zeros_like(df_Versicolor['sepal_len
gth']))
plt.xlabel('sepal_length')
plt.show()
#univariate for sepal width
plt.scatter(df_Setosa['petal_width'],np.zeros_like(df_Setosa['petal_width']))
plt.scatter(df_Virginica['petal_width'],np.zeros_like(df_Virginica['petal_width']
))
plt.scatter(df\_Versicolor['petal\_width'], np.zeros\_like(df\_Versicolor['petal\_width'], np.zeros\_like(df\_Versi
h']))
plt.xlabel('petal_width')
plt.show()
```

```
#univariate for sepal length
plt.scatter(df_Setosa['petal_length'],np.zeros_like(df_Setosa['petal_length']))
plt.scatter(df_Virginica['petal_length'],np.zeros_like(df_Virginica['petal_length'
]))
plt.scatter(df_Versicolor['petal_length'],np.zeros_like(df_Versicolor['petal_leng
th']))
plt.xlabel('petal length')
plt.show()
#bivariate sepal.width vs petal.width
sns.FacetGrid(df,hue='species',height=5).map(plt.scatter,"sepal_width","petal_
width").add_legend();
plt.show()
#bivariate sepal.length vs petal.length
sns.FacetGrid(df,hue='species',height=5).map(plt.scatter,"sepal_length","petal
_length").add_legend();
plt.show()
#multivariate all the features
sns.pairplot(df,hue='species',size=2)
```

Output:

_	_	_
•	_	•
_	→	$\overline{}$
	٠	

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

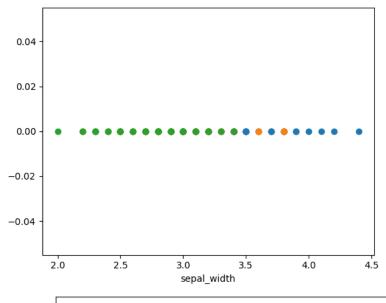
150 rows × 5 columns

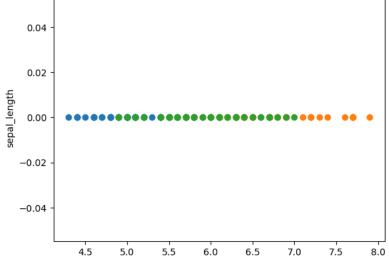
-	$\overline{}$	_
-	-	$\overline{}$
	÷	_

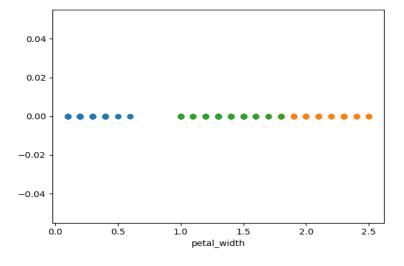
→ ▼	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa

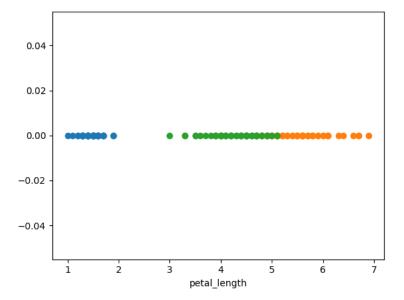
→	12	4.8	3.0	1.4	0.1	setosa
	13	4.3	3.0	1.1	0.1	setosa
	14	5.8	4.0	1.2	0.2	setosa
	15	5.7	4.4	1.5	0.4	setosa
	16	5.4	3.9	1.3	0.4	setosa
	17	5.1	3.5	1.4	0.3	setosa
	18	5.7	3.8	1.7	0.3	setosa
	19	5.1	3.8	1.5	0.3	setosa
	20	5.4	3.4	1.7	0.2	setosa
	21	5.1	3.7	1.5	0.4	setosa
	22	4.6	3.6	1.0	0.2	setosa
	23	5.1	3.3	1.7	0.5	setosa
	24	4.8	3.4	1.9	0.2	setosa
0	25	5.0	3.0	1.6	0.2	setosa
→	26	5.0	3.4	1.6	0.4	setosa
	27	5.2	3.5	1.5	0.2	setosa
	28	5.2	3.4	1.4	0.2	setosa
	29	4.7	3.2	1.6	0.2	setosa
	30	4.8	3.1	1.6	0.2	setosa
	31	5.4	3.4	1.5	0.4	setosa
	32	5.2	4.1	1.5	0.1	setosa
	33	5.5	4.2	1.4	0.2	setosa
	34	4.9	3.1	1.5	0.1	setosa
	35	5.0	3.2	1.2	0.2	setosa
	36	5.5	3.5	1.3	0.2	setosa
	37	4.9	3.1	1.5	0.1	setosa
	38	4.4	3.0	1.3	0.2	setosa
	39	5.1	3.4	1.5	0.2	setosa
	40	5.0	3.5	1.3	0.3	setosa
	41	4.5	2.3	1.3	0.3	setosa
	42	4.4	3.2	1.3	0.2	setosa
	43	5.0	3.5	1.6	0.6	setosa

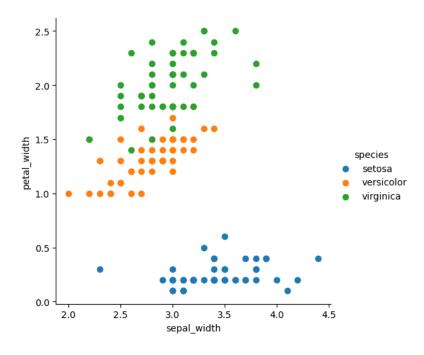
44	5.1	3.8	1.9	0.4	setosa
45	4.8	3.0	1.4	0.3	setosa
46	5.1	3.8	1.6	0.2	setosa
47	4.6	3.2	1.4	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa
49	5.0	3.3	1.4	0.2	setosa

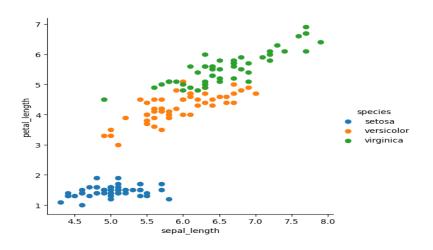


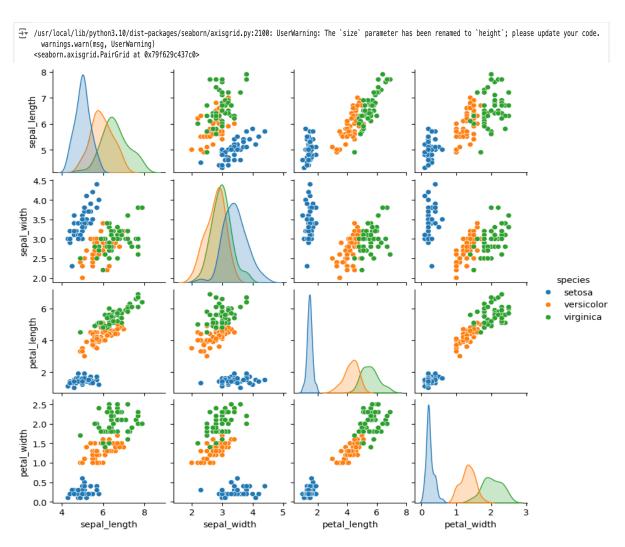












Result:

Thus, the python program to implement univariate, bivariate and multivariate has been successfully implemented and the results have been verified and analysed.

Ex No: 2

Date:

A PYTHON PROGRAM TO IMPLEMENT SIMPLE LINEAR REGRESSION USING LEAST SQUARE METHOD

<u>Aim:</u>

To implement a python program for constructing a simple linear regression using least square method.

Algorithm:

Step 1: Import necessary libraries:

pandas for data manipulation and matplotlib.pyplot for plotting.

Step 2: Read the dataset:

- Use the pandas `read_csv` function to read the dataset (e.g., headbrain.csv).
- Store the dataset in a variable (e.g., `data`).

Step 3: Prepare the data:

- Extract the independent variable (X) and dependent variable (y) from the dataset.
- Reshape X and y to be 2D arrays if needed.

Step 4: Calculate the mean:

Calculate the mean of X and y.

Step 5: Calculate the coefficients:

• Calculate the slope (m) using the formula:

$$m = rac{\sum_{i=1}^{n} (X_i - ar{X})(y_i - ar{y})}{\sum_{i=1}^{n} (X_i - ar{X})^2}$$

• Calculate the intercept (b) using the formula: $b=ar{y}-mar{X}$

Step 6: Make predictions:

Use the calculated slope and intercept to make predictions for each X value:

$$\hat{y} = mx + b$$

Step 7: Plot the regression line:

- Plot the original data points (X, y) as a scatter plot.
- Plot the regression line (X, predicted_y) as a line plot.

Step 8: Calculate the R-squared value:

- Calculate the total sum of squares (TSS) using the formula: $TSS = \sum_{i=1}^n (y_i \bar{y})^2$
- Calculate the residual sum of squares (RSS) using the formula: $RSS = \sum_{i=1}^n (y_i \hat{y}_i)^2$
- Calculate the R-squared value using the formula: $R^2=1-rac{RSS}{TSS}$

Step 9: Display the results:

• Print the slope, intercept, and R-squared value.

Step 10: Complete the program:

• Combine all the steps into a Python program.

• Run the program to perform simple linear regression on the dataset.

