RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

AI23331 – FUNDAMENTALS OF MACHINE LEARNING

Laboratory Record Notebook

Name: **Sangamithra V**

Year / Branch /Section: **2nd year / B.Tech (AI&DS- 'C')**

University Register No: **2116231801147**

College Roll No: **231801147**

Semester: **3rd Semester**

Academic Year: **2024 - 2025**

AI23331 – FUNDAMENTALS OF MACHINE LEARNING

NAME	**Sangamithra V**
ROLL NO.	**2116231801147**
DEPT	**AIDS**
SEC	**‘C’**

RAJALAKSHMI
ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

NAME……**Sangamithra V**……

ACADEMIC YEAR..**2024-2025**...SEMESTER…**III**……..BRANCH……**AIDS-'C'**……

UNIVERSITY REGISTER No. **2116231801147**

Certified that this is the bonafide record of work done by the above students in the **AI23331 – FUNDAMENTALS OF MACHINE LEARNING** during the year 2024 - 2025.

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on 26/11/2024

Internal Examiner **External Examiner**

INDEX

	List of Experiments	DATE
1	Univariate, Bivariate, and Multivariate Regression	**14/8/2024**
2	Simple Linear Regression Using Least Square Method	**21/8/2024**
3	Logistic Regression Model	**28/8/2024**
4	Single Layer Perceptron	**4/9/2024**
5	Multi-Layer Perceptron with Backpropagation	**11/9/2024**
6	Face Recognition Using SVM Classifier	**18/9/2024**
7	Decision Tree Implementation	**9/10/2024**
8	Boosting Algorithm Implementation	**16/10/2024**
9	K-Nearest Neighbors (KNN) and K-Means Clustering	**22/10/2024**
10	Dimensionality Reduction Using Principal Component Analysis (PCA)	**28/10/2024**
11	Mini Project: Developing a Simple Application Using TensorFlow/Keras	

Ex No: 1
Date:14/8/2024

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.

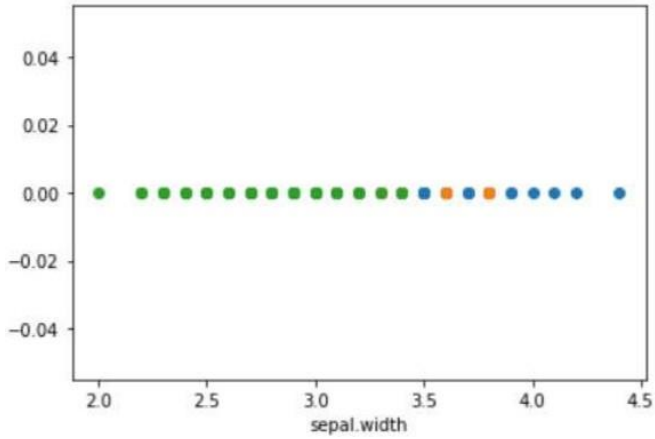
PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
df = pd.read_csv('../input/iris-dataset/iris.csv')
df.head(150)
df.shape
```

(150,5)

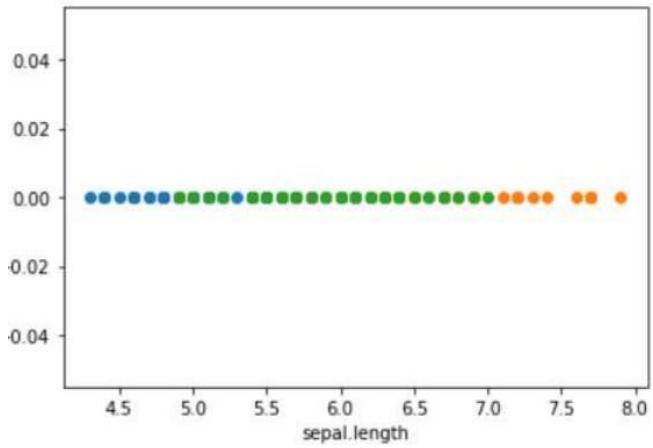
#univariate for sepal width

```
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica']
df_Versicolor=df.loc[df['variety']=='Versicolor']
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.width']))
plt.xlabel('sepal.width')
plt.show()
```



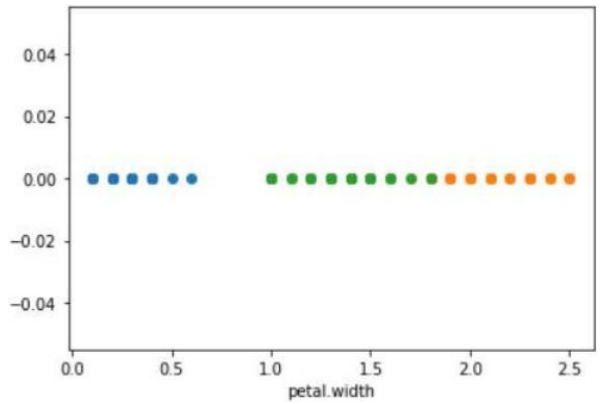
#univariate for sepal length

```
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica']
df_Versicolor=df.loc[df['variety']=='Versicolor']
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.length']))
plt.xlabel('sepal.length')
plt.show()
```



#univariate for petal width

```
df.loc[df['variety']=='Setosa']
df_Setosa=df.loc[df['variety']=='Setosa']
df_Virginica=df.loc[df['variety']=='Virginica']
df_Versicolor=df.loc[df['variety']=='Versicolor']
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']))
plt.xlabel('petal.width')
plt.show()
```



#univariate for petal length

df.loc[df['variety']=='Setosa']

df_Setosa=df.loc[df['variety']=='Setosa']

df_Virginica=df.loc[df['variety']=='Virginica']

df_Versicolor=df.loc[df['variety']=='Versicolor']

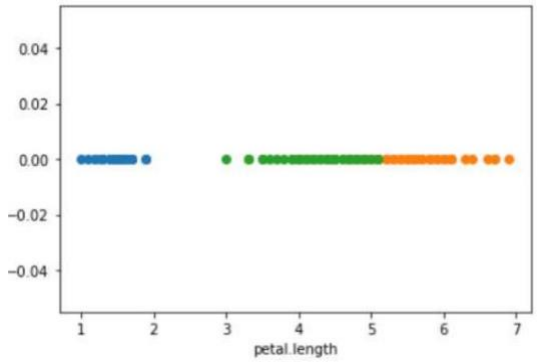
plt.scatter(df_Setosa['petal.length'],np.zeros_like(df_Setosa['petal.length']))

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.length']))

plt.xlabel('petal.length')

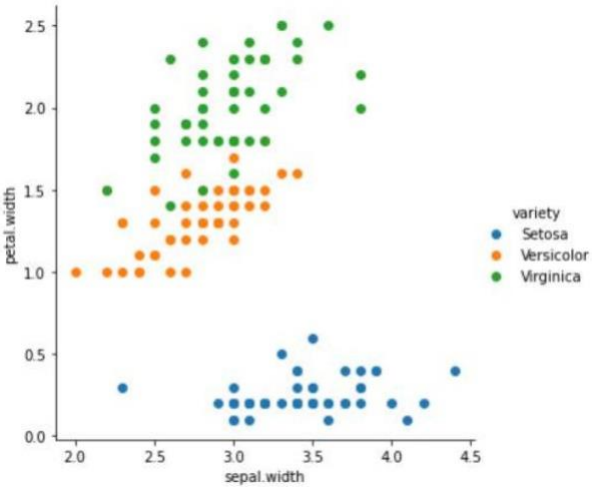
plt.show()



Q

#bivariate sepal.width vs petal.width

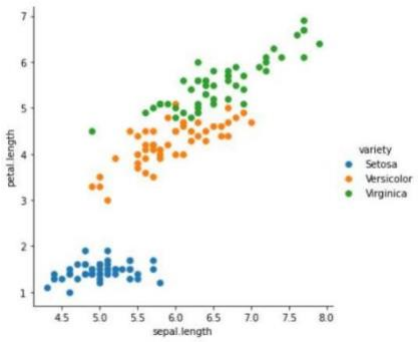
sns.FacetGrid(df,hue='variety',size=5).map(plt.scatter,"sepal.width","petal.width").add_legend();



plt.show()

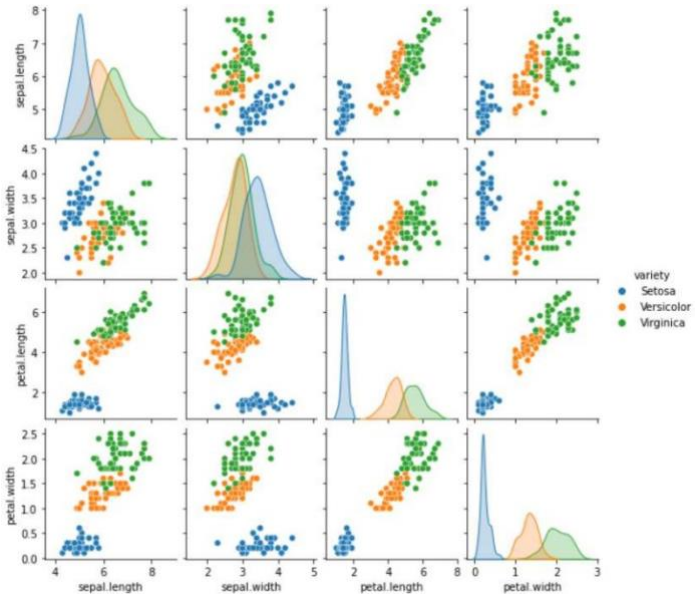
#bivariate sepal.length vs petal.length

sns.FacetGrid(df,hue='variety',size=5).map(plt.scatter,"sepal.length","petal.length").add_legend();
plt.show()



#multivariate all the features

sns.pairplot(df,hue="variety",size=2)



RESULT: -

Thus, the python program to implement univariate, bivariate and multivariate regression features for the given iris dataset is analyzed and the features are plotted using scatter plot

Ex No: 2

Date:21/8/2024

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

Aim*:*

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:

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

- Calculate the intercept (b) using the formula: $b = \bar{y} - m\bar{X}$

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: $R^2 = 1 - \frac{RSS}{TSS}$

Step 9: Display the results:

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

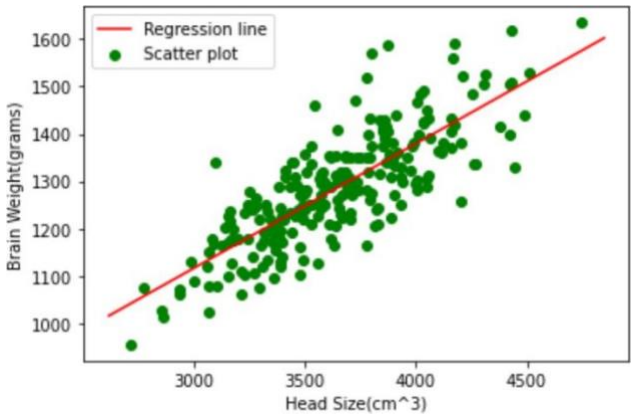
Step 10: Complete the program:

- Combine all the steps into a Python program.
- Run the program to perform simple linear regression on the dataset.

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
data = pd.read_csv('headbrain.csv')
x, y = np.array(list(data['Head Size(cm^3)'])), np.array(list(data['Brain Weight(grams)']))
print(x[:5], y[:5])
[4512 3738 4261 3777 4177] [1530 1297 1335 1282 1590]
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()
```




```
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)
```

