A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING AND GRADIENT BOOSTING

Exp No. 8A

Name: MOHAMED FUZAIL A

Roll no.: 241801162

Code:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, plot_tree


# Create dataset

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]


# Visualize dataset

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

plt.title("Data Distribution (X1 vs X2)")

plt.show()
```

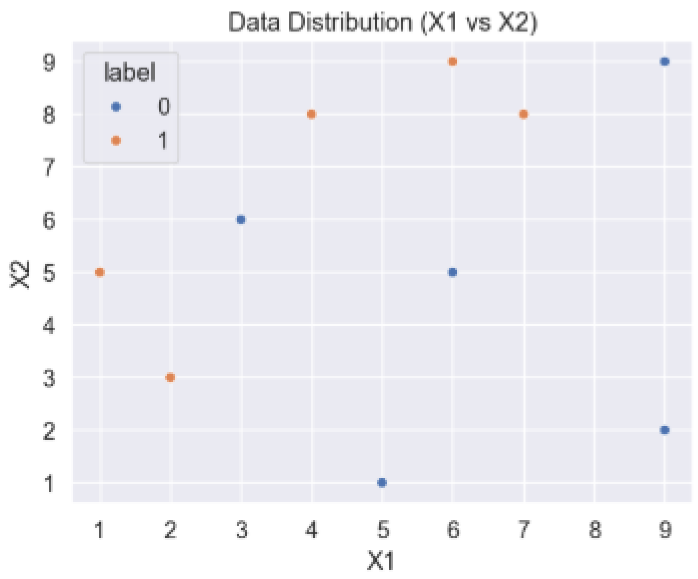
```
# Initialize equal weights
df['weights'] = 1 / df.shape[0]

# First weak learner
dt1 = DecisionTreeClassifier(max_depth=1)
x = df.iloc[:, 0:2].values
y = df.iloc[:, 2].values
dt1.fit(x, y)

# Predictions
df['y_pred'] = dt1.predict(x)

# Calculate model weight
def calculate_model_weight(error):
    return 0.5 * np.log((1 - error) / error)

alpha1 = calculate_model_weight(0.3)

# Update row 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)
```

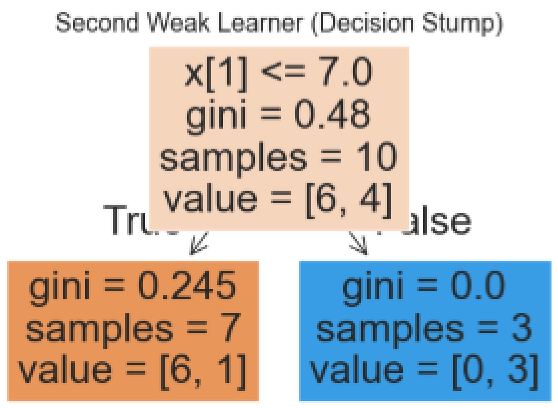
```
df['updated_weights'] = df.apply(update_row_weights, axis=1)
df['normalized_weights'] = df['updated_weights'] / df['updated_weights'].sum()
df['cumsum_upper'] = np.cumsum(df['normalized_weights'])
df['cumsum_lower'] = df['cumsum_upper'] - df['normalized_weights']

# Resampling new dataset
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)
second_df = df.iloc[index_values, [0, 1, 2, 3]]

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

# Visualize second weak learner decision tree
```

```
plt.figure(figsize=(6, 4))
plot_tree(dt2, filled=True)
plt.title("Second Weak Learner (Decision Stump)")
plt.show()

# Predictions from second model
second_df['y_pred'] = dt2.predict(x)
alpha2 = calculate_model_weight(0.1)

# Update weights again
def update_row_weights2(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_weights2, axis=1)
second_df['normalized_weights'] = second_df['updated_weights'] /
second_df['updated_weights'].sum()
second_df['cumsum_upper'] = np.cumsum(second_df['normalized_weights'])
second_df['cumsum_lower'] = second_df['cumsum_upper'] -
second_df['normalized_weights']

