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

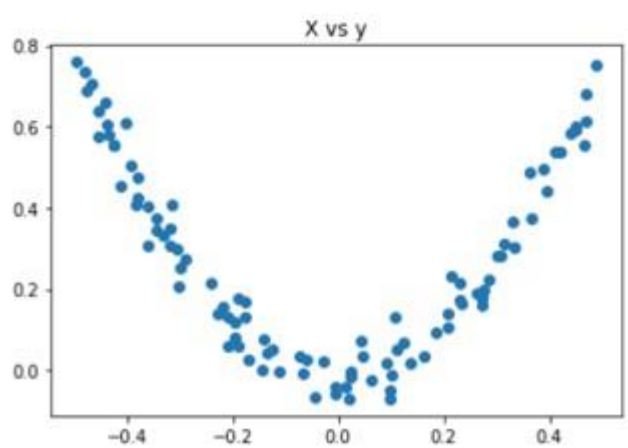
[6]:

	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
```

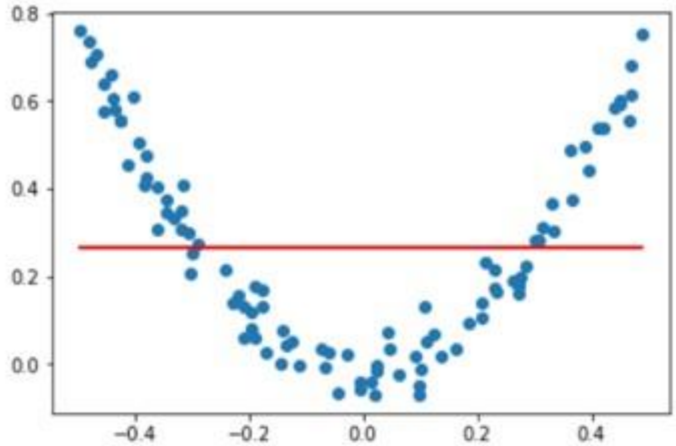
[13]:

	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>]



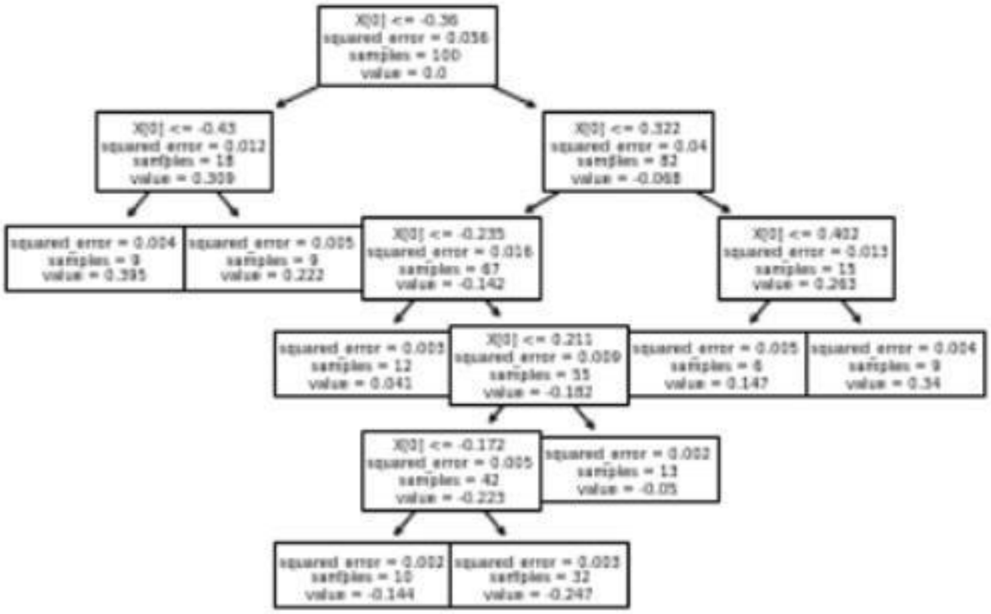

```
from sklearn.tree import DecisionTreeRegressor

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()
```

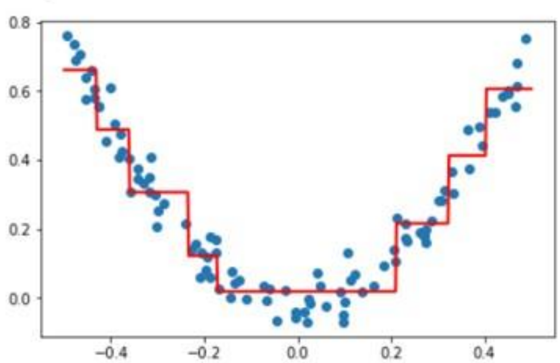



```
X_test = np.linspace(-0.5, 0.5, 500)

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]:

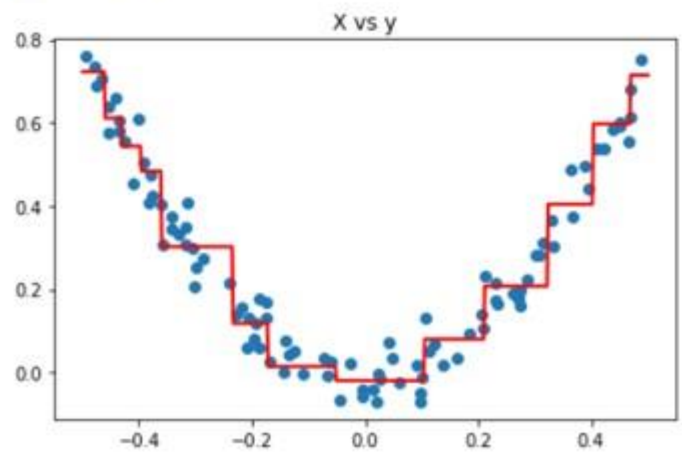
	X	y	pred1	res1	pred2	res2
0	-0.125460	0.051573	0.265458	-0.213885	0.018319	0.033254
1	0.450714	0.594480	0.265458	0.329021	0.605884	-0.011404
2	0.231994	0.166052	0.265458	-0.099407	0.215784	-0.049732
3	0.098658	-0.070178	0.265458	-0.335636	0.018319	-0.088497
4	-0.343981	0.343986	0.265458	0.078528	0.305964	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)

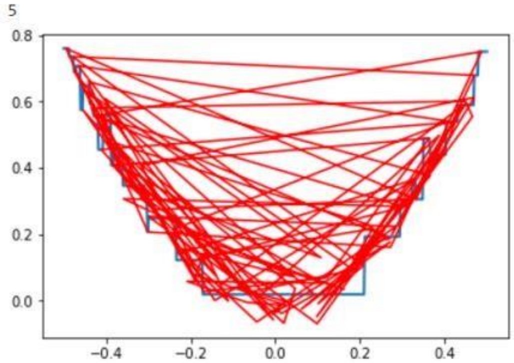
x1 = np.linspace(-0.5, 0.5, 500) y_pred = sum(lr * regressor.predict(x1.reshape(-1, 1)) for regressor in regs)

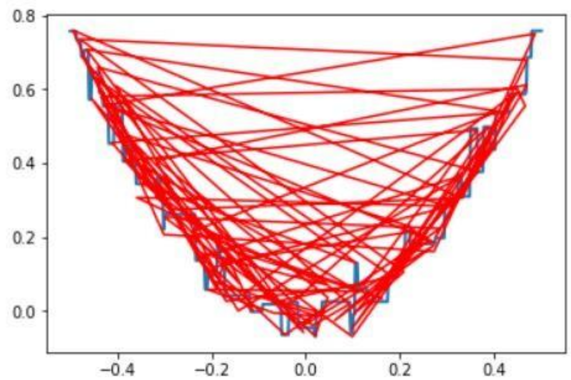
print(number) plt.figure()
plt.plot(x1, y_pred, linewidth=2)
plt.plot(X[:, 0], foo,"r") plt.show()

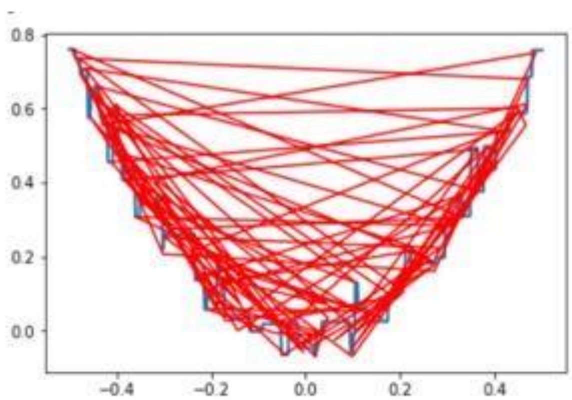
gradient_boost(X,y,number-1,lr,count+1,regs,foo=foo)

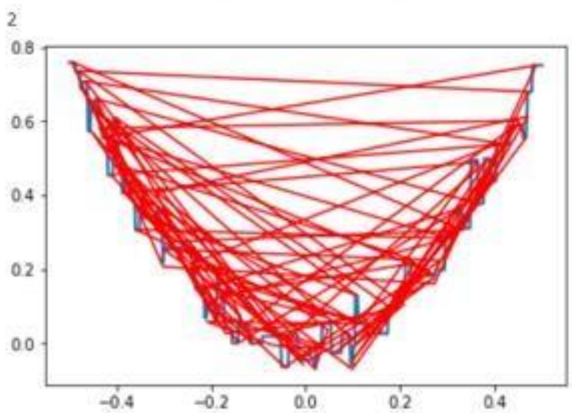
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5 y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
gradient_boost(X,y,5,lr=1)

5









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.