In-built Package

```
from sklearn.linear_model import LinearRegression
x = x.reshape((len(x),1))
```

```
reg=LinearRegression()
reg=reg.fit(x, y)
print(reg.score(x, y))
```

RESULT:

Thus, the python program to implement simple linear regression using least square method for the given head brain dataset is analyzed and the linear regression line is constructed successfully

Ex no: 3
Date:28/8/2
024

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).

PROGRAM :

```
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('../input/suvcars/suv_data.csv')
print(data.head())
```

```
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

```
x = data.iloc[:, [2, 3]].values
y = data.iloc[:, 4].values
In-built Function
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,:])
```

```
[[-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]]
```

```
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)

[0000000101000000000100101000000000001001]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print ("Confusion Matrix : \n", cm)

    Confusion Matrix :
     [[31  1]
     [ 1  7]]

from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))

 Accuracy :  0.95
```

User Defined function

```
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])
```

```
(-1.017692393473028, 0.5361288690822568)
```

```
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)
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)
```

```
0.7583333333333334
0.823529411764706
Accuracy is :  0.925
```

RESULT:-

Thus, the python program to implement logistic regression for the given suv_cars dataset is analyzed and the logistic regression model is classifies successfully. The performance of the developed model is measured using F1-score and Accuracy

Ex. No.: 4
Date:4/9/2024

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.

PROGRAM

```
import numpy as np
import pandas as pd
input_value=np.array ([[0,0] ,[0,1], [1,1], [1,0]])
input_value.shape
(4,2)
output = np.array([0,0,1,0])
output = output.reshape(4,1)
output.shape
#(4,1)
weights=np.array([[0.1],[0.3]])
weights
#array ([[0.1], [0.3]])
bias = 0.2
def sigmoid_func(x):
return 1/(1+np.exp(-x))
def der(x):
return sigmoid_func(x)*(1 - sigmoid_func(x))
for epochs in range(15000):
input_arr = input_value
```

```
weighted_sum=np.dot(input_arr,weights)+bias
first_output=sigmoid_func(weighted_sum)
error=first_output - output
total_error=np.square(np.subtract(first_output,output)).mean()
first_der=error
second_der=der(first_output)
derivative=first_der*second_der
t_input = input_value.T
final_derivative=np.dot(t_input,derivative)
weights=weights - (0.05 * final_derivative)
for i in derivative:
bias=bias-(0.05*i)
print(weights)
print(bias)
#[16.57299223]
#[16.57299223]]
#[-25.14783487]
pred=np.array([1,0])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.00018876]
pred=np.array([1,1])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.99966403]

pred=np.array([0,0])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
```

```
print(res)
#[1.19793729e-11]
pred=np.array([0,1])
result = np.dot(pred,weights)+bias
res = sigmoid_func(result)
print(res)
#[0.00063036]
```

RESULT:-

Thus, the python program to implement Single Layer Perceptron has been executed successfully.

Ex. No.: 5
Date:11/9/2024

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.

PROGRAM:

```
import pandas as pd
import numpy as np
bnotes = pd.read_csv('../input/banknotes-dataset/bank_note_data.csv')
bnotes.head(10)
```

```
bnotes.shape
```

```
(1372, 5)
```

	Image.Var	Image.Skew	Image.Curt	Entropy	Class
0	3.62160	8.6661	-2.80730	-0.44699	0
1	4.54590	8.1674	-2.45860	-1.46210	0
2	3.86600	-2.6383	1.92420	0.10645	0
3	3.45660	9.5228	-4.01120	-3.59440	0
4	0.32924	-4.4552	4.57180	-0.98880	0
5	4.36840	9.6718	-3.96060	-3.16250	0
6	3.59120	3.0129	0.72888	0.56421	0
7	2.09220	-6.8100	8.46360	-0.60216	0
8	3.20320	5.7588	-0.75345	-0.61251	0
9	1.53560	9.1772	-2.27180	-0.73535	0

```
bnotes.describe(include='all')
```

	Image.Var	Image.Skew	Image.Curt	Entropy	Class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

```
x = bnotes.drop('Class',axis=1)
y = bnotes['Class']
print(x.head(2))
print(y.head(2))
```

```
   Image.Var  Image.Skew  Image.Curt  Entropy
0     3.6216      8.6661     -2.8073 -0.44699
1     4.5459      8.1674     -2.4586 -1.46210
0    0
1    0
Name: Class, dtype: int64
```

```
from sklearn.model_selection import train_test_split
#train_test ratio = 0.2
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
from sklearn.neural_network import MLPClassifier
# activation function : relu
mlp = MLPClassifier(max_iter=500,activation='relu')
mlp.fit(x_train,y_train)
```

```
MLPClassifier(max_iter=500)
```

```
pred = mlp.predict(x_test)
print(pred)
```

```
[0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0
 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0
 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0
 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1
 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1
 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1
 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0]
```

from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(y_test,pred)

array([[153, 0], [0, 122]])

print(classification_report(y_test,pred))

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       153
           1       1.00      1.00      1.00       122

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

 # activation function : logistic

mlp = MLPClassifier(max_iter=500,activation='logistic')

mlp.fit(x_train,y_train)

MLPClassifier(activation='logistic', max_iter=500)

pred = mlp.predict(x_test)

print(pred)

```
[0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0
 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0
 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0
 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1
 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1
 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1
 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0]
```

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
```

```
array([[153,   0],
       [  0, 122]])
```

```
print(classification_report(y_test,pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00       153
               1       1.00      1.00      1.00       122

        accuracy                           1.00       275
       macro avg       1.00      1.00      1.00       275
    weighted avg       1.00      1.00      1.00       275
```

 # activation function : tanh

```
mlp = MLPClassifier(max_iter=500,activation='tanh')
mlp.fit(x_train,y_train)
pred = mlp.predict(x_test)
print(pred)
```

```
[0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 1 1 0
 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 1 0 0 0
 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0
 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1
 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1
 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1
 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 0]
```

```
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y_test,pred)
```

```
array([[153,   0],
       [  0, 122]])
```

```
print(classification_report(y_test,pred))
```

```
                precision    recall  f1-score   support

             0       1.00      1.00      1.00       153
             1       1.00      1.00      1.00       122

      accuracy                           1.00       275
     macro avg       1.00      1.00      1.00       275
  weighted avg       1.00      1.00      1.00       275
```

 # activation function : identity

```
mlp = MLPClassifier(max_iter=500,activation='identity')
mlp.fit(x_train,y_train)
```

```
MLPClassifier(activation='identity', max_iter=500)
```

```
pred = mlp.predict(x_test)
print(pred)
```

```
[0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0
 0 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 0 0
 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 0 0
 0 0 0 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 0 1
 1 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1
 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1
 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0
 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0]
```

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
```

```
array([[150,   3],
       [  3, 119]])
```

```
print(classification_report(y_test,pred))
```

```
                precision    recall  f1-score   support

             0       0.98      0.98      0.98       153
             1       0.98      0.98      0.98       122
```

accuracy			0.98	275
macro avg	0.98	0.98	0.98	275
weighted avg	0.98	0.98	0.98	275

#train_test ratio = 0.3

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=**0.3**)

from sklearn.neural_network import MLPClassifier

 # activation function : relu

mlp = MLPClassifier(max_iter=500,activation='relu')

mlp.fit(x_train,y_train)

MLPClassifier(max_iter=500)
pred = mlp.predict(x_test)

print(pred)

```
[0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1
 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 0 1 0
 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1
 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0
 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1
 0 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1
 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 1 1 0 0]
```

from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(y_test,pred)

```
array([[220,   0],
       [  0, 192]])
```

print(classification_report(y_test,pred))

	precision	recall	f1-score	support
0	1.00	1.00	1.00	220
1	1.00	1.00	1.00	192
accuracy			1.00	412
macro avg	1.00	1.00	1.00	412
weighted avg	1.00	1.00	1.00	412

 # activation function : logistic