Code:

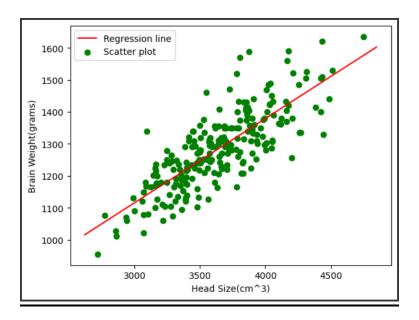
```
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data =
pd.read csv('/content/drive/MyDrive/Datasets/headbrain.csv')
x, y = np.array(list(data['Head Size(cm^3)'])),np.array(list(data['Brain
Weight(grams)']))
print(x[:5], y[:5])
def get line(x, y):
 x_m, y_m = np.mean(x), np.mean(y)
 print(x m, y m)
 x d, y d = x-x m, y-y m
 m = np.sum(x d*y d)/np.sum(x d**2)
 c = y m - (m*x m)
 print(m, c)
 return lambda x : m*x+c
lin = get line(x, y)
X = np.linspace(np.min(x)-100, np.max(x)+100, 1000)
Y = np.array([lin(x) for x in X])
plt.plot(X, Y, color='red', label='Regression line')
plt.scatter(x, y, color='green', label='Scatter plot')
plt.xlabel('Head Size(cm^3)')
plt.ylabel('Brain Weight(grams)')
plt.legend()
plt.show()
```

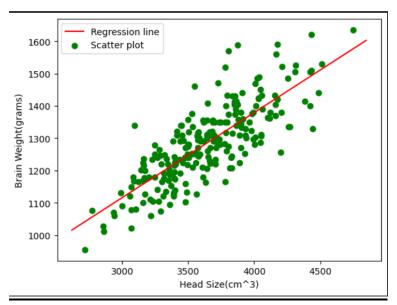
```
X = np.linspace(np.min(x)-100, np.max(x)+100, 1000)
Y = np.array([lin(x) for x in X])
plt.plot(X, Y, color='red', label='Regression line')
plt.scatter(x, y, color='green', label='Scatter plot')
plt.xlabel('Head Size(cm^3)')
plt.ylabel('Brain Weight(grams)')
plt.legend()
plt.show()
def get error(line fuc, x, y):
 y_m = np.mean(y)
 y_pred = np.array([line_fuc(_) for _ in x])
 ss t = np.sum((y-y m)**2)
 ss r = np.sum((y-y pred)**2)
 return 1-(ss_r/ss_t)
get_error(lin, x, y)
from sklearn.linear model import LinearRegression
x = x.reshape((len(x),1))
reg=LinearRegression()
reg=reg.fit(x, y)
print(reg.score(x, y))
```

Output:

[4512 3738 4261 3777 4177] [1530 1297 1335 1282 1590]

3633.9915611814345 1282.873417721519 0.2634293394893993 325.5734210494428





0.639311719957

0**.**639311719957

Result:

Thus, the python program to Simple Linear Regression using Least Square Method has been successfully implemented and the results have been verified and analysed.

Ex no: 3

Date:

A PYTHON PROGRAM TO IMPLEMENT LOGISTIC MODEL

Aim:

To implement python program for the logistic model using suv car dataset.

Algorithm:

Step 1: Import Necessary Libraries:

- pandas for data manipulation
- sklearn.model_selection for train-test split
- sklearn.preprocessing for data preprocessing
- sklearn.linear_model for logistic regression
- matplotlib.pyplot for plotting

Step 2: Read the Dataset:

• Use pandas to read the suv_cars.csv dataset into a DataFrame.

Step 3: Preprocess the Data:

- Select the relevant columns for the analysis (e.g., 'Age', 'EstimatedSalary', 'Purchased').
- Encode categorical variables if necessary (e.g., using LabelEncoder or OneHotEncoder).
- Split the data into features (X) and target variable (y).

Step 4: Split the Data:

 Split the dataset into training and testing sets using train_test_split.

Step 5: Feature Scaling:

 Standardize the features using StandardScaler to ensure they have the same scale.

Step 6: Create and Train the Model:

- Create a logistic regression model using LogisticRegression from sklearn.linear_model.
- Train the model on the training data using the fit method.
 - Create a function named "Sigmoid ()" which will define the sigmoid values using the
 - formula (1/1+e-z) and return the computed value.
 - Create a function named "initialize()" which will initialize the values with zeroes and assign the value to "weights" variable, initializes with ones and assigns the value to variable "x" and returns both "x" and "weights".
 - Create a function named "fit" which will be used to plot the graph according to the training data.
 - Create a predict function that will predict values according to the training model created using the fit function.
 - Invoke the standardize() function for "x-train" and "x-test"

Step 7: Make Predictions:

- Use the trained model to make predictions on the test data using the predict method.
 - Use the "predict()" function to predict the values of the testing data and assign the value to "y_pred" variable.
 - Use the "predict()" function to predict the values of the training data and assign the value to "y_trainn" variable.
 - Compute f1_score for both the training and testing data and assign the values to "f1_score_tr" and "f1_score_te" respectively

Step 8: Evaluate the Model:

 Calculate the accuracy of the model on the test data using the score method.

(Accuracy = (tp+tn)/(tp+tn+fp+fn)).

• Generate a confusion matrix and classification report to further evaluate the model's performance.

Step 9: Visualize the Results:

 Plot the decision boundary of the logistic regression model (optional).

Code:

import pandas as pd

import numpy as np

from numpy import log,dot,exp,shape

from sklearn.metrics import confusion_matrix

data = pd.read_csv('/content/drive/MyDrive/suv_data.csv')

```
print(data.head())
x = data.iloc[:, [2, 3]].values
y = data.iloc[:, 4].values
from sklearn.model selection import train test split
x train, x test, y train, y test=train test split(x,y,test size=0.10,
random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x train=sc.fit transform(x train)
x test=sc.transform(x test)
print (x train[0:10,:])
from sklearn.linear model import LogisticRegression
classifier=LogisticRegression(random_state=0)
classifier.fit(x train,y train)
LogisticRegression (random state=0)
y pred = classifier.predict(x test)
print(y pred)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)
```

```
from sklearn.metrics import accuracy score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
from sklearn.model selection import train test split
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.10,
random state=0)
def Std(input data):
  mean0 = np.mean(input data[:, 0])
  sd0 = np.std(input_data[:, 0])
  mean1 = np.mean(input data[:, 1])
  sd1 = np.std(input data[:, 1])
  return lambda x:((x[0]-mean0)/sd0, (x[1]-mean1)/sd1)
my std = Std(x)
my std(x train[0])
def standardize(X tr):
  for i in range(shape(X tr)[1]):
    X_{tr}[:,i] = (X_{tr}[:,i] - np.mean(X_{tr}[:,i]))/np.std(X_{tr}[:,i])
def F1_score(y,y_hat):
  tp,tn,fp,fn = 0,0,0,0
  for i in range(len(y)):
    if y[i] == 1 and y hat[i] == 1:
         tp += 1
    elif y[i] == 1 and y hat [i] == 0:
         fn += 1
```

```
elif y[i] == 0 and y hat[i] == 1:
         fp += 1
    elif y[i] == 0 and y_hat[i] == 0:
         tn += 1
  precision = tp/(tp+fp)
  recall = tp/(tp+fn)
  f1_score = 2*precision*recall/(precision+recall)
  return f1 score
class LogisticRegression:
  def sigmoid(self, z):
    sig = 1 / (1 + exp(-z))
    return sig
  def initialize(self, X):
    weights = np.zeros((shape(X)[1] + 1, 1))
    X = np.c [np.ones((shape(X)[0], 1)), X]
    return weights, X
  def fit(self, X, y, alpha=0.001, iter=400):
    weights, X = self.initialize(X)
    def cost(theta):
       z = dot(X, theta)
```

```
cost0 = y.T.dot(log(self.sigmoid(z)))
       cost1 = (1 - y).T.dot(log(1 - self.sigmoid(z)))
       cost = -((cost1 + cost0)) / len(y)
       return cost
    cost_list = np.zeros(iter,)
    for i in range(iter):
       weights = weights - alpha * dot(X.T, self.sigmoid(dot(X,
weights)) - np.reshape(y, (len(y), 1)))
       cost_list[i] = cost(weights).item()
    self.weights = weights
    return cost list
  def predict(self, X):
    z = dot(self.initialize(X)[1], self.weights)
    lis = []
    for i in self.sigmoid(z):
       if i > 0.5:
         lis.append(1)
       else:
         lis.append(0)
     return lis
standardize(x_train)
```

```
standardize(x_test)
obj1 = LogisticRegression()
model = obj1.fit(x_train, y_train)
y_pred = obj1.predict(x_test)
y_trainn = obj1.predict(x_train)
f1_score_tr = F1_score(y_train, y_trainn)
f1_score_te = F1_score(y_test, y_pred)
print(f1_score_tr)
print(f1_score_te)
conf_mat = confusion_matrix(y_test, y_pred)
accuracy = (conf_mat[0, 0] + conf_mat[1, 1]) / sum(sum(conf_mat))
print("Accuracy is : ", accuracy)
```

Output:

```
User ID Gender Age EstimatedSalary
                                                 Purchased
     0 15624510
                                          19000
                    Male
       15810944
                                                         0
       15668575 Female
                            26
                                          43000
                                          57000
                                                         0
        15603246 Female
                           27
        15804002
                    Male
                                          76000
[[-1.05714987 0.53420426]
[ 0.2798728 -0.51764734]
[-1.05714987 0.41733186]
[-0.29313691 -1.45262654]
[ 0.47087604 1.23543867]
[-1.05714987 -0.34233874]
[-0.10213368 0.30045946]
[ 1.33039061 0.59264046]
[-1.15265148 -1.16044554]
[ 1.04388575  0.47576806]]
```

 $[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

0 0 1]

Confusion Matrix:

[[31 1]

[1 7]]

Accuracy: 0.95

(-1.017692393473028, 0.5361288690822568)

0.75833333333333334

0.823529411764706 Accuracy is: 0.925

Result:

Thus, the python program to implement logistic model has been successfully implemented and the results have been verified and analyzed.

Ex. No.: 4

Date:

A PYTHON PROGRAM TO IMPLEMENT SINGLE LAYER PERCEPTRON

Aim:

To implement python program for the single layer perceptron.

Algorithm:

Step 1: Import Necessary Libraries:

• Import numpy for numerical operations.

Step 2: Initialize the Perceptron:

- Define the number of input features (input_dim).
- Initialize weights (W) and bias (b) to zero or small random values.

Step 3: Define Activation Function:

- Choose an activation function (e.g., step function, sigmoid, or ReLU).
- User Defined function sigmoid func(x):
 - Compute 1/(1+np.exp(-x)) and return the value.
- User Defined function der(x):
 - Compute the product of value of sigmoid_func(x) and (1 sigmoid_func(x)) and return the value.

Step 4; Define Training Data:

• Define input features (X) and corresponding target labels (y).

Step 5: Define Learning Rate and Number of Epochs:

 Choose a learning rate (alpha) and the number of training epochs.

Step 6: Training the Perceptron:

- For each epoch:
 - For each input sample in the training data:
 - Compute the weighted sum of inputs (z) as the dot product of input features and weights plus bias (z = np.dot(X[i], W) + b).
 - Apply the activation function to get the predicted output (y pred).
 - Compute the error (error = y[i] y_pred).
 - Update the weights and bias using the learning rate and error (W += alpha * error * X[i]; b += alpha * error).

Step 7: Prediction:

• Use the trained perceptron to predict the output for new input data.

Step 8: Evaluate the Model:

• Measure the performance of the model using metrics such as accuracy, precision, recall, etc.

Code:

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
input_dim=2
W=np.zeros(input_dim)
b=0.0

def sigmoid_func(x):
```

```
return 1/(1 + np.exp(-x))
def der(x):
  sigmoid = sigmoid func(x)
  return sigmoid * (1 - sigmoid)
np.random.seed(42)
x = np.array([[150,8],
       [130,7],
       [180,6],
       [170,5]])
y = np.array([0,0,1,1])
alpha = 0.1
epochs = 10000
for epoch in range(epochs):
  for i in range(len(x)):
    z = np.dot(x[i], W) + b
    y pred = sigmoid func(z)
    error = y[i] - y_pred
    W += alpha * error * x[i]
    b += alpha * error
def predict(X):
  z = np.dot(X, W) + b
  return (sigmoid_func(z) > 0.5).astype(int)
y pred = predict(x)
accuracy = accuracy_score(y, y_pred)
precision = precision score(y, y pred)
recall = recall_score(y, y_pred)
F1_score = f1_score(y, y_pred)
print("Prediction:",y pred)
print("Accuracy:", accuracy)
```

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", F1_score)

Output:

Prediction: [0 0 1 1]

Accuracy: 1.0 Precision: 1.0 Recall: 1.0 F1 Score: 1.0

Result:

Thus, the python program to implement single layer perceptron has been successfully implemented and the results have been verified and analysed.

Ex. No.: 5

Date:

A PYTHON PROGRAM TO IMPLEMENT MULTI LAYER PERCEPTRON WITH BACK PROPOGATION

Aim:

To implement multilayer perceptron with back propagation using python.

Algorithm:

Step 1: Import the Necessary Libraries

- Import pandas as pd.
- Import numpy as np.

Step 2: Read and Display the Dataset

- Use `pd.read_csv("banknotes.csv")` to read the dataset.
- Assign the result to a variable (e.g., `data`).
- Display the first ten rows using `data.head(10)`.

Step 3: Display Dataset Dimensions

• Use the `.shape` attribute on the dataset (e.g., `data.shape`).

Step 4: Display Descriptive Statistics

• Use the `.describe()` function on the dataset (e.g., `data.describe()`).

Step 5: Import Train-Test Split Module

• Import `train_test_split` from `sklearn.model_selection`.

Step 6: Split Dataset with 80-20 Ratio

- Assign the features to a variable (e.g., `X = data.drop(columns='target')`).
- Assign the target variable to another variable (e.g., `y = data['target']`).
- Use `train_test_split` to split the dataset into training and testing sets with a ratio of 0.2.
- Assign the results to `x_train`, `x_test`, `y_train`, and `y_test`.