# Third weak learner weight
alpha3 = calculate_model_weight(0.7)
print("Model Weights: ", alpha1, alpha2, alpha3)
```

```
# Sample queries
query1 = np.array([1, 5]).reshape(1, 2)
query2 = np.array([9, 9]).reshape(1, 2)

print("\nPredictions:")
print("dt1:", dt1.predict(query1))
print("dt2:", dt2.predict(query1))
print("Weighted Vote (Query1):", np.sign(alpha1 * 1 + alpha2 * 1 + alpha3 * 1))

print("dt1:", dt1.predict(query2))
print("dt2:", dt2.predict(query2))
print("Weighted Vote (Query2):", np.sign(alpha1 * 1 + alpha2 * -1 + alpha3 * -1))
```

output:





```
Model Weights:  0.42364893019360184 1.0986122886681098 -0.4236489301936017

Predictions:
dt1: [1]
dt2: [0]
Weighted Vote (Query1): 1.0
dt1: [1]
dt2: [1]
Weighted Vote (Query2): -1.0
```

A PYTHON PROGRAM TO IMPLEMENT GRADIENT BOOSTING

EXP NO. 8B

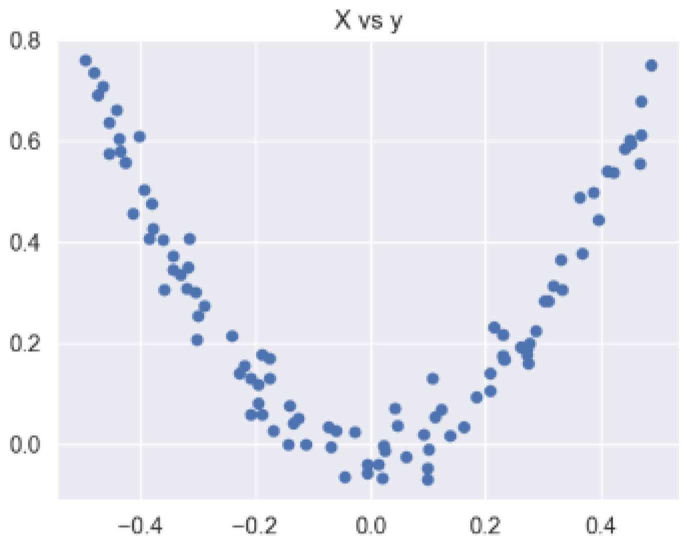
Code:

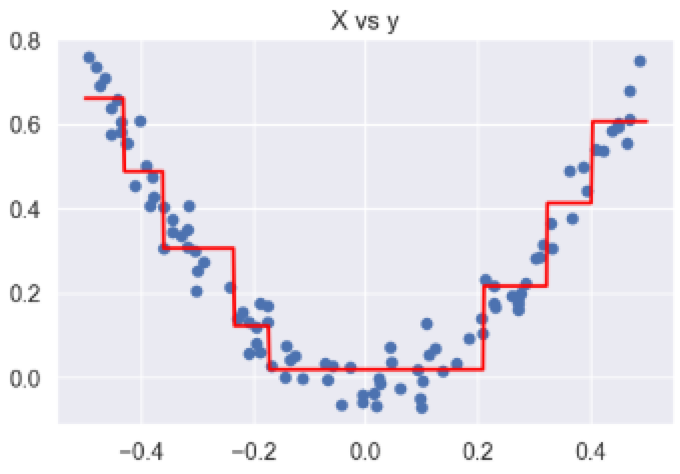
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeRegressor, plot_tree

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

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

df['pred1'] = df['y'].mean()
df['res1'] = df['y'] - df['pred1']

tree1 = DecisionTreeRegressor(max_leaf_nodes=8)
```