```
mlp = MLPClassifier(max_iter=500,activation='logistic')
mlp.fit(x_train,y_train)
MLPClassifier(max_iter=500,activation='logistic')

pred = mlp.predict(x_test)
print(pred)
MLPClassifier(max_iter=500,activation='tanh')
```

```
[0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1
 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 0 1 0
 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1
 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0
 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1
 0 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1
 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 1 1 0 0]
```

```
from sklearn.metrics import classification_report,confusion_matrix
confusion_matrix(y_test,pred)
```

 # activation function : tanh

```
mlp = MLPClassifier(max_iter=500,activation='tanh')
mlp.fit(x_train,y_train)
pred = mlp.predict(x_test)
print(pred)
```

```
[0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1
 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 0 1 0
 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1
 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0
 0 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1
 0 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1
 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 1 1 0 0]
```

from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(y_test,pred)

```
array([[220,   0],
       [  0, 192]])
```

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       220
           1       1.00      1.00      1.00       192

    accuracy                           1.00       412
   macro avg       1.00      1.00      1.00       412
weighted avg       1.00      1.00      1.00       412
```

 # activation function : identity

mlp = MLPClassifier(max_iter=500,activation='identity')

mlp.fit(x_train,y_train)

MLPClassifier(max_iter=500,activation='identity')

pred = mlp.predict(x_test)

print(pred)

```
[0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1
 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0
 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 0 1 1 1 1
 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0
 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1
 0 1 1 0 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1
 0 1 0 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1
 0 1 1 0 0]
```

from sklearn.metrics import classification_report,confusion_matrix

confusion_matrix(y_test,pred)

```
array([[217,   3],
       [  1, 191]])
```

print(classification_report(y_test,pred))

	precision	recall	f1-score	support
0	1.00	0.99	0.99	220
1	0.98	0.99	0.99	192
accuracy			0.99	412
macro avg	0.99	0.99	0.99	412
weighted avg	0.99	0.99	0.99	412

RESULT:

Thus the python program to implement multi layer perceptron with back propagation on the given dataset(banknotes.csv) has been executed successfully and it's results have been analyzed successfully for different activation functions(relu,logistic,tanh,identity) with two different training-testing ratios(0.2 and 0.3)

Ex no: 6
Date:18/9/2
024

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

1. Import numpy as np.
2. Import pandas as pd.
3. Import SVM from sklearn.
4. Import matplotlib.pyplot as plt.
5. Import seaborn as sns.
6. Set the font_scale attribute to 1.2 in seaborn.

Step 2: Load and Display Dataset

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

Step 3: Plot Initial Data

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

Step 4: Prepare Data for SVM

1. Extract "Sugar" and "Butter" columns from the recipes dataset and assign to variable `sugar_butter`.
2. Create a new variable `type_label`.
3. For each value in the "Type" column, assign 0 if it is "Muffin" and 1 otherwise.

Step 5: Train SVM Model

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

Step 6: Calculate Decision Boundary

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

Step 7: Calculate Support Vector Boundaries

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

Step 8: Plot Decision Boundary

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

Step 9: Plot Support Vector Boundaries

1. Plot the decision boundary with `xx`, `yy_down`, and `'k--'`.

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

Step 10: Import Additional Libraries

1. Import `confusion_matrix` from `sklearn.metrics`.
2. Import `classification_report` from `sklearn.metrics`.
3. Import `train_test_split` from `sklearn.model_selection`.

Step 11: Split Dataset

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

Step 12: Train New Model

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

Step 13: Make Predictions

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

Step 14: Evaluate Model

1. Display the confusion matrix.
2. Display the classification report.

PROGRAM:

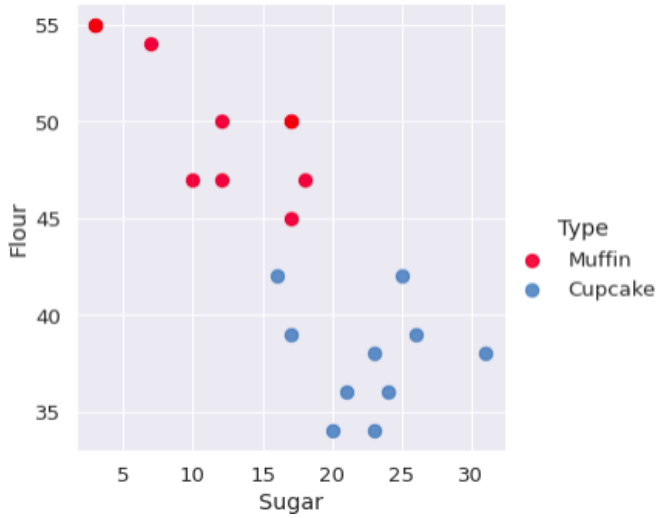
```
import numpy as np
import pandas as pd
from sklearn import svm
import matplotlib.pyplot as plt
```

```
import seaborn as sns; sns.set(font_scale=1.2)
recipes=pd.read_csv('../input/muffins-datset/recipes_muffins_cupcakes.csv')
recipes.head()
recipes.shape
```

```
(20, 9)
```

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

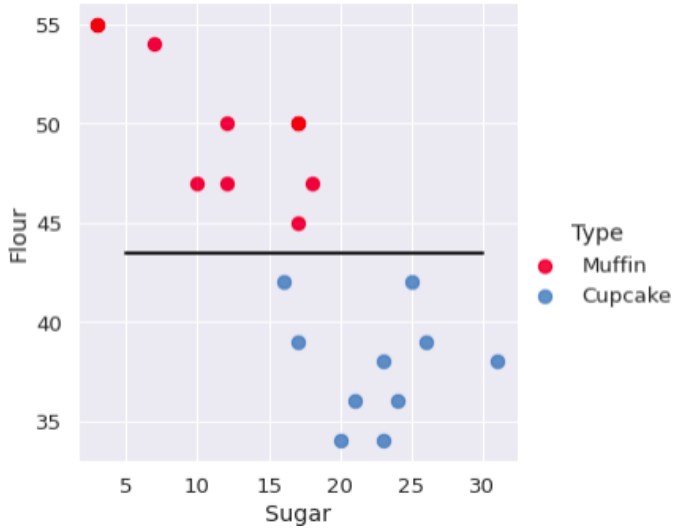
```
<seaborn.axisgrid.FacetGrid at 0x7fca4a9fda90>
```




```
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
SVC(kernel='linear')
w=model.coef_[0] #seperating the hyperplane
a=-w[0]/w[1]
xx=np.linspace(5,30)
yy=a*xx-(model.intercept_[0]/w[1])
b=model.support_vectors_[0] #plot to seperate hyperplane that pass
```

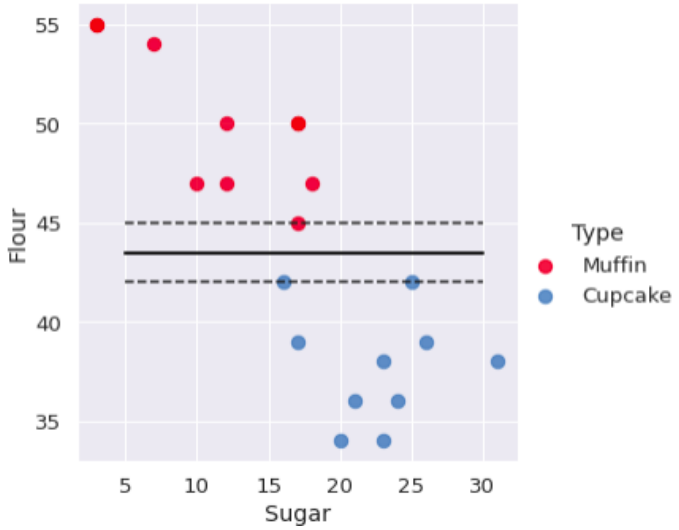
```
yy_down=a*xx+(b[1]-a*b[0])
b=model.support_vectors_[-1]
yy_up=a*xx+(b[1]-a*b[0])
sns.lmplot('Sugar','Flour',data=recipes,hue='Type',palette='Set1',fit_reg=False,sc
atter_kws={"s":70})
plt.plot(xx,yy,linewidth=2,color='black')
```

```
[<matplotlib.lines.Line2D at 0x7fca4a98ba50>]
```




```
scatterkws={"s":70})
 plt.plot(xx,yy,linewidth=2,color='black')
sns.lmplot('Sugar','Flour',data=recipes,hue='Type',palette='Set1',fit_reg=False,sc
atter_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,facecol
or='none')
```

```
<matplotlib.collections.PathCollection at 0x7fca4a88071
```




```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
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)
```

```
[0 0 1 0]
```

```
print(confusion_matrix(y_test,pred))
```

```
[[2 0]
 [1 1]]
```

```
print(classification_report(y_test,pred))
```

	precision	recall	f1-score	support
0	0.67	1.00	0.80	2
1	1.00	0.50	0.67	2
accuracy			0.75	4
macro avg	0.83	0.75	0.73	4
weighted avg	0.83	0.75	0.73	4

RESULT:

Thus the python program to implement SVM classifier model has been executed successfully and the classified output has been analyzed for the given dataset(muffins.csv)

Ex. No.: 7
Date:9/10/2024

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

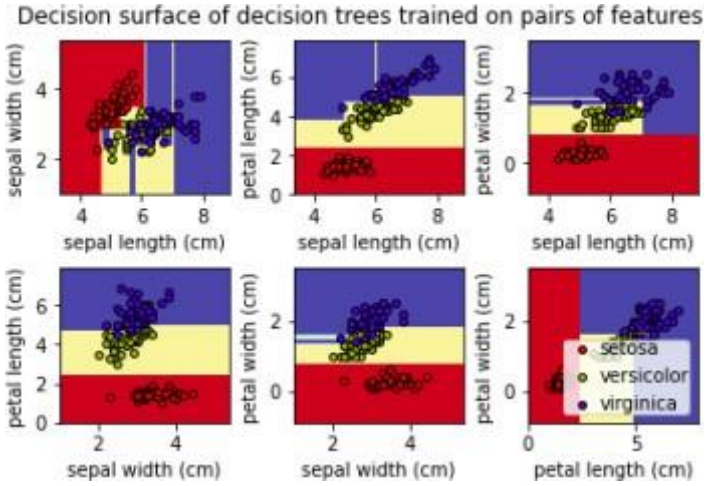
1. Set the title of the plot to "Decision tree trained on all the iris features" (e.g., `plt.title("Decision tree trained on all the iris features")`).
2. Display the plot using `plt.show()`.

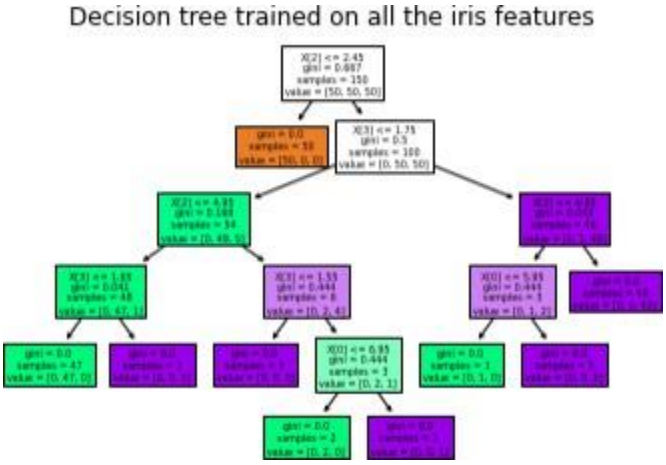
PROGRAM:

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





RESULT:

Thus the python program to implement Decision Tree for the given dataset has been successfully implemented and the results have been verified and analyzed

Ex. No.: 8 a.
Date:16/10/
2024

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.

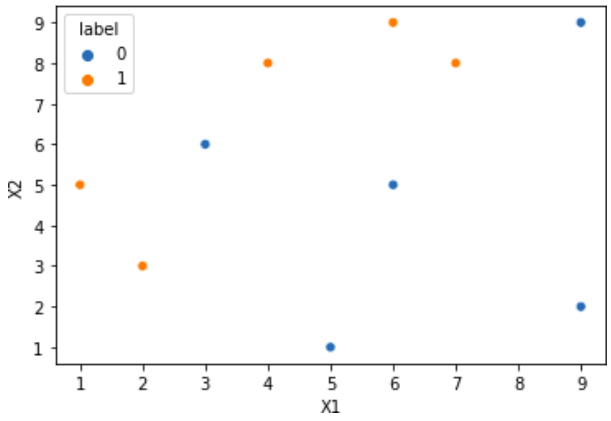
PROGRAM:

```
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]
```

	X1	X2	label
0	1	5	1
1	2	3	1
2	3	6	0
3	4	8	1
4	5	1	0
5	6	9	1
6	6	5	0
7	7	8	1
8	9	9	0
9	9	2	0

```
import seaborn as sns
sns.scatterplot(x=df['X1'],y=df['X2'],hue=df['label'])
```

```
<AxesSubplot:xlabel='X1', ylabel='X2'>
```

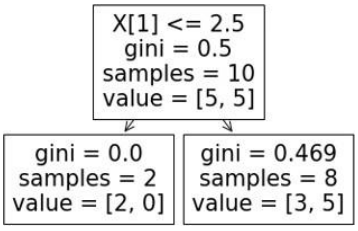



```
df['weights']=1/df.shape[0]
```

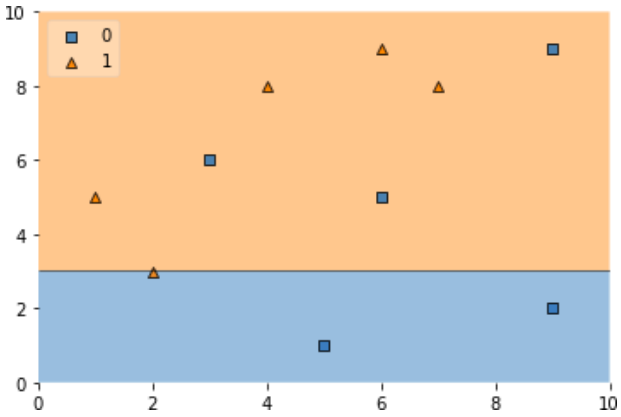
	X1	X2	label	weights
0	1	5	1	0.1
1	2	3	1	0.1
2	3	6	0	0.1
3	4	8	1	0.1
4	5	1	0	0.1
5	6	9	1	0.1
6	6	5	0	0.1
7	7	8	1	0.1
8	9	9	0	0.1
9	9	2	0	0.1

```
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_tree(dt1)
```

```
[Text(0.5, 0.75, 'X[1] <= 2.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.75, 0.25, 'gini = 0.469\nsamples = 8\nvalue = [3, 5]')]
```



plot_decision_regions (x,yclf=dt1, legend=2)



dfl'y pred'] = dt1.predict(x)

	X1	X2	label	weights	y_pred
0	1	5	1	0.1	1
1	2	3	1	0.1	1
2	3	6	0	0.1	1
3	4	8	1	0.1	1
4	5	1	0	0.1	0
5	6	9	1	0.1	1
6	6	5	0	0.1	1
7	7	8	1	0.1	1
8	9	9	0	0.1	1
9	9	2	0	0.1	0

```
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
```

0.42364893019360184

```
# 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)
```

	X1	X2	label	weights	y_pred	updated_weights
0	1	5	1	0.1	1	0.065508
1	2	3	1	0.1	1	0.065508
2	3	6	0	0.1	1	0.152653
3	4	8	1	0.1	1	0.065508
4	5	1	0	0.1	0	0.065508
5	6	9	1	0.1	1	0.065508
6	6	5	0	0.1	1	0.152653
7	7	8	1	0.1	1	0.065508
8	9	9	0	0.1	1	0.152653
9	9	2	0	0.1	0	0.065508

```
df['updated_weights'].sum()
```

```
0.9165153319682015
```

```
df['normalized_weights']=df['updated_weights']/df['updated_weights'].sum()
```

	X1	X2	label	weights	y_pred	updated_weights	normalized_weights
0	1	5	1	0.1	1	0.065508	0.071475
1	2	3	1	0.1	1	0.065508	0.071475
2	3	6	0	0.1	1	0.152653	0.166559
3	4	8	1	0.1	1	0.065508	0.071475
4	5	1	0	0.1	0	0.065508	0.071475
5	6	9	1	0.1	1	0.065508	0.071475
6	6	5	0	0.1	1	0.152653	0.166559
7	7	8	1	0.1	1	0.065508	0.071475
8	9	9	0	0.1	1	0.152653	0.166559
9	9	2	0	0.1	0	0.065508	0.071475

df['normalized_weights'].sum()

```
1.0
```

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_lower','cumsum

	X1	X2	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	1	0.152653	0.142950	0.309508
3	4	8	1	0.1	1	0.065508	0.309508	0.380983
4	5	1	0	0.1	0	0.065508	0.380983	0.452458
5	6	9	1	0.1	1	0.065508	0.452458	0.523933
6	6	5	0	0.1	1	0.152653	0.523933	0.690492
7	7	8	1	0.1	1	0.065508	0.690492	0.761967
8	9	9	0	0.1	1	0.152653	0.761967	0.928525
9	9	2	0	0.1	0	0.065508	0.928525	1.000000

```
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
```

```
[6, 6, 0, 6, 7, 5, 1, 8, 4, 6]
```

```
second_df = df.iloc[index_values,[0,1,2,3]]
second_df
```

	X1	X2	label	weights
6	6	5	0	0.1
6	6	5	0	0.1
0	1	5	1	0.1
6	6	5	0	0.1
7	7	8	1	0.1
5	6	9	1	0.1
1	2	3	1	0.1
8	9	9	0	0.1
4	5	1	0	0.1
6	6	5	0	0.1

```
dt2 = DecisionTreeClassifier(max_depth=1)
```

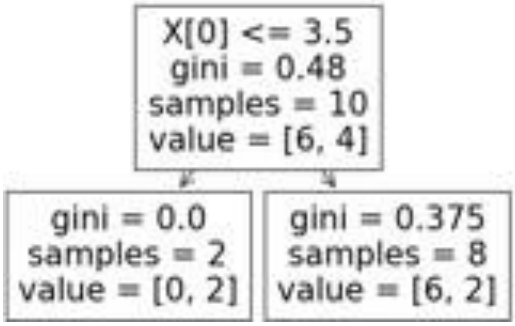
```
x = second_df.iloc[:,0:2].values
y = second_df.iloc[:,2].values
```

```
dt2.fit(x,y)
```

```
DecisionTreeClassifier(max_depth=1)
```