Step 7: Import MLPClassifier Module

• Import `MLPClassifier` from `sklearn.neural network`.

Step 8: Initialize MLPClassifier

- Create an instance of `MLPClassifier` with `max_iter=500` and `activation='relu'`.
- Assign the instance to a variable (e.g., `clf`).

Step 9: Fit the Classifier

• Fit the model using `clf.fit(x train, y train)`.

Step 10: Make Predictions

- Use the `.predict()` function on `x_test` (e.g., `pred = clf.predict(x_test)`).
- Display the predictions.

Step 11: Import Metrics Modules

- Import `confusion matrix` from `sklearn.metrics`.
- Import `classification_report` from `sklearn.metrics`.

Step 12: Display Confusion Matrix

- Use `confusion matrix(y test, pred)` to generate the confusion matrix.
- Display the confusion matrix.

Step 13: Display Classification Report

- Use `classification_report(y_test, pred)` to generate the classification report.
- Display the classification report.

Step 14: Repeat Steps 9-13 with Different Activation Functions

- Initialize `MLPClassifier` with `activation='logistic'`.
- Fit the model and make predictions.

- Display the confusion matrix and classification report.
- Repeat for `activation='tanh'`.
- Repeat for `activation='identity'`.

Step 15: Repeat Steps 7-14 with 70-30 Ratio

- Use `train_test_split` to split the dataset into training and testing sets with a ratio of 0.3.
- Assign the results to `x_train`, `x_test`, `y_train`, and `y_test`.
- Repeat Steps 7-14 with the new training and testing sets.

Code:

```
import pandas as pd
import numpy as np
from sklearn.model selection import train test split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
bnotes = pd.read csv('../content/drive/MyDrive/bank note data.csv')
print(bnotes.head(10))
x = bnotes.drop('Class', axis=1)
y = bnotes['Class']
print(x.head(2))
print(y.head(2))
def train_and_evaluate(activation, x_train, y_train, x_test, y_test):
  mlp = MLPClassifier(max iter=500, activation=activation)
  mlp.fit(x_train, y_train)
  pred = mlp.predict(x test)
  print(f"Predictions using activation function '{activation}':\n{pred}\n")
  cm = confusion matrix(y test, pred)
  print(f"Confusion Matrix for '{activation}':\n{cm}\n")
  report = classification_report(y_test, pred)
  print(f"Classification Report for '{activation}':\n{report}\n")
```

```
for activation in ['relu', 'logistic', 'tanh', 'identity']:
  train_and_evaluate(activation, x_train, y_train, x_test, y_test)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
for activation in ['relu', 'logistic', 'tanh', 'identity']:
  train_and_evaluate(activation, x_train, y_train, x_test, y_test)
Output:
               Image.Skew
                         Image.Curt
 \rightarrow
      Image.Var
                                   Entropy
        3.62160
4.54590
3.86600
                                  -0.44699
-1.46210
                   8.6661
                           -2.80730
                   8.1674
-2.6383
                           -2.45860
1.92420
                                   0.10645
        3.45660
                   9.5228
                           -4.01120 -3.59440
        0.32924
4.36840
                   -4.4552
                                  -0.98880
                   9.6718
                                  -3.16250
                           -3.96060
                            0.72888
8.46360
        3.59120
                   3.0129
                                   0.56421
        2.09220
3.20320
                  -6.8100
                                  -0.61251
                   5.7588
                           -0.75345
       1.53560
Image.Var
3.6216
               9.1772
Image.Skew
8.6661
                         -2.27180
Image.Curt
                                  -0.73535
                                   Entropy
                            -2.8073
                                  -0.44699
    0 1
1 0
0 0
                                                            0 0 0
0 1 1
0 1 0
                                                 0
1
                                                      0
0
1
0
1
                                                       0
1
1
0
0
                                                           0
1
1
1
0
                                                         1
0
1
0
1
      10111000001
                        1
      0 1 0
      [[143 0]
[ 0 132]]
    Classification Report for 'relu':
                         recall f1-score
                precision
                                           support
₹
                            1.00
             0
                    1.00
                                     1.00
                                               143
             1
                    1.00
                                     1.00
                                               132
                             1.00
                                     1.00
                                               275
       accuracy
                    1.00
                             1.00
                                     1.00
                                               275
      macro avo
                    1.00
                             1.00
                                     1.00
    weighted ava
    Predictions using activation function 'logistic':
    [1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0
     1 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1
     1 1 1 0 1 1 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 1
                                                               0 0 0
     0 1 1 0 0 0 0 0 1 1 1 1 1 1 0]
    Confusion Matrix for 'logistic':
    [[143
     [ 0 132]]
    Classification Report for 'logistic':
                precision
                           recall f1-score
                    1.00
                             1.00
                                     1.00
                    1.00
                             1.00
                                     1.00
                                               132
             1
                                     1.00
                                               275
       accuracy
      macro avg
                    1.00
                             1.00
                                     1.00
                                               275
```

1.00

1.00

weighted avg

1.00

275

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

```
Predictions using activation function 'tanh':
1 1 0 1 1 1 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 1 1 0 0 0 0
        1 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0
 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 0]
Confusion Matrix for 'tanh':
[[143 0]
 [ 0 132]]
Classification Report for 'tanh':
         precision
                recall f1-score
                           support
                  1.00
       0
            1.00
                        1.00
                               143
       1
            1.00
                  1.00
                        1.00
                               132
                        1.00
                               275
  accuracy
            1.00
                  1.00
  macro avo
                        1.00
                               275
            1.00
                  1.00
                        1.00
                               275
weighted avg
Predictions using activation function 'identity':
[1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1
 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0
 0\;1\;1\;0\;0\;1\;0\;0\;1\;1\;0\;1\;0\;0\;0\;1\;1\;1\;1\;0\;1\;1\;1\;0\;0\;1\;1\;0\;1\;0\;0\;0\;1\;0\;0\;1\;1
 Confusion Matrix for 'identity':
[[141 2]
[ 0 132]]
Classification Report for 'identity':
        precision
                     f1-score
                  0.99
            0.99
                  1.00
                        0.99
                               132
                               275
  accuracy
                        0.99
            0.99
                  0.99
                        0.99
  macro avq
                               275
weighted avg
            0.99
                  0.99
                        0.99
0 0
1 0
0 1
0 0
1 1
0 0
                                  1 1
1 1
0 1
0 1
0 0
0 0
                              1
0
1
                                    0
0
0
  Confusion Matrix for 'relu':
[[239 0]
[ 0 173]]
```

```
Classification Report for 'relu':
                                  recall f1-score
                    precision
                                                       support
                                     1.00
                                                1.00
                          1.00
                                                            239
                          1.00
                                                            173
                                     1.00
                                                1.00
         accuracy
    macro avg
weighted avg
                          1.00
                                     1.00
                                                1.00
                                                            412
                                                            412
                          1.00
                                     1.00
                                                1.00
    Predictions using activation function 'logistic'
     [0 0 0 0 0 1 0 0 1
0 1 0 0 1 1 0 1 1
                         0 1 1 1 1
1 0 1 1 0
                                     1 0 0 1 0 0 0 1 1
0 1 1 0 1 1 1 1 1
0 1 0 0 0 1 0 0 1
                                                        0
                                                                      0
0
                                                           1
0
                                                             1
                                                               1
                                1 1 0
1 1 0
1 0 1
                                                    0 1 0
                                                           1
1
0
                              0
                                                         0
                                                                    1
                                                                      0
            0 0
1 1
                1
1
                                           1
1
                                                             0 1 0
                   0
1
                     0
0
                       0
0
                         0
                              1
0
                                       1
0
                                         1
0
                                             0
                                                  1
                                                      1
1
1
1
                                                         0
1
                                                                      0
1
1
0
0
                     0
                            0
                              0
                                1
                                  0
                                     1
                                       1
                                         0
                                           0
                                              0
                                                  0
                                                    1
                                                         0
                                                           1
1
0
                                                                    0
                                    0
          1 0 0 0 1 1 0 0
                                       1
0
                                                    1
0
                                                         1
0
                     0
                       0
                         0 0
                              1
0
                                1
                                  0
                                         1
1
                                           1
                                             0
1
                                                0
                                                  1
0
                                                               0
                                                                  0
                                                                        1
1
0
                                                                               1
1
                                                             0
1
1
                                                                    1
0
1
                     1
                                0
                                  0
                                           0
                                                1
                            0
            1 0 1 1 0 1 0 1
                                                                      1
1
0
                                                                          1 0
          0
                                                           0
                                                               0
                                                                  0
                                                                               0
1
       0 0
                                                           0
                                                               0
    Confusion Matrix for 'logistic':
     [[234
         0 173]]
O
   Classification Report for 'logistic':
                    precision
                                   recall f1-score
                                                         support
<del>_</del>
                 0
                          1.00
                                      0.98
                                                 0.99
                                                              239
                 1
                          0.97
                                      1.00
                                                 0.99
                                                              173
                                                 0.99
                                                              412
        accuracy
    macro avg
weighted avg
                          0.99
                                      0.99
                                                 0.99
                                                              412
                          0.99
                                     0.99
                                                 0.99
                                                              412
                                     1 0 0 1 1
0 1 1 1 1
1 0 1 0 1
                                                                        0 0
                                                                            1 0
                                                                                 1
1
0
                                                                        0
                                                                          0
                                                                             0
0
                                                                                        0
                                                             0
1
1
0
1
                                                                      1
1
0
0
                                                               0
1
0
                                                                 0 1 0
                                                                    0
0
0
                                                                                        0
1
0
                                                                             0
                                                                          0
1
1
              0 0
1 0
                     0
                       0
                                 1
                                   0 0
0 1
0 0
1 1
                   1
0
                          0
1
                            0
0
                               1
0
                                                                 0
0
                                                                    0
1
                                                                             0
0
                                                                               0
0
                                                                                        1
0
                                                               0
1
1
            1 1 1 0 0
0 1 0 1 0
1 0 1 0 0
1 1]
                                                                      1
                        1
0
                          0
0
                            1 0
0 0
                                 1
                                                             0
0
                                                                 0
                                                                   0
1
                                                                        1
                                                                          1
0
                                                                             1
0
                                                                               0
                     0 1 0 0
                              0
                                 a
                                   0
                                                               0
                                                                 0
    Confusion Matrix for 'tanh':
    [[236 3]
[ 0 173]]
0
  Classification Report for 'tanh':
                            recall f1-score
                precision
                                             support
<del>_</del>
                    1.00
                              0.99
                                       0.99
                                                 239
                              1.00
                                       0.99
                                                 173
                     0.98
                                       0.99
                                                 412
       accuracy
                    0.99
                              0.99
                                       0.99
                                                 412
   weighted avg
                    0.99
                              0.99
                                       0.99
   Predictions using activation function 'identity':
   0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 0
                                                    0
                                                    0
    0 0 1 1 11
   Confusion Matrix for 'identity':
   [[233 6]
[ 2 171]]
   Classification Report for 'identity':
                            recall f1-score
                precision
                                             support
                              0.97
                     0.99
                                       0.98
                                                 239
                              0.99
                                       0.98
                                                 173
                                       0.98
                                                 412
       accuracy
   macro avg
weighted avg
                              0.98
                    0.98
                                       0.98
                                                 412
                     0.98
                              0.98
                                       0.98
```

Thus, t successfully im	the python prog	gram to imple I the results h	ement multi- lave been ver	layer percept	ron has been lysed.
j	1				J

Ex no: 6 Date:

A PYTHON PROGRAM TO IMPLEMENT SVM CLASSIFIER MODEL

Aim:

To implement a SVM classifier model using python and determine its accuracy.

Algorithm:

Step 1: Import Necessary Libraries

- Import numpy as np.
- Import pandas as pd.
- Import SVM from sklearn.
- Import matplotlib.pyplot as plt.
- Import seaborn as sns.
- Set the font_scale attribute to 1.2 in seaborn.

Step 2: Load and Display Dataset

- Read the dataset (muffins.csv) using `pd.read_csv()`.
- Display the first five instances using the `head()` function.

Step 3: Plot Initial Data

- Use the `sns.lmplot()` function.
- Set the x and y axes to "Sugar" and "Flour".
- Assign "recipes" to the data parameter.
- Assign "Type" to the hue parameter.
- Set the palette to "Set1".
- Set fit_reg to False.
- Set scatter_kws to {"s": 70}.
- Plot the graph.

Step 4: Prepare Data for SVM

- Extract "Sugar" and "Butter" columns from the recipes dataset and assign to variable `sugar butter`.
- Create a new variable `type_label`.

• For each value in the "Type" column, assign 0 if it is "Muffin" and 1 otherwise.

Step 5: Train SVM Model

- Import the SVC module from the svm library.
- Create an SVC model with kernel type set to linear.
- Fit the model using `sugar_butter` and `type_label` as the parameters.

Step 6: Calculate Decision Boundary

- Use the 'model.coef_' function to get the coefficients of the linear model.
- Assign the coefficients to a list named 'w'.
- Calculate the slope `a` as `w[0] / w[1]`.
- Use `np.linspace()` to generate values from 5 to 30 and assign to variable `xx`.
- Calculate the intercept using the first value of the model intercept and divide by `w[1]`.
- Calculate the decision boundary line `y` as `a * xx -(model.intercept [0] / w[1])`.

Step 7: Calculate Support Vector Boundaries

- Assign the first support vector to variable 'b'.
- Calculate 'yy_down' as `a * xx + (b[1] a * b[0])`.
- Assign the last support vector to variable `b`.
- Calculate 'yy up' using the same method.

Step 8: Plot Decision Boundary

- Use the `sns.lmplot()` function again with the same parameters as in Step 3.
- Plot the decision boundary line `xx` and `yy`.

Step 9: Plot Support Vector Boundaries

- Plot the decision boundary with `xx`, `yy_down`, and `'k--'`.
- Plot the support vector boundaries with `xx`, `yy_up`, and `'k--'`.
- Scatter plot the first and last support vectors.

Step 10: Import Additional Libraries

- Import `confusion_matrix` from `sklearn.metrics`.
- Import `classification report` from `sklearn.metrics`.
- Import `train_test_split` from `sklearn.model_selection`.

Step 11: Split Dataset

- Assign `x_train`, `x_test`, `y_train`, and `y_test` using `train_test_split`.
- Set the test size to 0.2.

Step 12: Train New Model

- Create a new SVC model named `model1`.
- Fit the model using the training data ('x_train' and 'y_train').

Step 13: Make Predictions

- Use the 'predict()' function on 'model1' with 'x_test' as the parameter.
- Assign the predictions to variable `pred`.

