Ex. No.: 8 a.

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.

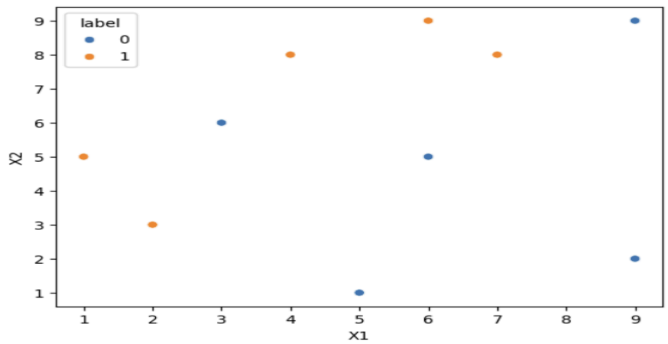
PROGRAM:

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

O/P:

	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

<Axes: xlabel='X1', ylabel='X2'>




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

O/P:

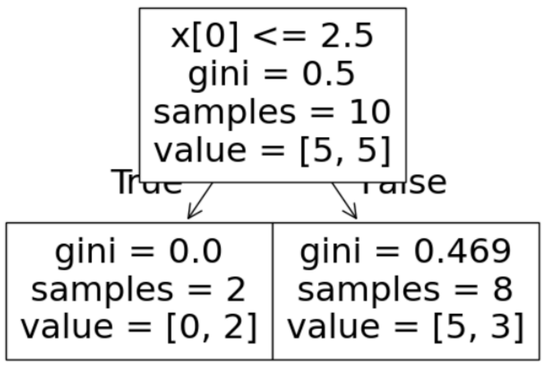
	X1	X2	label	weights
0	1	5	1	0.1
1	2	3	1	0.1
2	3	6	0	0.1
3	4	8	1	0.1
4	5	1	0	0.1
5	6	9	1	0.1
6	6	5	0	0.1
7	7	8	1	0.1
8	9	9	0	0.1
9	9	2	0	0.1

```
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)
```

O/P:

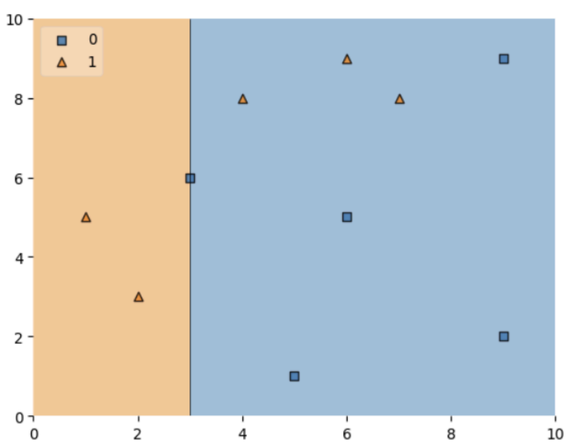
```
[Text(0.5, 0.75, 'x[0] <= 2.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.375, 0.5, 'True  '),
 Text(0.75, 0.25, 'gini = 0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(0.625, 0.5, '  False')]
```




```
plot_decision_regions(x, y, clf=dt1, legend=2)
```

O/P:

Axes: >



```
df['y pred'] = dt1.predict(x)
display (df)
```

O/P:

	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	0
3	4	8	1	0.1	0
4	5	1	0	0.1	0
5	6	9	1	0.1	0
6	6	5	0	0.1	0
7	7	8	1	0.1	0
8	9	9	0	0.1	0
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)
# Step -4 Update weights
def update_row_weights(row,alpha):
  if row['label'] == row['y pred']:
    return row['weights']* np.exp(-alpha)
  else:
    return row['weights']* np.exp(alpha)
df['updated_weights'] = df.apply(lambda row:
update_row_weights(row, alpha1), axis=1)
display (df)
```

O/P:

	X1	X2	label	weights	y pred	updated_weights
0	1	5	1	0.1	1	0.065465
1	2	3	1	0.1	1	0.065465
2	3	6	0	0.1	0	0.065465
3	4	8	1	0.1	0	0.152753
4	5	1	0	0.1	0	0.065465
5	6	9	1	0.1	0	0.152753
6	6	5	0	0.1	0	0.065465
7	7	8	1	0.1	0	0.152753
8	9	9	0	0.1	0	0.065465
9	9	2	0	0.1	0	0.065465

df['updated_weights'].sum()

O/P:

```
np.float64(0.9165151389911682)
```

df['cumsum_upper'] = np.cumsum(df['normalized weights'])

df['cumsum_lower']=df['cumsum_upper'] - df['normalized weights']

display(df[['X1','X2','label','weights','y pred','updated_weights','cumsum_lower','cumsum_upper']])

O/P:

	X1	X2	label	weights	y pred	updated_weights	cumsum_lower	cumsum_upper
0	1	5	1	0.1	1	0.065465	0.000000	0.071429
1	2	3	1	0.1	1	0.065465	0.071429	0.142857
2	3	6	0	0.1	0	0.065465	0.142857	0.214286
3	4	8	1	0.1	0	0.152753	0.214286	0.380952
4	5	1	0	0.1	0	0.065465	0.380952	0.452381
5	6	9	1	0.1	0	0.152753	0.452381	0.619048
6	6	5	0	0.1	0	0.065465	0.619048	0.690476
7	7	8	1	0.1	0	0.152753	0.690476	0.857143
8	9	9	0	0.1	0	0.065465	0.857143	0.928571
9	9	2	0	0.1	0	0.065465	0.928571	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
```

O/P:

```
[6, 0, 5, 4, 6, 7, 5, 8, 3, 3]
```

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

O/P:

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

```
dt2 = DecisionTreeClassifier(max_depth=1)
x = second_df.iloc[:,0:2].values
y = second_df.iloc[:,2].values
dt2.fit(x,y)
```

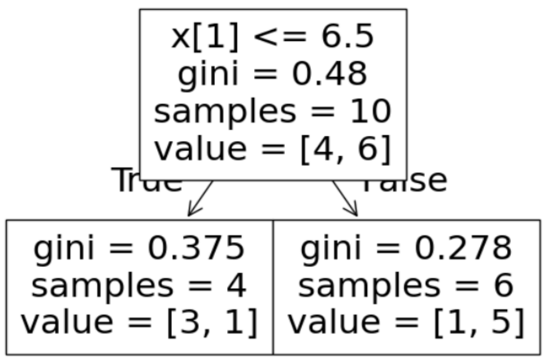
O/P:

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

plot_tree(dt2)

O/P:

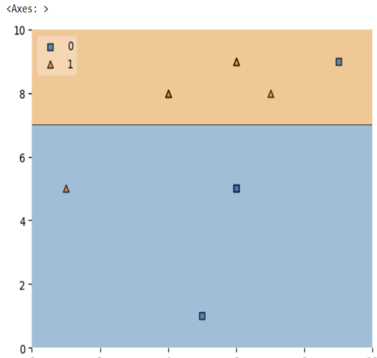
```
[Text(0.5, 0.75, 'x[1] <= 6.5\ngini = 0.48\nsamples = 10\nvalue = [4, 6]'),
 Text(0.25, 0.25, 'gini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.375, 0.5, 'True  '),
 Text(0.75, 0.25, 'gini = 0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.625, 0.5, '  False')]
```



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

O/P:

<Axes: >



```
second_df['y_pred'] = dt2.predict(x)
second_df
alpha2 = calculate_model_weight(0.1)
display(second_df)
```

O/P:

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

Alpha2

O/P:

```
np.float64(1.0986122886681098)
```

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

second_df

display(second_df)

second_df['normalized_weights'].sum()

O/P:

	X1	X2	label	weights	y_pred	updated_weights	normalized_weights
6	6	5	0	0.1	0	0.033622	0.038922
0	1	5	1	0.1	0	0.297427	0.344313
5	6	9	1	0.1	1	0.033622	0.038922
4	5	1	0	0.1	0	0.033622	0.038922
6	6	5	0	0.1	0	0.033622	0.038922
7	7	8	1	0.1	1	0.033622	0.038922
5	6	9	1	0.1	1	0.033622	0.038922
8	9	9	0	0.1	1	0.297427	0.344313
3	4	8	1	0.1	1	0.033622	0.038922
3	4	8	1	0.1	1	0.033622	0.038922

```
np.float64(0.9999999999999999)
```

second_df['cumsum_upper'] = np.cumsum(second_df['normalized_weights'])

second_df['cumsum_lower'] = second_df['cumsum_upper'] - second_df['normalized_weights']

second_df[['X1','X2','label','weights','y_pred','normalized_weights','cumsum_lower','cumsum_upper']]

O/P:

	X1	X2	label	weights	y_pred	normalized_weights	cumsum_lower	cumsum_upper
6	6	5	0	0.1	0	0.038922	0.000000	0.038922
0	1	5	1	0.1	0	0.344313	0.038922	0.383235
5	6	9	1	0.1	1	0.038922	0.383235	0.422157
4	5	1	0	0.1	0	0.038922	0.422157	0.461078
6	6	5	0	0.1	0	0.038922	0.461078	0.500000
7	7	8	1	0.1	1	0.038922	0.500000	0.538922
5	6	9	1	0.1	1	0.038922	0.538922	0.577843
8	9	9	0	0.1	1	0.344313	0.577843	0.922157
3	4	8	1	0.1	1	0.038922	0.922157	0.961078
3	4	8	1	0.1	1	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

O/P:

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

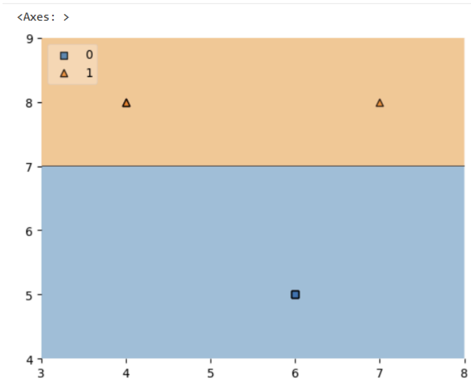
```
from sklearn.tree import DecisionTreeClassifier
dt3 = DecisionTreeClassifier(max_depth=1)
x = third_df.iloc[:,0:2].values
y = third_df.iloc[:,2].values
dt3.fit(x,y)
```

O/P:

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

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

O/P:



third_df['y_pred'] = dt3.predict(x)

third_df

alpha3 = calculate_model_weight(0.7)

Alpha3

O/P:

```
np.float64(-0.4236489301936017)
```

print(alpha1,alpha2,alpha3)

O/P:

```
0.42364893019360184 1.0986122886681098 -0.4236489301936017
```

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

dt1.predict(query)

O/P:

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

dt2.predict(query)

O/P:

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

dt3.predict(query)

O/P;

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

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

O/P:

```
np.float64(1.09861228866811)
```

np.sign(1.09)

O/P:

```
np.float64(1.0)
```

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

dt1.predict(query)

O/P:

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

dt2.predict(query)

O/P:

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

dt3.predict(query)

O/P:

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

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

O/P:

```
np.float64(-0.2513144282809062)
```

np.sign(-0.25)

O/P:

```
np.float64(-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:

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:

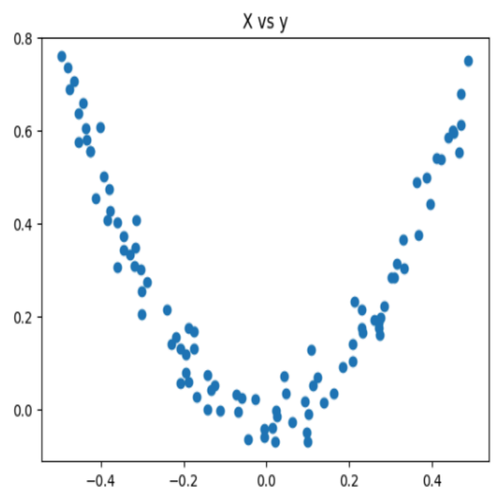
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)
df = pd.DataFrame()
df['X'] = X.reshape(100)
df['y'] = y
df
```

	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

100 rows × 2 columns

```
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
```

	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

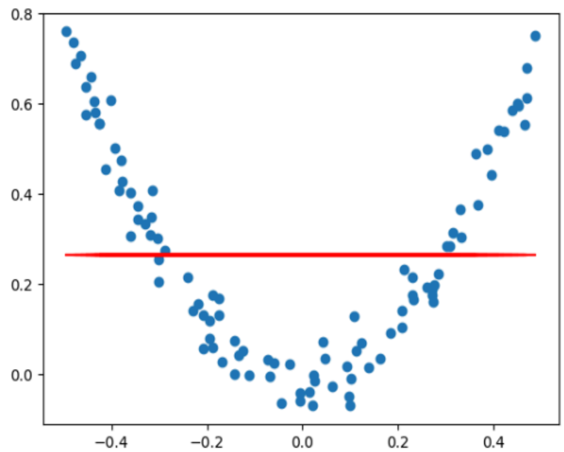
```
df['res1'] = df['y'] - df['pred1']
df
```

	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

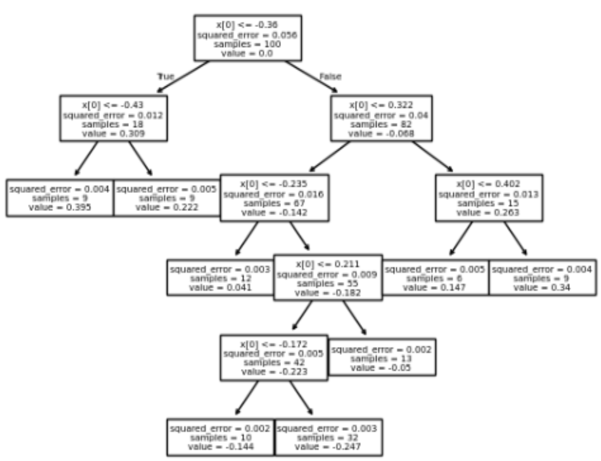
100 rows × 4 columns

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

```
[<matplotlib.lines.Line2D at 0x7aa803798c20>]
```

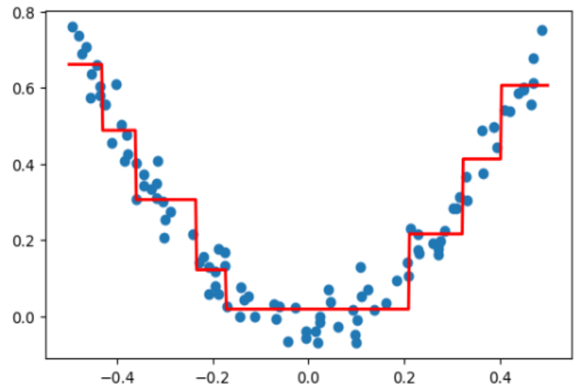



```
from sklearn.tree import DecisionTreeRegressor
tree1 = DecisionTreeRegressor(max_leaf_nodes=8)
tree1.fit(df['X'].values.reshape(100,1),df['res1'].values)
DecisionTreeRegressor(max_leaf_nodes=8)
from sklearn.tree import plot_tree
plot_tree(tree1)
plt.show()
```




```
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'])
```

