**Ex. No.: 8b**

**Date: 11/10/24**

**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)

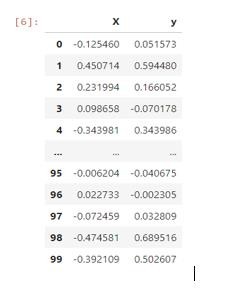
X = np.random.rand(100, 1) - 0.5 y = 3\*X[:, 0]\*\*2 + 0.05 \* np.random.randn(100)

df = pd.DataFrame()

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

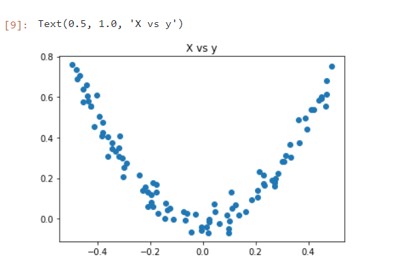
df['y'] = y

df

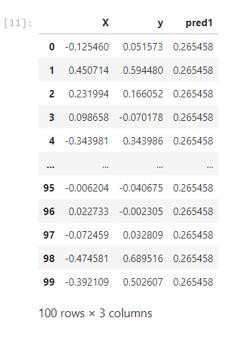


plt.scatter(df['X'],df['y']) plt.title('X vs y')

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

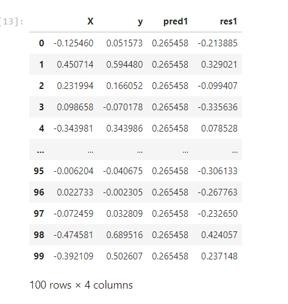


df['pred1'] = df['y'].mean() df

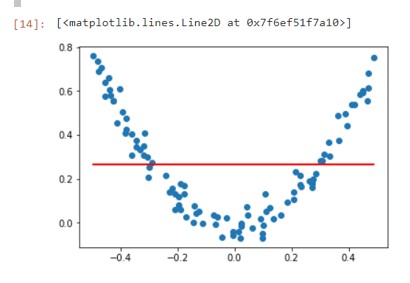


df['res1'] = df['y'] - df['pred1']

df



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



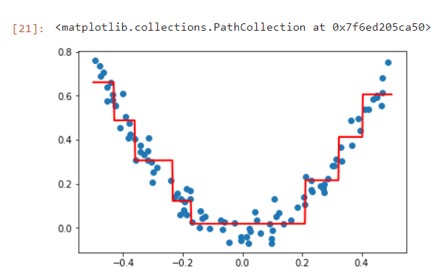
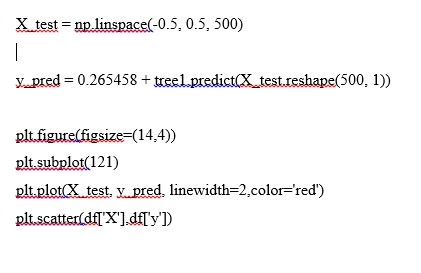
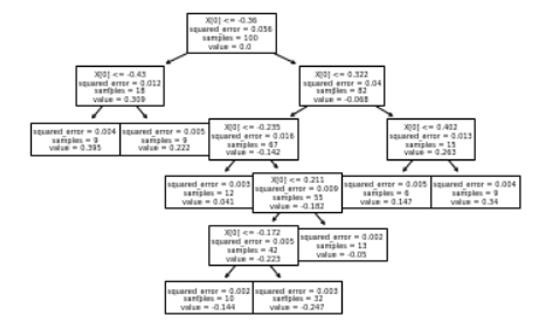
from sklearn.tree import DecisionTreeRegressor

tree1 = DecisionTreeRegressor(max\_leaf\_nodes=8)

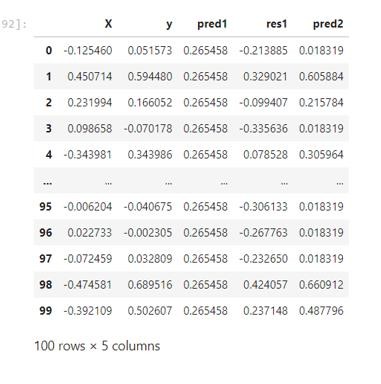
tree1.fit(df['X'].values.reshape(100,1),df['res1'].values)

DecisionTreeRegressor(max\_leaf\_nodes=8)

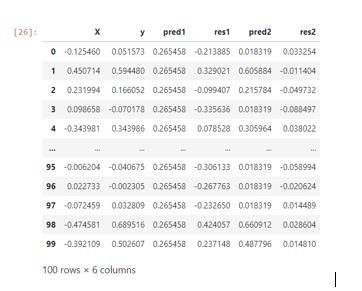
from sklearn.tree import plot\_tree plot\_tree(tree1) plt.show()

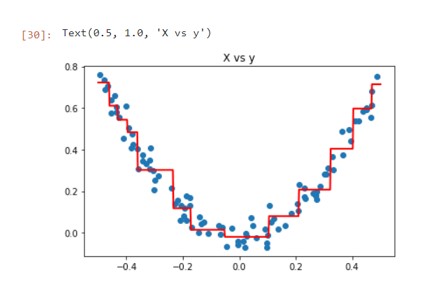
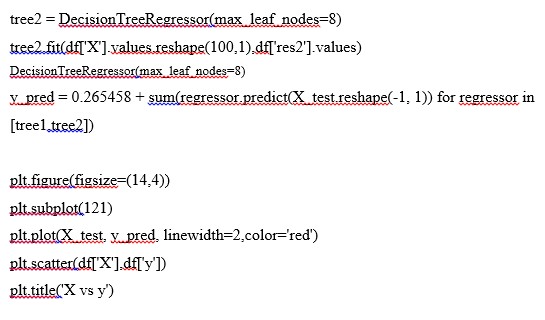


df['pred2'] = 0.265458 + tree1.predict(df['X'].values.reshape(100,1)) df



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





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)

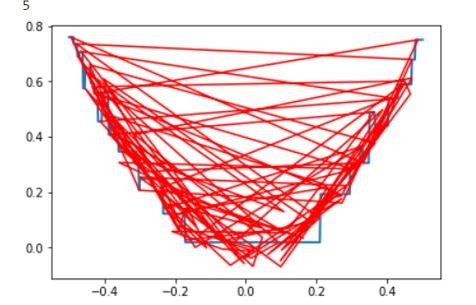
x1 = np.linspace(-0.5, 0.5, 500) y\_pred = sum(lr \* regressor.predict(x1.reshape(-1, 1)) for regressor in regs)

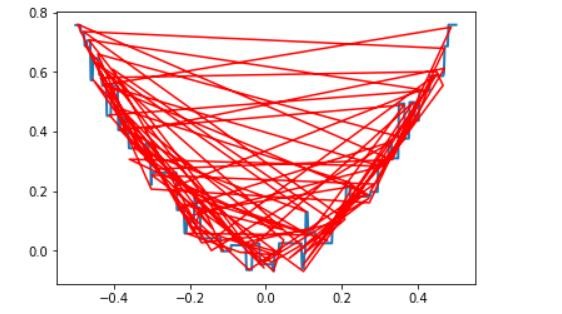
print(number) plt.figure() plt.plot(x1, y\_pred, linewidth=2) plt.plot(X[:, 0], foo,"r") plt.show()

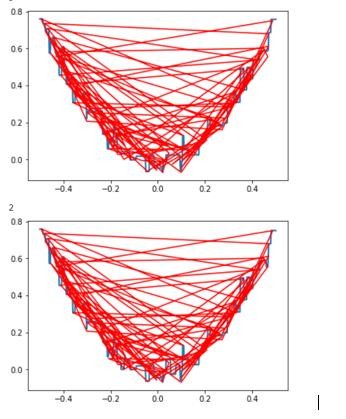
gradient\_boost(X,y,number-1,lr,count+1,regs,foo=foo)

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)







**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.