Step 14: Evaluate Model

- Display the confusion matrix.
- Display the classification report.

```
import numpy as np
import pandas as pd
from sklearn import svm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

sns.set(font_scale=1.2)

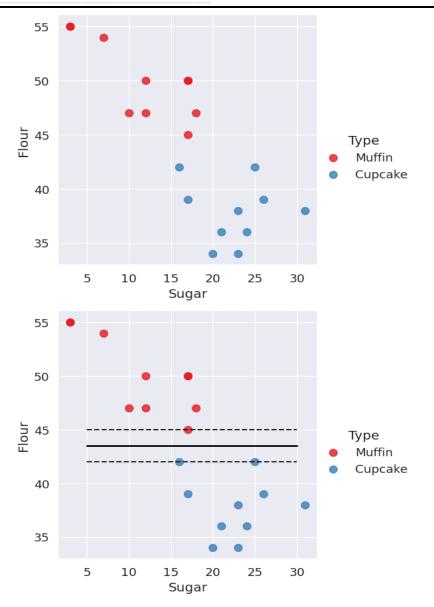
recipes = pd.read_csv('recipes_muffins_cupcakes.csv')
print(recipes.head())
print(recipes.shape)

sns.lmplot(x='Sugar', y='Flour', data=recipes, hue='Type', palette='Set1', fit_reg=False, scatter_kws={"s": 70})
```

```
sugar_butter = recipes[['Sugar', 'Flour']].values
type_label = np.where(recipes['Type'] == 'Muffin', 0, 1)
model = svm.SVC(kernel='linear')
model.fit(sugar butter, type label)
w = model.coef [0]
a = -w[0] / w[1]
xx = np.linspace(5, 30)
yy = a * xx - (model.intercept_[0] / w[1])
b = model.support vectors [0]
yy_down = a * xx + (b[1] - a * b[0])
b = model.support vectors [-1]
yy_up = a * xx + (b[1] - a * b[0])
sns.Implot(x='Sugar', y='Flour', data=recipes, hue='Type', palette='Set1',
fit reg=False, scatter kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(xx, yy down, 'k--')
plt.plot(xx, yy_up, 'k--')
plt.scatter(model.support vectors [:, 0], model.support vectors [:, 1], s=80,
facecolors='none')
x train, x test, y train, y test = train test split(sugar butter, type label,
test size=0.2)
model1 = svm.SVC(kernel='linear')
model1.fit(x train, y train)
pred = model1.predict(x_test)
print(pred)
print(confusion matrix(y test, pred))
print(classification report(y test, pred, zero division=1))
plt.show()
```

Output:

				Butter	Egg	Baking Powder	Vanilla		
<pre>0 Muffin</pre>	55	28	3	7	5	2	0	0	
1 Muffin	47	24	12	6	9	1	0	0	
2 Muffin	47	23	18	6	4	1	0	0	
3 Muffin	45	11	17	17	8	1	0	0	
4 Muffin	50	25	12	6	5	2	1	0	
(20, 9)									
[1 0 1 0]									
[[2 0]									
[0 2]]									
	pre	cision	reca	ll f1-	score	support			
0)	1.00	1.0	00	1.00	2			
1		1.00			1.00				
accuracy	,				1.00	4			
macro avo		1.00	1.0	00	1.00				



Result:

Thus, the python program to implement SVM classifier model has been successfully implemented and the results have been verified and analysed.

Date:

A PYTHON PROGRAM TO IMPLEMENT DECISION TREE

Aim:

To implement a decision tree using a python program for the given dataset and plot the trained decision tree.

Algorithm:

Step 1: Import the Iris Dataset

1. Import 'load iris' from 'sklearn.datasets'.

Step 2: Import Necessary Libraries

- 1. Import numpy as np.
- 2. Import matplotlib.pyplot as plt.
- 3. Import `DecisionTreeClassifier` from `sklearn.tree`.

Step 3: Declare and Initialize Parameters

- 1. Declare and initialize `n_classes = 3`.
- 2. Declare and initialize `plot_colors = "ryb"`.
- 3. Declare and initialize `plot_step = 0.02`.

Step 4: Prepare Data for Model Training

- 1. Load the iris dataset using 'load iris()'.
- 2. Assign the dataset's data to variable `X`.
- 3. Assign the dataset's target to variable 'Y'.

Step 5: Train the Model

- 1. Create an instance of `DecisionTreeClassifier`.
- 2. Fit the classifier using `clf.fit(X, Y)`.

Step 6: Initialize Pair Index and Plot Graph

1. Loop through each pair of features using `for pairidx, pair in enumerate(combinations (range(X.shape[1]), 2)):`

- 2. Inside the loop, assign `X` with the selected pair of features (e.g., `X = iris.data[:, pair]`).
- 3. Assign `Y` with the target list (e.g., `Y = iris.target`).

Step 7: Assign Axis Limits

- 1. Inside the loop, assign `x_min` with the minimum value of the selected feature minus 1 (e.g., `x_min, x_max = X[:, 0].min() 1, X[:, 0].max() + 1`).
- 2. Assign `x_max` with the maximum value of the selected feature plus 1.
- 3. Assign `y_min` with the minimum value of the second selected feature minus 1 (e.g., `y_min, y_max = X[:, 1].min() 1, X[:, 1].max() + 1`).
- 4. Assign 'y_max' with the maximum value of the second selected feature plus 1.

Step 8: Create Meshgrid

- 1. Use `np.meshgrid` to create a grid of values from `x_min` to `x_max` and `y_min` to `y_max` with steps of `plot_step`.
- 2. Assign the results to variables `xx` and `yy`.

Step 9: Plot Graph with Tight Layout

- 1. Use `plt.tight layout()` to adjust the layout of the plots.
- 2. Set `h pad=0.5`, `w pad=0.5`, and `pad=2.5`.

Step 10: Predict and Reshape

- 1. Use the classifier to predict on the meshgrid (e.g., `Z = clf.predict(np.c [xx.ravel(), yy.ravel()])`).
- 2. Reshape 'Z' to the shape of 'xx'.

Step 11: Plot Decision Boundary

1. Use `plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)` to plot the decision boundary with the "RdYlBu" color scheme.

Step 12: Plot Feature Pairs

1. Inside the loop, label the x-axis and y-axis with the feature names (e.g., `plt.xlabel(iris.feature_names[pair[0]])` and `plt.ylabel(iris.feature_names[pair[1]])`).

Step 13: Plot Training Points

1. Use `plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.RdYlBu, edgecolor='k', s=15)` to plot the training points with the "RdYlBu" color scheme, black edge color, and size 15.

Step 14: Plot Final Decision Tree

- 1. Set the title of the plot to "Decision tree trained on all the iris features" (e.g., `plt.title("Decision tree trained on all the iris features")`).
- 2. Display the plot using `plt.show()`.

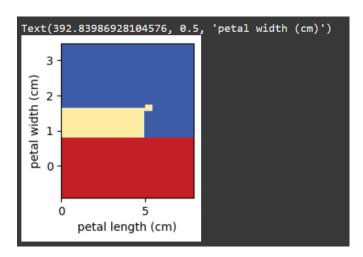
```
from sklearn.datasets import load_iris
iris = load_iris()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
# Parameters
```

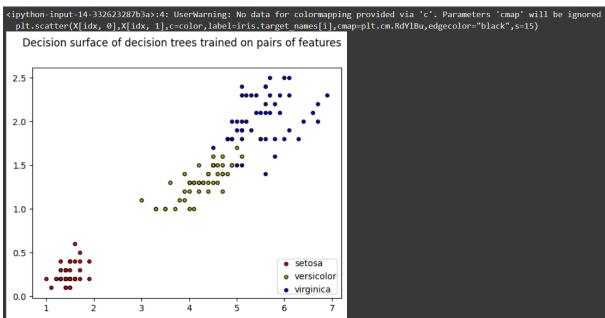
```
# Parameters
n_classes = 3
plot_colors = "ryb"
plot_step = 0.02
for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3], [1, 2], [1, 3], [2, 3]]):
  # We only take the two corresponding features
  X = iris.data[:, pair]
  y = iris.target

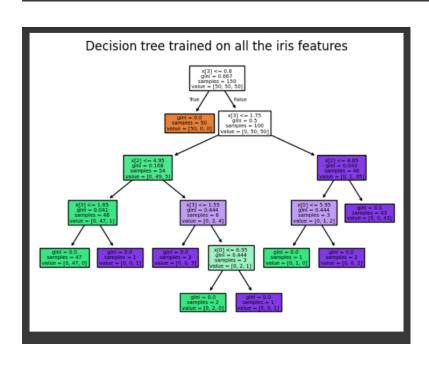
# Train
clf = DecisionTreeClassifier().fit(X, y)
```

```
# Plot the decision boundary
plt.subplot(2, 3, pairidx + 1)
x min, x max = X[:, 0].min() - 1, X[:, 0].max() + 1
y min, y max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x min, x max, plot step),
np.arange(y min, y max, plot step))
plt.tight layout(h pad=0.5, w pad=0.5, pad=2.5)
Z = clf.predict(np.c [xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)
plt.xlabel(iris.feature names[pair[0]])
plt.ylabel(iris.feature names[pair[1]])
# Plot the training points
for i, color in zip(range(n_classes), plot_colors):
 idx = np.where(y == i)
 plt.scatter(X[idx, 0],X[idx,
1],c=color,label=iris.target names[i],cmap=plt.cm.RdYlBu,edgecolor=
"black",s=15)
plt.suptitle("Decision surface of decision trees trained on pairs of
features")
plt.legend(loc="lower right", borderpad=0, handletextpad=0)
plt.axis("tight")
plt.show()
from sklearn.tree import plot tree
plt.figure()
clf = DecisionTreeClassifier().fit(iris.data,iris.target)
plot tree(clf, filled=True)
plt.title("Decision tree trained on all the iris features")
plt.show()
```

OUTPUT:







Result: Thus, the python program to implement Decision Tree has been successfully implemented and the results have been verified and analysed.

Date:

A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING

Aim:

To implement a python program for Ada Boosting.

Algorithm:

Step 1: Import Necessary Libraries

Import numpy as np.

Import pandas as pd.

Import DecisionTreeClassifier from sklearn.tree.

Import train test split from sklearn.model selection.

Import accuracy score from sklearn.metrics.

Step 2: Load and Prepare Data

Load your dataset using pd.read_csv() (e.g., df = pd.read_csv('data.csv')).

Separate features (X) and target (y).

Split the dataset into training and testing sets using train test split().

Step 3: Initialize Parameters

Set the number of weak classifiers n estimators.

Initialize an array weights for instance weights, setting each weight to 1 / number_of_samples.

Step 4: Train Weak Classifiers

Loop for n_estimators iterations:

Train a weak classifier using DecisionTreeClassifier(max_depth=1) on the training data weighted by weights.

Predict the target values using the trained weak classifier.

Calculate the error rate err as the sum of weights of misclassified samples divided by the sum of all weights.

Compute the classifier's weight alpha using 0.5 * np.log((1 - err) / err).

Update the weights: multiply the weights of misclassified samples by np.exp(alpha) and the weights of correctly classified samples by np.exp(-alpha).

Normalize the weights so that they sum to 1.

Append the trained classifier and its weight to lists classifiers and alphas.

Step 5: Make Predictions

For each sample in the testing set:

Initialize a prediction score to 0.

For each trained classifier and its weight:

Add the classifier's prediction (multiplied by its weight) to the prediction score.

Take the sign of the prediction score as the final prediction.

Step 6: Evaluate the Model

Compute the accuracy of the AdaBoost model on the testing set using accuracy score().

Step 7: Output Results

Print or plot the final accuracy and possibly other evaluation metrics.

Code:

```
import pandas as pd
import numpy as np
from mlxtend.plotting import plot_decision_regions
df = pd.DataFrame()
df['X1']=[1,2,3,4,5,6,6,7,9,9]
df['X2']=[5,3,6,8,1,9,5,8,9,2]
df['label']=[1,1,0,1,0,1,0,1,0,0]
import seaborn as sns
sns.scatterplot(x=df['X1'],y=df['X2'],hue=df['label'])
df['weights']=1/df.shape[0]
```

from sklearn.tree import DecisionTreeClassifier

```
dt1 = DecisionTreeClassifier(max_depth=1)
x = df.iloc[:,0:2].values
y = df.iloc[:,2].values
# Step 2 - Train 1st Model
dt1.fit(x,y)
from sklearn.tree import plot_tree
```

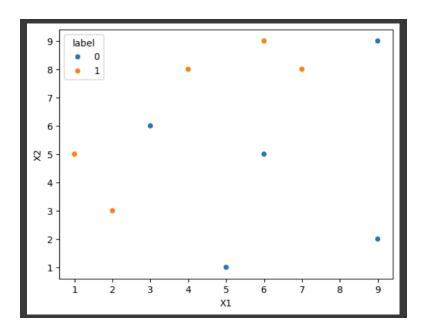
```
plot decision_regions (x,y,clf=dt1, legend=2)
df['y pred'] = dt1.predict(x)
def calculate model weight(error):
 return 0.5*np.log((1-error)/(error))
# Step - 3 Calculate model weight
alpha1 = calculate model weight(0.3)
alpha1
# Step -4 Update weights
def update row weights(row,alpha=0.423):
  if row['label'] == row['y pred']:
    return row['weights']* np.exp(-alpha)
  else:
    return row['weights']* np.exp(alpha)
df['updated weights'] = df.apply(update row weights,axis=1)
df['normalized weights'] = df['updated weights'] /
df['updated weights'].sum() # Calculating normalized weights by
dividing updated weights by sum of all updated weights
df['normalized weights'].sum()
df['cumsum_upper'] = np.cumsum(df['normalized weights'])
df['cumsum lower']=df['cumsum upper'] - df['normalized weights']
df[['X1','X2','label','weights','y pred','updated weights','cumsum low
er','cumsum upper']]
def create new dataset(df):
  indices=[]
  for i in range(df.shape[0]):
    a = np.random.random()
    for index,row in df.iterrows():
       if row['cumsum upper']>a and a>row['cumsum lower']:
         indices.append(index)
  return indices
```

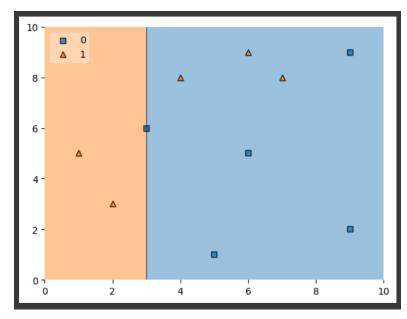
```
index values = create new dataset(df)
index values
second df = df.iloc[index values, [0,1,2,3]]
second df
dt2 = DecisionTreeClassifier(max_depth=1)
x = second df.iloc[:,0:2].values
y = second df.iloc[:,2].values
dt2.fit(x,y)
plot tree(dt2)
plot decision regions(x, y, clf=dt2, legend=2)
second df['y pred'] = dt2.predict(x)
second df
alpha2 = calculate model weight(0.1)
alpha2
# Step 4 - Update weights
def update row weights(row,alpha=1.09):
  if row['label'] == row['y_pred']:
    return row['weights'] * np.exp(-alpha)
  else:
    return row['weights'] * np.exp(alpha)
second df['updated weights'] =
second_df.apply(update_row_weights,axis=1)
second df
second df['normalized weights'] = second df['updated weights'] /
second df['updated weights'].sum()
second df['normalized weights'].sum()
```

```
second df['cumsum upper'] =
np.cumsum(second df['normalized weights'])
second df['cumsum lower'] = second df['cumsum upper'] -
second_df['normalized weights']
second df[['X1','X2','label','weights','y_pred','normalized_weights','cu
msum lower', 'cumsum upper']]
alpha3 = calculate_model_weight(0.7)
alpha3
from sklearn.tree import DecisionTreeClassifier
print(alpha1,alpha2,alpha3)
dt3 = DecisionTreeClassifier(max_depth=2)
# Fit dt3 before making predictions
dt3.fit(x, y) # Assuming 'x' and 'y' are your training data from
previous cells.
query = np.array([1,5]).reshape(1,2)
dt1.predict(query)
dt2.predict(query)
dt3.predict(query)
alpha1*1 + alpha2*(1) + alpha3*(1)
np.sign(1.09)
query = np.array([9,9]).reshape(1,2)
dt1.predict(query)
dt2.predict(query)
dt3.predict(query)
alpha1*(1) + alpha2*(-1) + alpha3*(-1)
```

np.sign(-0.25)