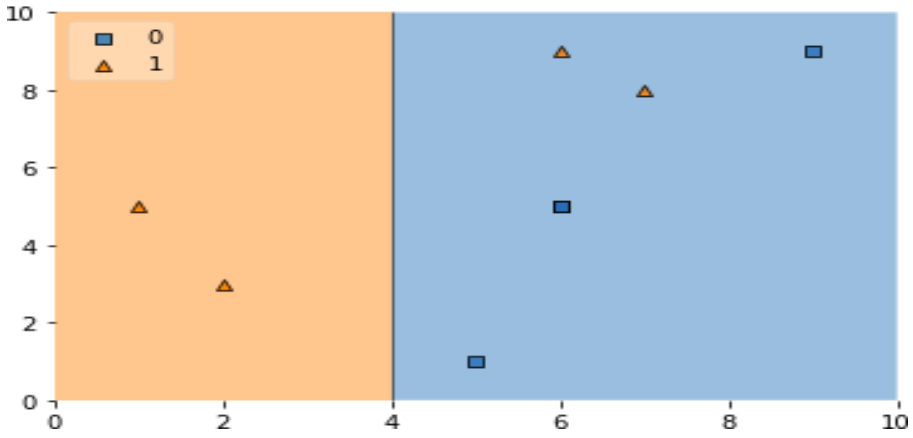
```
plot_tree(dt2)
```

```
[Text(0.5, 0.75, 'X[0] <= 3.5\ngini = 0.48\nsamples = 10\nvalue = [6, 4]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.75, 0.25, 'gini = 0.375\nsamples = 8\nvalue = [6, 2]')]
```




```
plot_decision_regions(x, y, clf=dt2, legend=2)
```

```
<AxesSubplot:>
```



second_df['y_pred'] = dt2.predict(x)

second_df

alpha2 = calculate_model_weight(0.1)

	X1	X2	label	weights	y_pred
6	6	5	0	0.1	0
6	6	5	0	0.1	0
0	1	5	1	0.1	1
6	6	5	0	0.1	0
7	7	8	1	0.1	0
5	6	9	1	0.1	0
1	2	3	1	0.1	1
8	9	9	0	0.1	0
4	5	1	0	0.1	0
6	6	5	0	0.1	0

alpha2

```
1.0986122886681098
```

 # 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['nomalized_weights'].sum()
```

	X1	X2	label	weights	y_pred	updated_weights
6	6	5	0	0.1	0	0.033622
6	6	5	0	0.1	0	0.033622
0	1	5	1	0.1	1	0.033622
6	6	5	0	0.1	0	0.033622
7	7	8	1	0.1	0	0.297427
5	6	9	1	0.1	0	0.297427
1	2	3	1	0.1	1	0.033622
8	9	9	0	0.1	0	0.033622
4	5	1	0	0.1	0	0.033622
6	6	5	0	0.1	0	0.033622

```
second_df['nomalized_weights'].sum()
```

```
0.9999999999999999
```

```
second_df['cumsum_upper'] = np.cumsum(second_df['nomalized_weights'])
```

```
second_df['cumsum_lower'] = second_df['cumsum_upper'] - second_df['nomalized_weights']
second_df[['X1','X2','label','weights','y_pred','nomalized_weights','cumsum_lower','cumsum_
upper']]
```

	X1	X2	label	weights	y_pred	nomalized_weights	cumsum_lower	cumsum_upper
6	6	5	0	0.1	0	0.038922	0.000000	0.038922
6	6	5	0	0.1	0	0.038922	0.038922	0.077843
0	1	5	1	0.1	1	0.038922	0.077843	0.116765
6	6	5	0	0.1	0	0.038922	0.116765	0.155687
7	7	8	1	0.1	0	0.344313	0.155687	0.500000
5	6	9	1	0.1	0	0.344313	0.500000	0.844313
1	2	3	1	0.1	1	0.038922	0.844313	0.883235
8	9	9	0	0.1	0	0.038922	0.883235	0.922157
4	5	1	0	0.1	0	0.038922	0.922157	0.961078
6	6	5	0	0.1	0	0.038922	0.961078	1.000000

```
index_values = create_new_dataset(second_df)
third_df = second_df.iloc[index_values,[0,1,2,3]]
third_df
```

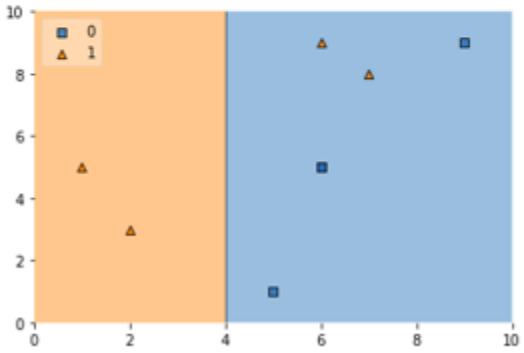
	X1	X2	label	weights
1	2	3	1	0.1
6	6	5	0	0.1
5	6	9	1	0.1
1	2	3	1	0.1
5	6	9	1	0.1
8	9	9	0	0.1
8	9	9	0	0.1
8	9	9	0	0.1
5	6	9	1	0.1
8	9	9	0	0.1

```
y = second_df.iloc[:,2].values
dt3.fit(X,y)
```

```
DecisionTreeClassifier(max_depth=1)
```

```
plot_decision_regions(X, y, clf=dt3, legend=2)
```

```
<AxesSubplot:>
```




```
third_df['y_pred] = dt.predict(X)
```

```
alpha3 = calculate_model_weight(0.7)
alpha3
```

```
-0.4236489301936017
```

```
print(alpha1,alpha2,alpha3)
```

```
0.42364893019360184 1.0986122886681098 -0.4236489301936017
```

```
query = np.array([1,5]).reshape(1,2)
dt1.predict(query)
```

```
array([1])
```

```
dt2.predict(query)
```

```
array([1])
```

dt3.predict(query)

```
array([1])
```

alpha1*1 + alpha2*(1) + alpha3*(1)

```
1.09861228866811
```

np.sign(1.09)

```
1.0
```

query = np.array([9,9]).reshape(1,2)

dt1.predict(query)

```
array([1])
```

dt2.predict(query)

```
array([0])
```

dt3.predict(query)

```
array([0])
```

alpha1*(1) + alpha2*(-1) + alpha3*(-1)

```
-0.2513144282809062
```

np.sign(-0.25)

```
-1.0
```

RESULT:

Thus the python program to implement Adaboosting has been executed successfully and the results have been verified and analyzed.

Ex. No.: 8b
Date:22/10/
2024

A PYTHON PROGRAM TO IMPLEMENT GRADIENT BOOSTING

Aim:

To implement a python program using the gradient boosting model.

Algorithm:

Step 1: Import Necessary Libraries

Import numpy as np.

Import pandas as pd.

Import train_test_split from sklearn.model_selection.

Import DecisionTreeRegressor from sklearn.tree.

Import mean_squared_error from sklearn.metrics.

Step 2: Prepare the Data

Load your dataset into a DataFrame using pd.read_csv('your_dataset.csv').

Split the dataset into features (X) and target (y).

Use train_test_split to split the data into training and testing sets.

Step 3: Initialize Parameters

Set the number of boosting rounds (e.g., n_estimators = 100).

Set the learning rate (e.g., learning_rate = 0.1).

Initialize an empty list to store the weak learners (decision trees).

Initialize an empty list to store the learning rates for each round.

Step 4: Initialize the Base Model

Compute the initial prediction as the mean of the target values (e.g., F0 = np.mean(y_train)).

Initialize the predictions to the base model's prediction (e.g., F = np.full(y_train.shape, F0)).

Step 5: Iterate Over Boosting Rounds

For each boosting round:

Compute the pseudo-residuals (negative gradient of the loss function) (e.g., residuals = y_train - F).

Fit a decision tree to the pseudo-residuals.

Make predictions using the fitted tree (e.g., tree_predictions = tree.predict(X_train)).

Update the predictions by adding the learning rate multiplied by the tree predictions (e.g., F += learning_rate * tree_predictions).

Append the fitted tree and the learning rate to their respective lists.

Step 6: Make Predictions on Test Data

Initialize the test predictions with the base model's prediction (e.g., F_test = np.full(y_test.shape, F0)).

For each fitted tree and its learning rate:

Make predictions on the test data using the fitted tree.

Update the test predictions by adding the learning rate multiplied by the tree predictions.

Step 7: Evaluate the Model

Compute the mean squared error on the training data.

Compute the mean squared error on the test data.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

df = pd.DataFrame()

df['X'] = X.reshape(100)
```

```
df['y'] = y
```

```
df
```

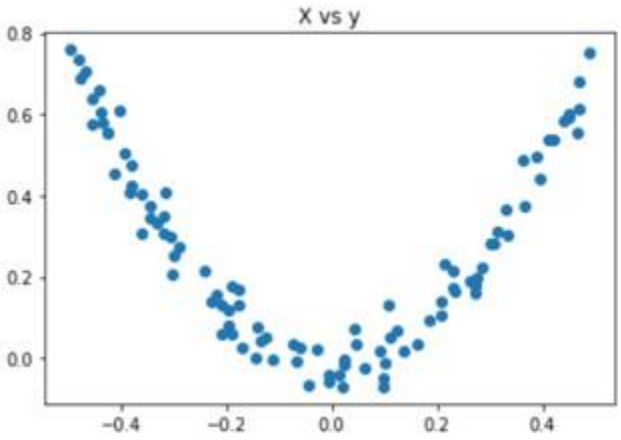
[6]:

	X	y
0	-0.125460	0.051573
1	0.450714	0.594480
2	0.231994	0.166052
3	0.098658	-0.070178
4	-0.343981	0.343986
...	...	...
95	-0.006204	-0.040675
96	0.022733	-0.002305
97	-0.072459	0.032809
98	-0.474581	0.689516
99	-0.392109	0.502607

```
plt.scatter(df['X'],df['y'])
plt.title('X vs y')
```

```
Text(0.5, 1.0, 'X vs y')
```

[9]: `Text(0.5, 1.0, 'X vs y')`




```
df['pred1'] = df['y'].mean()
df
```

[11]:

	X	y	pred1
0	-0.125460	0.051573	0.265458
1	0.450714	0.594480	0.265458
2	0.231994	0.166052	0.265458
3	0.098658	-0.070178	0.265458
4	-0.343981	0.343986	0.265458
...	...	...	...
95	-0.006204	-0.040675	0.265458
96	0.022733	-0.002305	0.265458
97	-0.072459	0.032809	0.265458
98	-0.474581	0.689516	0.265458
99	-0.392109	0.502607	0.265458

100 rows × 3 columns

df['res1'] = df['y'] - df['pred1']

df

[13]:

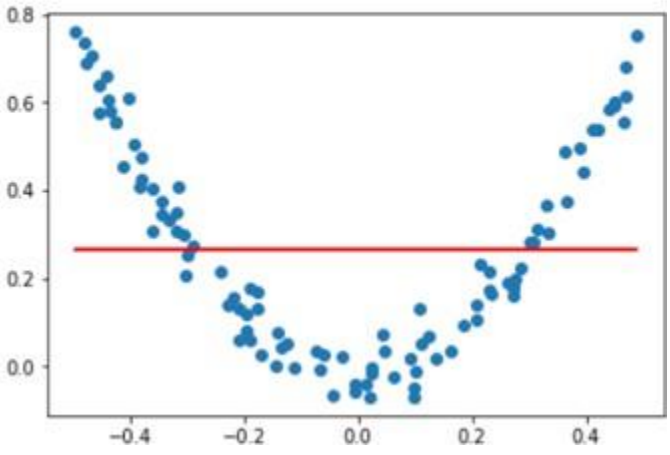
	X	y	pred1	res1
0	-0.125460	0.051573	0.265458	-0.213885
1	0.450714	0.594480	0.265458	0.329021
2	0.231994	0.166052	0.265458	-0.099407
3	0.098658	-0.070178	0.265458	-0.335636
4	-0.343981	0.343986	0.265458	0.078528
...	...	...	...	...
95	-0.006204	-0.040675	0.265458	-0.306133
96	0.022733	-0.002305	0.265458	-0.267763
97	-0.072459	0.032809	0.265458	-0.232650
98	-0.474581	0.689516	0.265458	0.424057
99	-0.392109	0.502607	0.265458	0.237148

100 rows × 4 columns

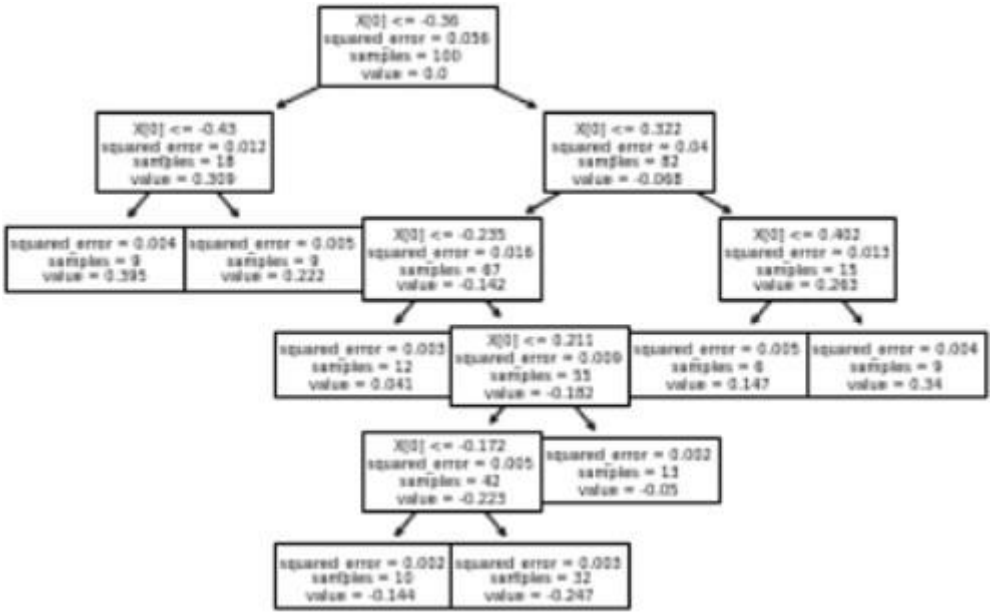
plt.scatter(df['X'],df['y'])

plt.plot(df['X'],df['pred1'],color='red')

[14]: [<matplotlib.lines.Line2D at 0x7f6ef51f7a10>]

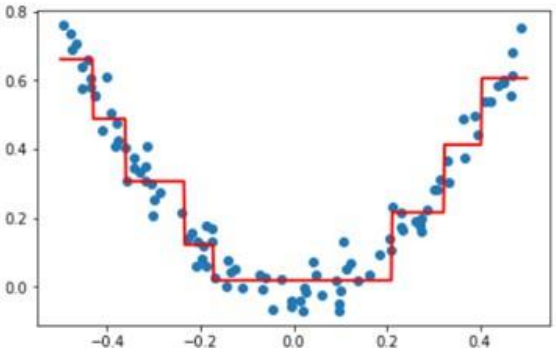



```
from sklearn.tree import DecisionTreeRegressor

tree1 = DecisionTreeRegressor(max_leaf_nodes=8)

tree1.fit(df['X'].values.reshape(100,1),df['res1'].values)

DecisionTreeRegressor(max_leaf_nodes=8)

from sklearn.tree import plot_tree
plot_tree(tree1)
plt.show()
```




```
X_test = np.linspace(-0.5, 0.5, 500)

y_pred = 0.265458 + tree1.predict(X_test.reshape(500, 1))

plt.figure(figsize=(14,4))
plt.subplot(121)
plt.plot(X_test, y_pred, linewidth=2,color='red')
plt.scatter(df['X'],df['y'])
```

```
[21]: <matplotlib.collections.PathCollection at 0x7f6ed205ca50>
```



df['pred2'] = 0.265458 + tree1.predict(df['X'].values.reshape(100,1))

df

92]:

	X	y	pred1	res1	pred2
0	-0.125460	0.051573	0.265458	-0.213885	0.018319
1	0.450714	0.594480	0.265458	0.329021	0.605884
2	0.231994	0.166052	0.265458	-0.099407	0.215784
3	0.098658	-0.070178	0.265458	-0.335636	0.018319
4	-0.343981	0.343986	0.265458	0.078528	0.305964
...	...	...	...	...	...
95	-0.006204	-0.040675	0.265458	-0.306133	0.018319
96	0.022733	-0.002305	0.265458	-0.267763	0.018319
97	-0.072459	0.032809	0.265458	-0.232650	0.018319
98	-0.474581	0.689516	0.265458	0.424057	0.660912
99	-0.392109	0.502607	0.265458	0.237148	0.487796

100 rows × 5 columns

df['res2'] = df['y'] - df['pred2']

df

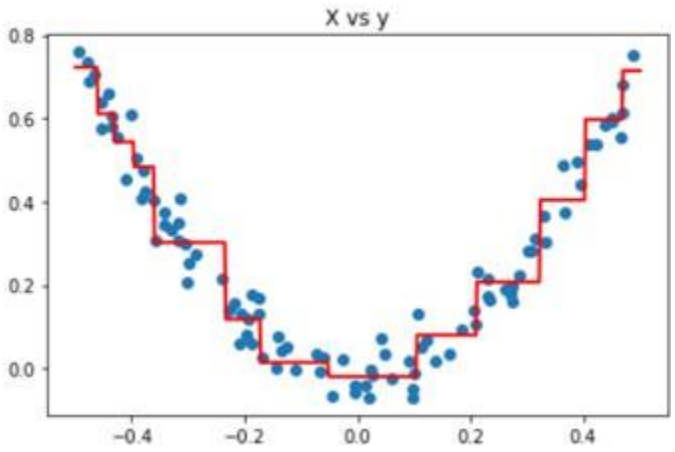
[26]:

	X	y	pred1	res1	pred2	res2
0	-0.125460	0.051573	0.265458	-0.213885	0.018319	0.033254
1	0.450714	0.594480	0.265458	0.329021	0.605884	-0.011404
2	0.231994	0.166052	0.265458	-0.099407	0.215784	-0.049732
3	0.098658	-0.070178	0.265458	-0.335636	0.018319	-0.088497
4	-0.343981	0.343986	0.265458	0.078528	0.305964	0.038022
...	...	...	...	...	...	...
95	-0.006204	-0.040675	0.265458	-0.306133	0.018319	-0.058994
96	0.022733	-0.002305	0.265458	-0.267763	0.018319	-0.020624
97	-0.072459	0.032809	0.265458	-0.232650	0.018319	0.014489
98	-0.474581	0.689516	0.265458	0.424057	0.660912	0.028604
99	-0.392109	0.502607	0.265458	0.237148	0.487796	0.014810

100 rows × 6 columns

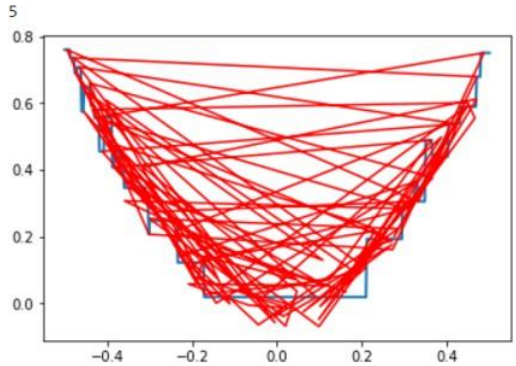
```
tree2 = DecisionTreeRegressor(max_leaf_nodes=8)
tree2.fit(df['X'].values.reshape(100,1),df['res2'].values)
DecisionTreeRegressor(max_leaf_nodes=8)
y_pred = 0.265458 + sum(regressor.predict(X_test.reshape(-1, 1)) for regressor in
[tree1,tree2])


plt.figure(figsize=(14,4))
plt.subplot(121)
plt.plot(X_test, y_pred, linewidth=2,color='red')
plt.scatter(df['X'],df['y'])
plt.title('X vs y')
```

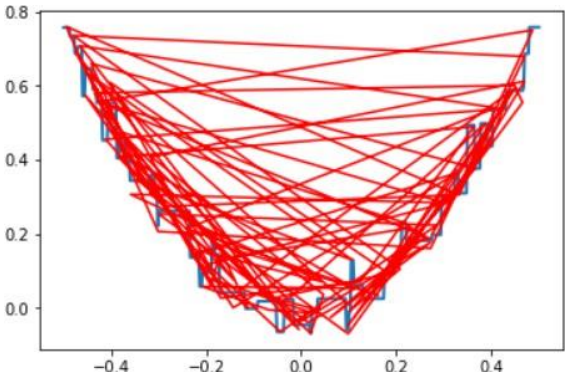
```
[30]: Text(0.5, 1.0, 'X vs y')
```

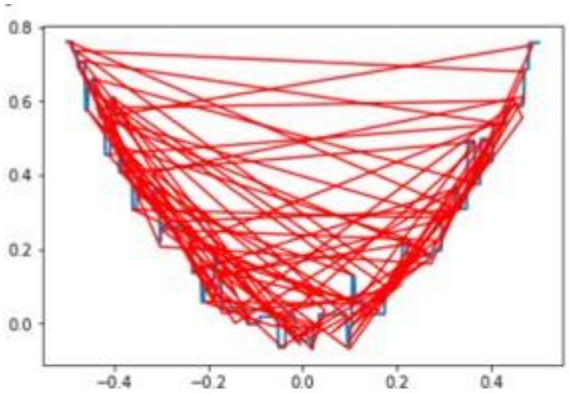



```
def gradient_boost(X,y,number,lr,count=1,regs=[],foo=None):
    if number == 0:
        return
    else:
# do gradient boosting
        if count > 1:
            y = y - regs[-1].predict(X)
        else:
```

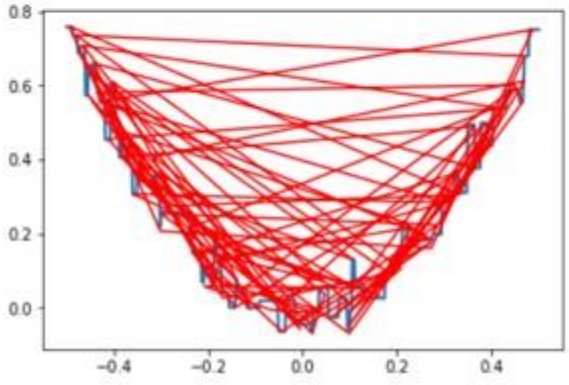
```
        foo = y
    tree_reg = DecisionTreeRegressor(max_depth=5, random_state=42)
    tree_reg.fit(X, y)

    regs.append(tree_reg)

    x1 = np.linspace(-0.5, 0.5, 500)
    y_pred = sum(lr * regressor.predict(x1.reshape(-1, 1)) for regressor in regs)

    print(number)
    plt.figure()
    plt.plot(x1, y_pred, linewidth=2)
    plt.plot(X[:, 0], foo,"r")
    plt.show()

    gradient_boost(X,y,number-1,lr,count+1,regs,foo=foo)

np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
gradient_boost(X,y,5,lr=1)
```




0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4


0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4

2


0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4

RESULT:

Thus, the python program to implement gradient boosting for the standard uniform distribution has been successfully implemented and the results have been verified and analyzed.

Ex. No.: 9 a.
Date:28/10/
2024

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.

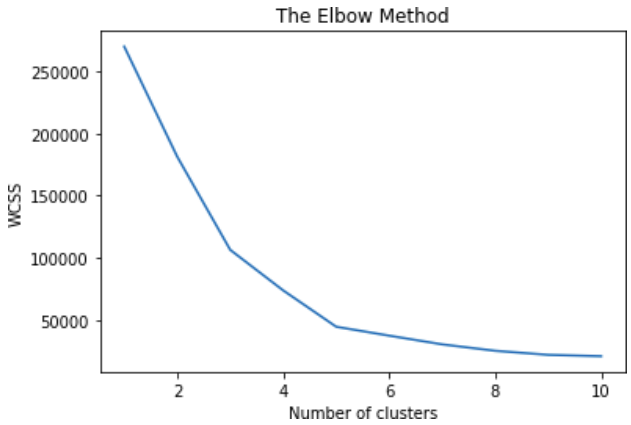
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('../input/mall-customers/Mall_Customers.csv')
X = dataset.iloc[:,[3,4]].values
print(dataset)
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
..	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

[200 rows x 5 columns]

```
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
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int32)
```

type(y_kmeans)

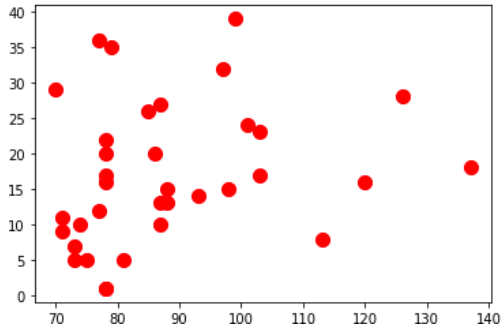
```
numpy.ndarray
```

y_kmeans

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int32)
```

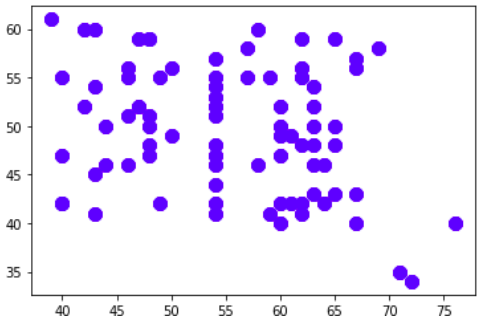
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')

```
<matplotlib.collections.PathCollection at 0x7f2c79858c90>
```



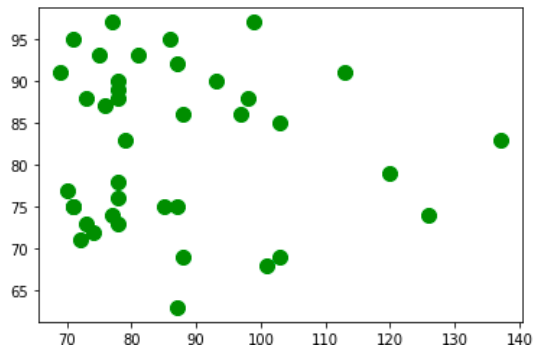
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')

```
<matplotlib.collections.PathCollection at 0x7f2c95155bd0>
```



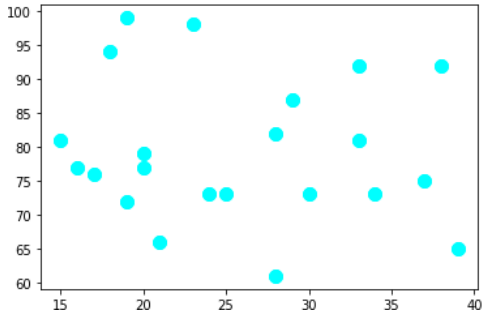
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')

```
<matplotlib.collections.PathCollection at 0x7f2c95063490>
```



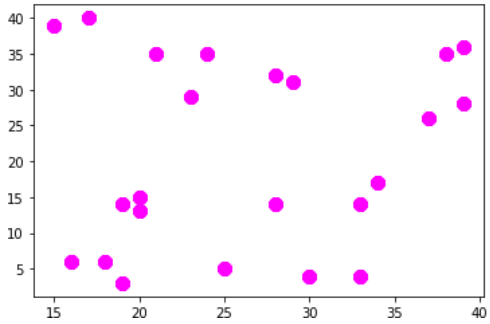
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

```
<matplotlib.collections.PathCollection at 0x7f2c94feb890>
```



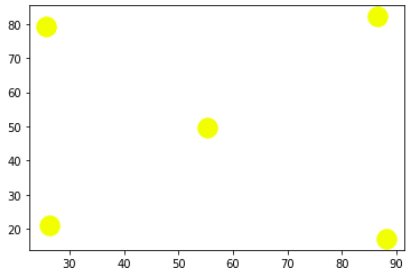
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

```
<matplotlib.collections.PathCollection at 0x7f2c94f756d0>
```

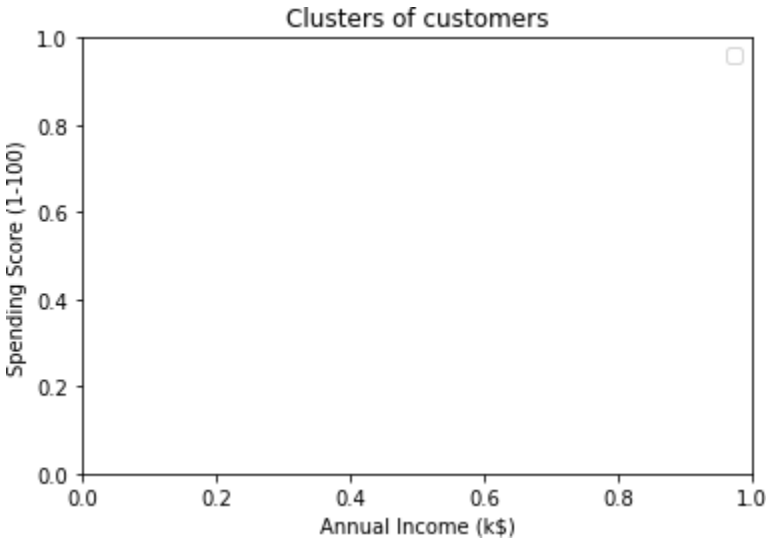


plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

```
<matplotlib.collections.PathCollection at 0x7f2c94f75650>
```

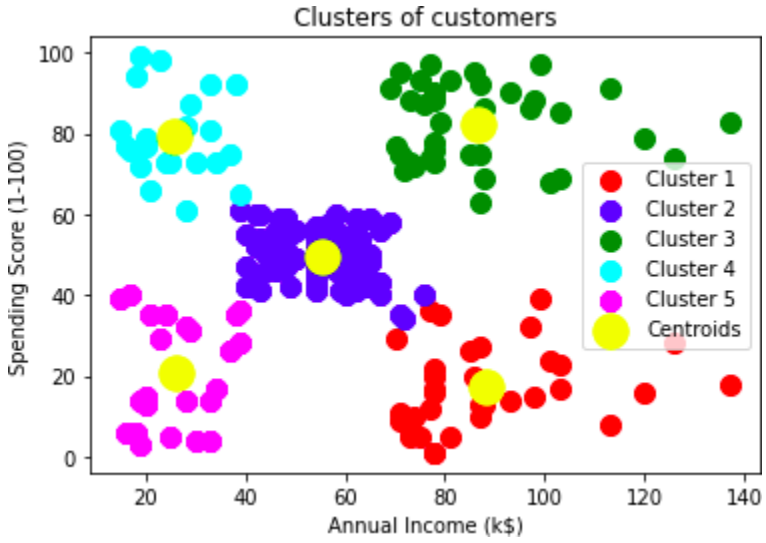



```
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```




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



RESULT:-

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

Ex. No.: 9 b.
Date:4/11/2
024

A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL

Aim:

To implement a python program using a K-Means Algorithm in a model.

Algorithm:

1. Import Necessary Libraries:

   Import required libraries like numpy, matplotlib.pyplot, and sklearn.cluster.

2. Load and Preprocess Data:

   Load the dataset.
   Preprocess the data if needed (e.g., scaling).

3. Initialize Cluster Centers:

   Choose the number of clusters (K).
   Initialize K cluster centers randomly.

4. Assign Data Points to Clusters:

   For each data point, calculate the distance to each cluster center.
   Assign the data point to the cluster with the nearest center.

5. Update Cluster Centers:

   Calculate the mean of the data points in each cluster.
   Update the cluster centers to the calculated means.

6. Repeat Steps 4 and 5:

   Repeat the assignment of data points to clusters and updating of cluster centers until convergence (i.e., when the cluster assignments do not change much between iterations).

7. Plot the Clusters:

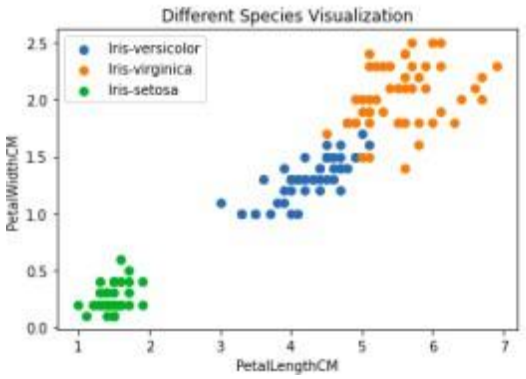
   Plot the data points and the cluster centers to visualize the clustering result.

PROGRAM:

data = pd.read_csv('../input/k-means-clustering/KNN (3).csv')

data.head(5)

```
Text(0.5, 1.0, 'Different Species Visualization')
```



req_data = data.iloc[:,1:]

req_data.head(5)

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

shuffle_index = np.random.permutation(req_data.shape[0])

#shuffling the row index of our dataset

req_data = req_data.iloc[shuffle_index] req_data.head(5)

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
45	4.8	3.0	1.4	0.3	Iris-setosa
50	7.0	3.2	4.7	1.4	Iris-versicolor
135	7.7	3.0	6.1	2.3	Iris-virginica
49	5.0	3.3	1.4	0.2	Iris-setosa
89	5.5	2.5	4.0	1.3	Iris-versicolor

```
train_size = int(req_data.shape[0]*0.7)
train_df = req_data.iloc[:train_size,:]
test_df = req_data.iloc[train_size:,:]
train = train_df.values
test = test_df.values
y_true = test[:,-1]
print('Train_Shape: ',train_df.shape)
print('Test_Shape: ',test_df.shape)
```

```
Train_Shape:  (105, 5)
Test_Shape:  (45, 5)
```

```
from math import sqrt
def euclidean_distance(x_test, x_train):
distance = 0
for i in range(len(x_test)-1):
distance += (x_test[i]-x_train[i])**2
return sqrt(distance)
def get_neighbors(x_test, x_train, num_neighbors):
distances = []
```

```
data = []
for i in x_train:
distances.append(euclidean_distance(x_test,i))
data.append(i)
distances = np.array(distances)
data = np.array(data)
sort_indexes = distances.argsort() #argsort() function returns indices by sorting
distances data in ascending order
data = data[sort_indexes] #modifying our data based on sorted indices, so that we
can get the nearest neighbors
return data[:num_neighbors]
def prediction(x_test, x_train, num_neighbors):
classes = []
neighbors = get_neighbors(x_test, x_train, num_neighbors)
for i in neighbors:
classes.append(i[-1])
predicted = max(classes, key=classes.count) #taking the most repeated class
return predicted
def predict_classifier(x_test):
classes = []
neighbors = get_neighbors(x_test, req_data.values, 5)
for i in neighbors:
classes.append(i[-1])
predicted = max(classes, key=classes.count)
print(predicted)
return predicted
def accuracy(y_true, y_pred):
num_correct = 0
for i in range(len(y_true)):
```

```
if y_true[i]==y_pred[i]:
num_correct+=1
accuracy = num_correct/len(y_true)
return accuracy
y_pred = []
for i in test:
y_pred.append(prediction(i, train, 5))
y_pred
```

```
['Iris-virginica',
 'Iris-versicolor',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-setosa',
```

```
accuracy = accuracy(y_true, y_pred)
accuracy
```

```
0.9555555555555556
```

```
test_df.sample(5)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
113	5.7	2.5	5.0	2.0	Iris-virginica
125	7.2	3.2	6.0	1.8	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica
94	5.6	2.7	4.2	1.3	Iris-versicolor
99	5.7	2.8	4.1	1.3	Iris-versicolor

RESULT:-

Thus the python program to implement the K-Means model has been successfully implemented and the results have been verified and analyzed

Ex. No.: 10
Date:10/11/2024

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.

PROGRAM:

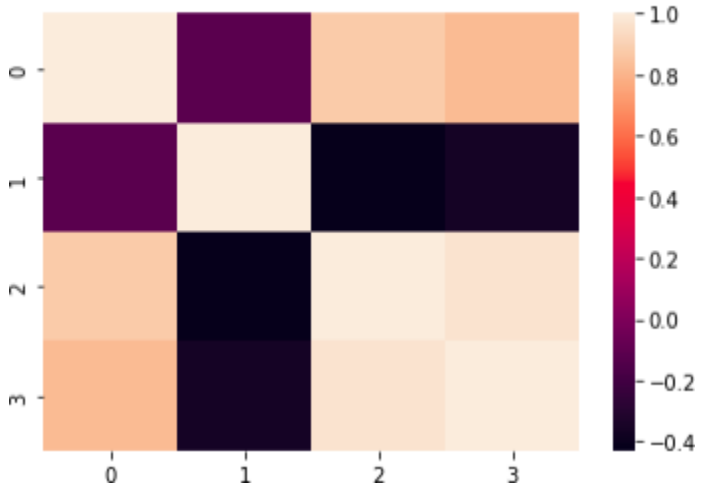
```
from sklearn import datasets
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns
iris = datasets.load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
df.head()
```

	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

```
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
scaled_data
```

```
sns.heatmap(scaled_data.corr())
```

```
<AxesSubplot:>
```

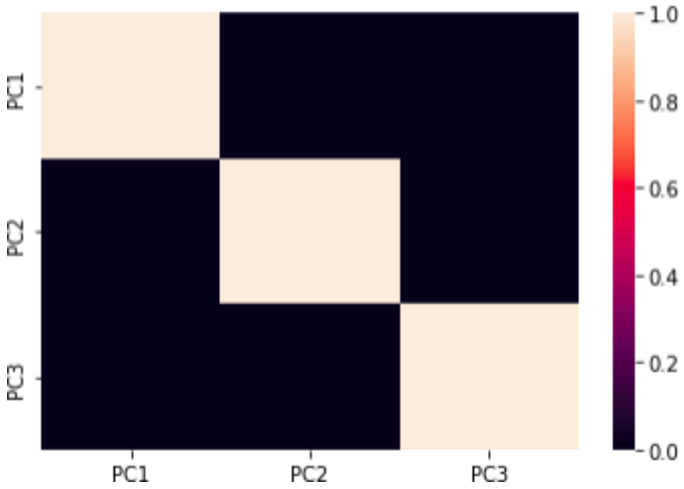



```
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
data_pca.head()
```

	PC1	PC2	PC3
0	-2.264703	0.480027	-0.127706
1	-2.080961	-0.674134	-0.234609
2	-2.364229	-0.341908	0.044201
3	-2.299384	-0.597395	0.091290
4	-2.389842	0.646835	0.015738

```
sns.heatmap(data_pca.corr())
```

```
<AxesSubplot:>
```



RESULT:-

Thus Dimensionality Reduction has been implemented using PCA in a python program successfully and the results have been analyze