EXPERIMENT -8 (b)

A python program using the gradient boosting model

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.

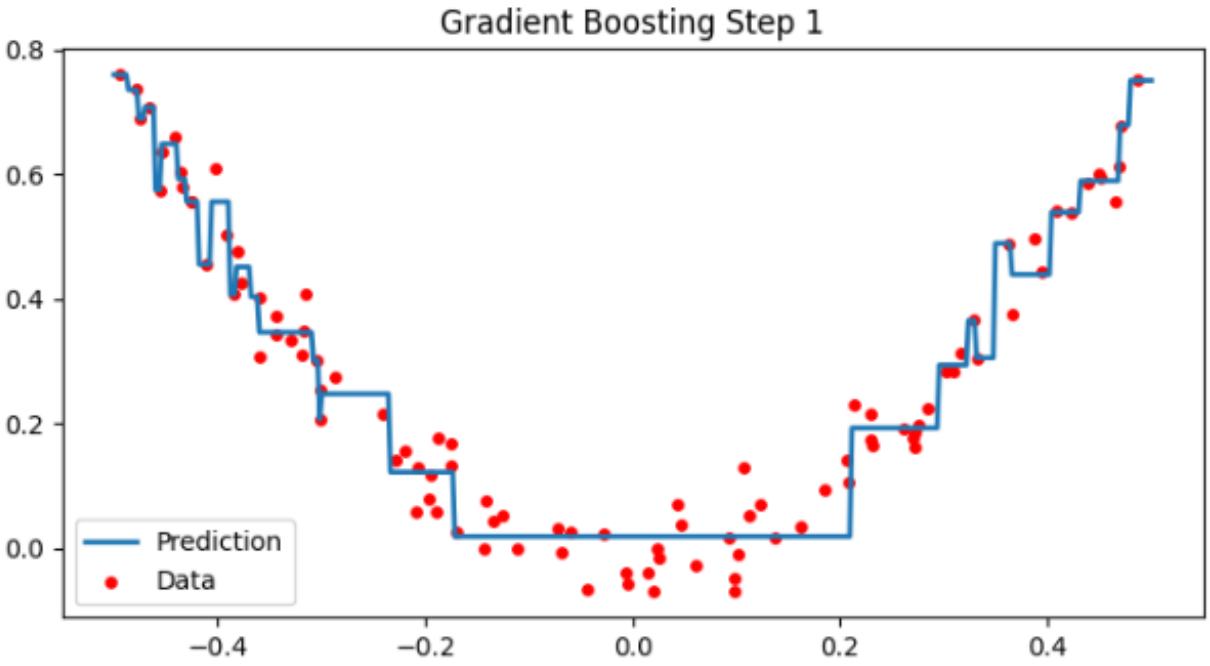
Step 7: Evaluate the Model Compute the mean squared error on the training data. Compute the mean squared error on the test data.

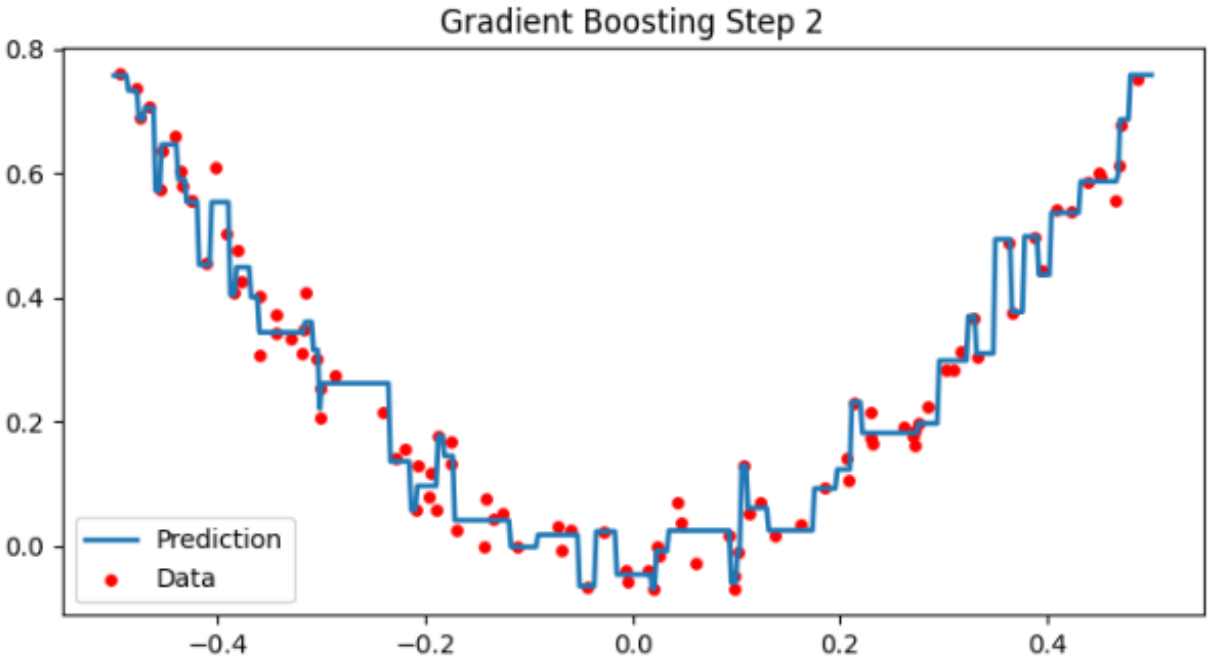
CODE

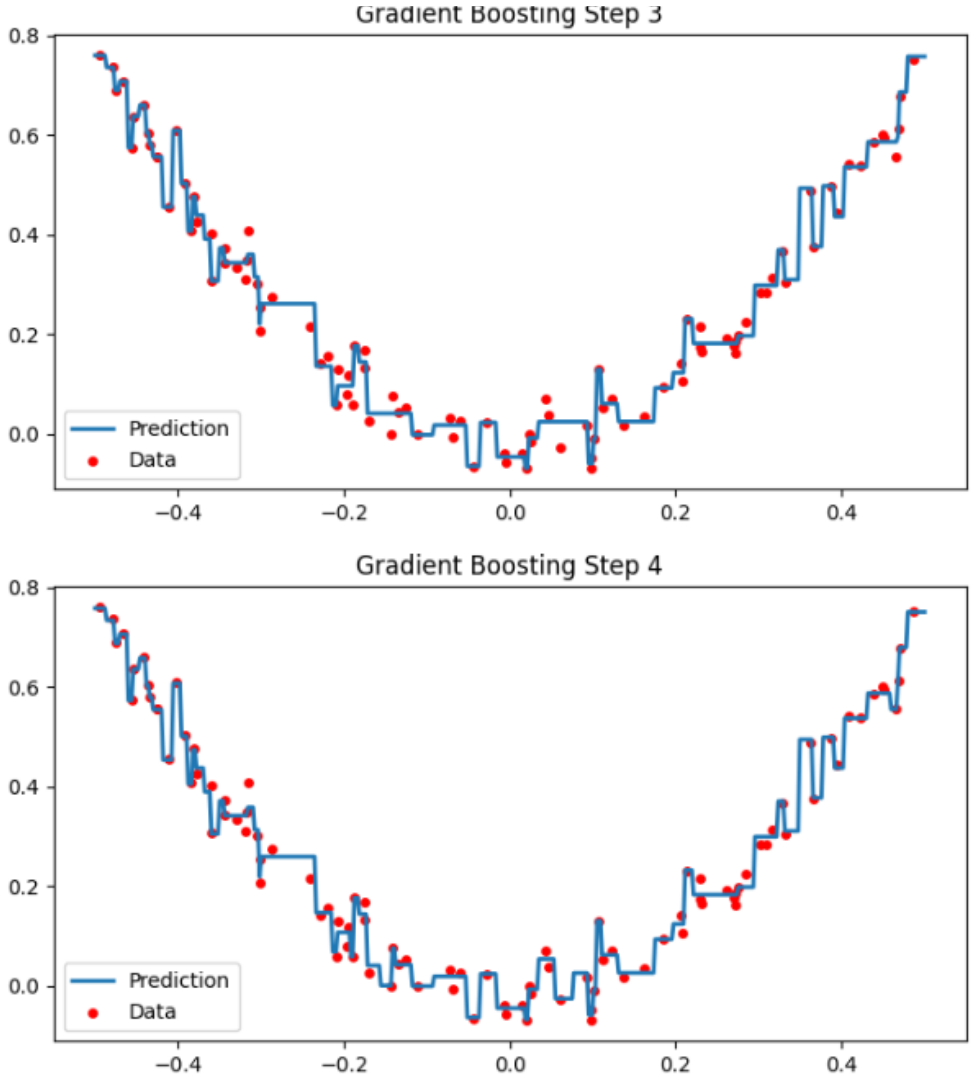
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
df = pd.DataFrame()
df['X'] = X.reshape(100)
df['y'] = y
df

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



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

```
df['res2'] = df['y'] - df['pred2']
from sklearn.tree import DecisionTreeRegressor
tree2 = DecisionTreeRegressor(max_leaf_nodes=8)
tree2.fit(df['X'].values.reshape(100,1),df['res2'].values)
y_pred = df['pred1'].iloc[0] + tree1.predict(X_test.reshape(-1,1)) +
tree2.predict(X_test.reshape(-1,1))
plt.figure(figsize=(14,4))
plt.subplot(121)
plt.plot(X_test, y_pred, linewidth=2, color='red')
plt.scatter(df['X'], df['y'])
plt.title('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)
```

OUTPUT


Gradient Boosting Step 1
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
Prediction
Data


Gradient Boosting Step 2
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
Prediction
Data



RESULT:

Thus a python program to implement gradient boosting is written and the output is verified.