<matplotlib.collections.PathCollection at 0x7aa803628e90>



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

	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

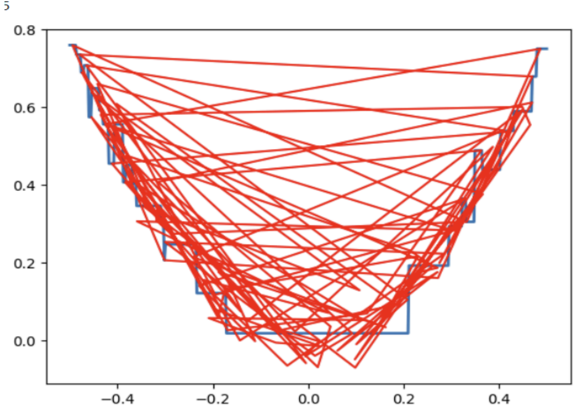
100 rows × 5 columns

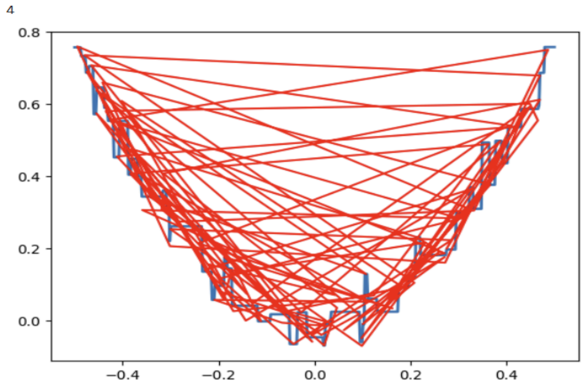
df['res2'] = df['y'] - df['pred2']
df

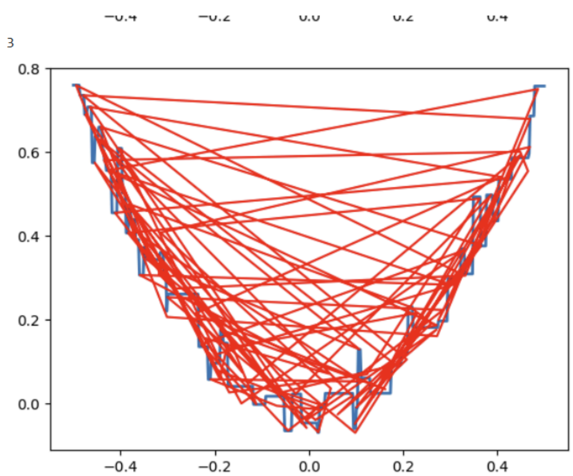
	X	y	pred1	res1	pred2	res2
0	-0.125460	0.051573	0.265458	-0.213885	0.018319	0.033254
1	0.450714	0.594480	0.265458	0.329021	0.605884	-0.011404
2	0.231994	0.166052	0.265458	-0.099407	0.215784	-0.049732
3	0.098658	-0.070178	0.265458	-0.335636	0.018319	-0.088497
4	-0.343981	0.343986	0.265458	0.078528	0.305964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

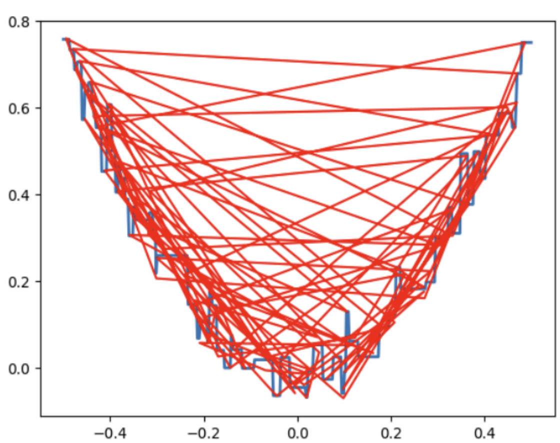
100 rows × 6 columns

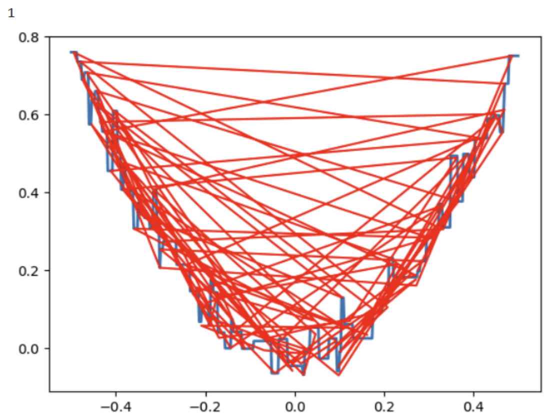
```
def gradient_boost(X,y,number,lr,count=1,regs=[],foo=None):
    if number == 0:
        return
    else:
        # do gradient boosting
        if count > 1:
            y = y - regs[-1].predict(X)
        else:
            foo = y
        tree_reg = DecisionTreeRegressor(max_depth=5, random_state=42)
…
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)
```




4
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4


3
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



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.