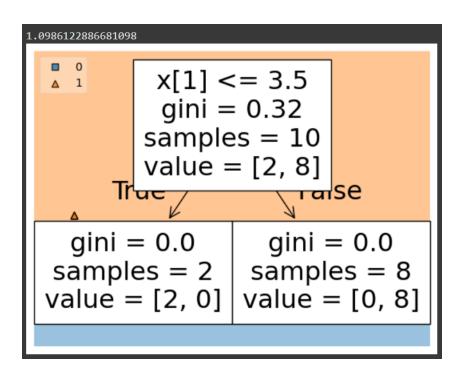
Output:





	X1	Х2	label	weights	y_pred	updated_weights	cumsum_lower	cumsum_upper
0	1	5	1	0.1	1	0.065508	0.000000	0.071475
1	2	3	1	0.1	1	0.065508	0.071475	0.142950
2	3	6	0	0.1	0	0.065508	0.142950	0.214425
3	4	8	1	0.1	0	0.152653	0.214425	0.380983
4	5	1	0	0.1	0	0.065508	0.380983	0.452458
5	6	9	1	0.1	0	0.152653	0.452458	0.619017
6	6	5	0	0.1	0	0.065508	0.619017	0.690492
7	7	8	1	0.1	0	0.152653	0.690492	0.857050
8	9	9	0	0.1	0	0.065508	0.857050	0.928525
9	9	2	0	0.1	0	0.065508	0.928525	1.000000

	X1	X2	label	weights
5	6	9	1	0.1
0	1	5	1	0.1
0	1	5	1	0.1
9	9	2	0	0.1
5	6	9	1	0.1
9	9	2	0	0.1
3	4	8	1	0.1
0	1	5	1	0.1
3	4	8	1	0.1
0	1	5	1	0.1



0.999999999999999

-0.4236489301936017

0.42364893019360184 1.0986122886681098 -0.4236489301936017
-1.0

Result:

Thus the python program to implement ADA Boosting has been successfully implemented and the results have been verified and analyzed.

Date:

A PYTHON PROGRAM TO IMPLEMENT KNN MODEL

Aim:

To implement a python program using a KNN Algorithm in a model.

Algorithm:

1. Import Necessary Libraries

- Import necessary libraries: pandas, numpy, train_test_split from sklearn.model_selection, StandardScaler from sklearn.preprocessing, KNeighborsClassifier from sklearn.neighbors, and classification_report and confusion_matrix from sklearn.metrics.
- 2. Load and Explore the Dataset
 - Load the dataset using pandas.
 - Display the first few rows of the dataset using df.head().
 - Display the dimensions of the dataset using df.shape().
 - Display the descriptive statistics of the dataset using df.describe().
- 3. Preprocess the Data
 - Separate the features (X) and the target variable (y).
 - Split the data into training and testing sets using train_test_split.
 - Standardize the features using StandardScaler.
- 4. Train the KNN Model
 - Create an instance of KNeighborsClassifier with a specified number of neighbors (k).
 - For each data point, calculate the Euclidean distance to all other data points.
 - Select the K nearest neighbors based on the calculated Euclidean distances.
 - Among the K nearest neighbors, count the number of data points in each category.
 - Assign the new data point to the category for which the number of neighbors is maximum.

- 5. Make Predictions
 - Use the trained model to make predictions on the test data.
 - Evaluate the Model
 - Generate the confusion matrix and classification report using the actual and predicted values.
 - Print the confusion matrix and classification report.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read csv('../input/mall-customers/Mall Customers.csv')
X = dataset.iloc[:,[3,4]].values
print(dataset)
from sklearn.cluster import KMeans
wcss =[]
for i in range (1,11):
  kmeans = KMeans(n clusters = i, init = 'k-means++', max iter = 300,
n init = 10, random state = 0)
  kmeans.fit(X)
  wcss.append(kmeans.inertia)
# Plot the graph to visualize the Elbow Method to find the optimal
number of cluster
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans=KMeans(n clusters= 5, init = 'k-means++', max iter = 300,
n init = 10, random state = 0)
```

```
y kmeans = kmeans.fit predict(X)
y_kmeans
type(y kmeans)
y kmeans
plt.scatter(X[y \text{ kmeans} == 0, 0], X[y \text{ kmeans} == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y \text{ kmeans} == 1, 0], X[y \text{ kmeans} == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y \text{ kmeans} == 2, 0], X[y \text{ kmeans} == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y \text{ kmeans} == 3, 0], X[y \text{ kmeans} == 3, 1], s = 100, c =
'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster centers [:, 0], kmeans.cluster centers [:,
1], s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
plt.scatter(X[y \text{ kmeans} == 0, 0], X[y \text{ kmeans} == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y \text{ kmeans} == 1, 0], X[y \text{ kmeans} == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
```

```
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()
```

Output:-

Result: -

Thus the python program to implement KNN model has been successfully implemented and the results have been verified and analyzed.

Date:

A PYTHON PROGRAM TO IMPLEMENT DIMENSIONALITY REDUCTION USING PCA

Aim:

To implement Dimensionality Reduction using PCA in a python program.

Algorithm:

Step 1: Import Libraries

Import necessary libraries, including pandas, numpy, matplotlib.pyplot, and sklearn.decomposition.PCA.

Step 2: Load the Dataset (iris dataset)

Load your dataset into a pandas DataFrame.

Step 3: Standardize the Data

Standardize the features of the dataset using StandardScaler from sklearn.preprocessing.

Step 4: Apply PCA

- Create an instance of PCA with the desired number of components.
- Fit PCA to the standardized data.
- Transform the data to its principal components using transform.

Step 5: Explained Variance Ratio

- Calculate the explained variance ratio for each principal component.
- Plot a scree plot to visualize the explained variance ratio.

Step 6: Choose the Number of Components

Based on the scree plot, choose the number of principal components that explain a significant amount of variance.

Step 7: Apply PCA with Chosen Components

Apply PCA again with the chosen number of components.

Step 8: Visualize the Reduced Data

- Transform the original data to the reduced dimension using the fitted PCA.
- Visualize the reduced data using a scatter plot.

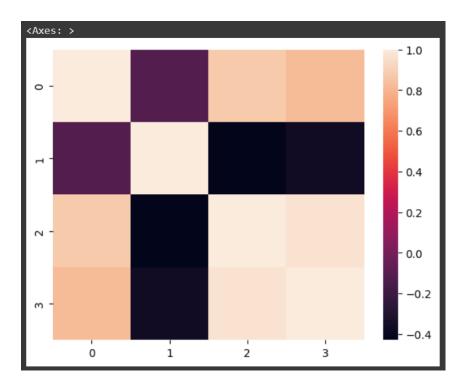
Step 9: Interpretation

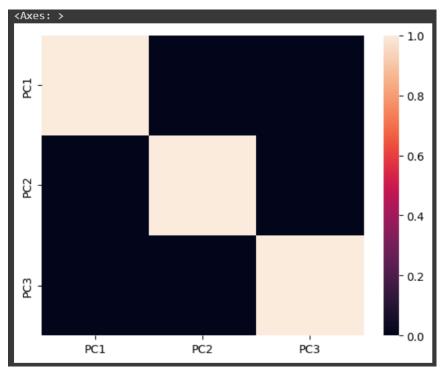
Interpret the results, considering the trade-offs between dimensionality reduction and information loss.

```
from sklearn import datasets
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns
iris = datasets.load iris()
df = pd.DataFrame(iris['data'], columns = iris['feature names'])
df.head()
scalar = StandardScaler()
scaled data = pd.DataFrame(scalar.fit transform(df)) #scaling the
data
scaled data
sns.heatmap(scaled data.corr())
pca = PCA(n components = 3)
pca.fit(scaled data)
data pca = pca.transform(scaled data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
data pca.head()
sns.heatmap(data pca.corr())
```

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2





Result: Thus the python program to implement Dimensionality Reduction using PCA has been successfully implemented and the results have been verified and analyzed.