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


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'])
plt.title("X vs y")
plt.show()


df['pred2'] = 0.265458 + tree1.predict(df['X'].values.reshape(100, 1))
df['res2'] = df['y'] - df['pred2']


tree2 = DecisionTreeRegressor(max_leaf_nodes=8)
tree2.fit(df['X'].values.reshape(100, 1), df['res2'].values)


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")
plt.show()


def gradient_boost(X, y, number, lr, count=1, regs=[], foo=None):
    if number == 0:
        return
    else:
        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)
```

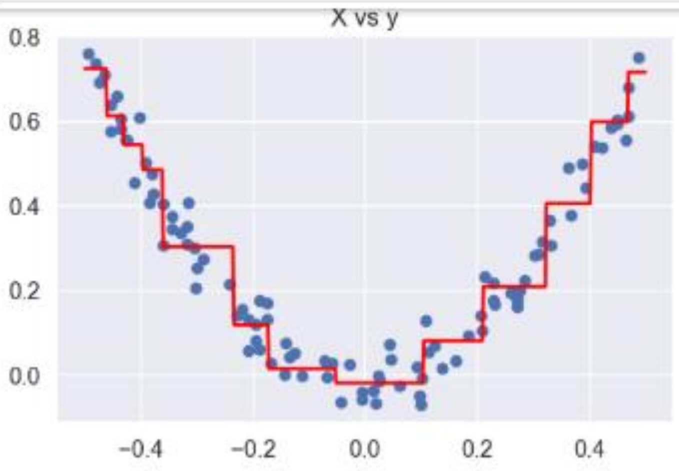
```
gradient_boost(X, y, 5, lr=1)
```

<u>output:</u>

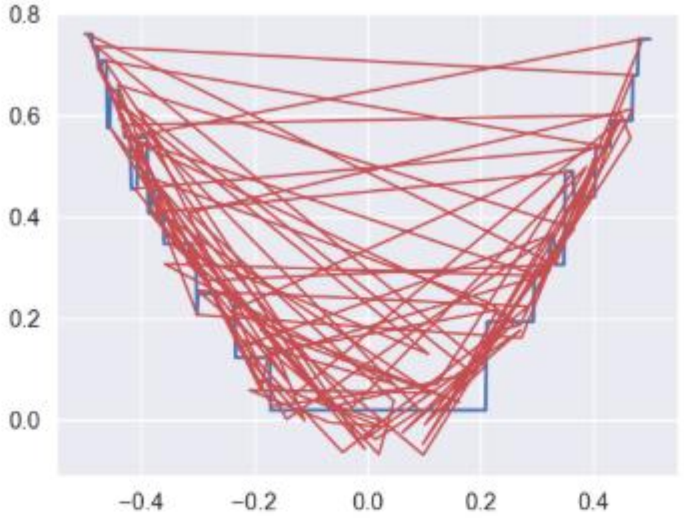


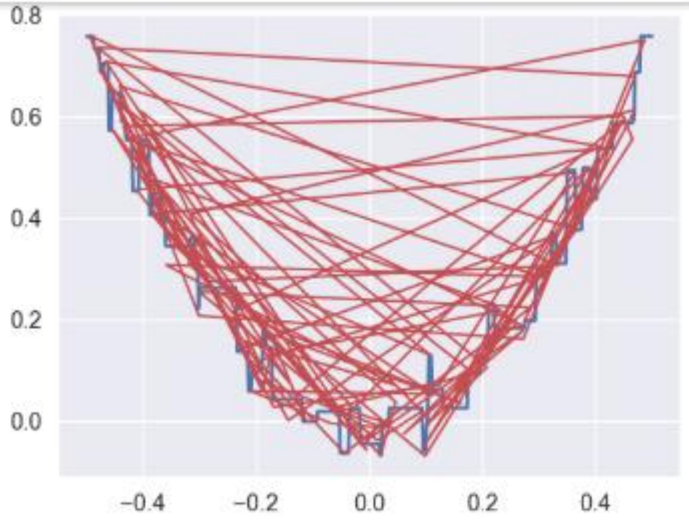


X vs y

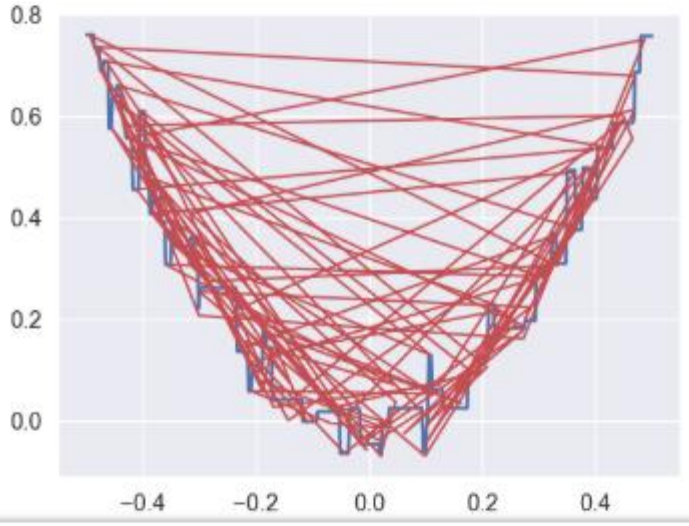


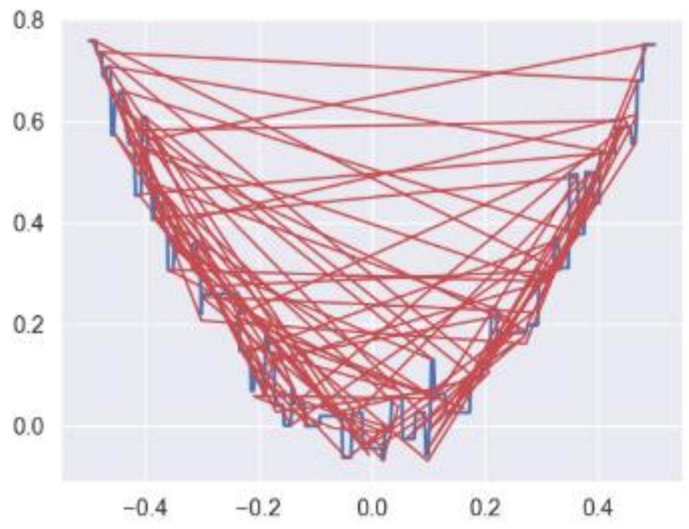
s



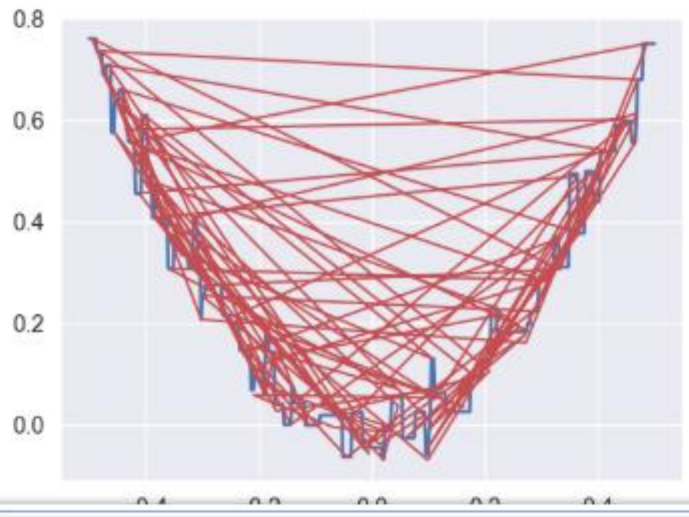


3





1



]: