A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING

Expt no. 8A

Name: Jayasuriya.j

Roll no: 241801102

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier,
plot_tree
from mlxtend.plotting import
plot_decision_regions

df = pd.DataFrame({
    'X1':[1,2,3,4,5,6,6,7,9,9],
    'X2':[5,3,6,8,1,9,5,8,9,2],
    'label':[1,1,0,1,0,1,0,1,0,0]
})
print("Original Data:\n", df)

# Scatter plot
plt.figure(figsize=(6,4))
plt.scatter(df['X1'], df['X2'], c=df['label'],
cmap='coolwarm', edgecolors='black')
plt.xlabel("X1"); plt.ylabel("X2")
plt.title("Scatter Plot of Data")
plt.show()
```

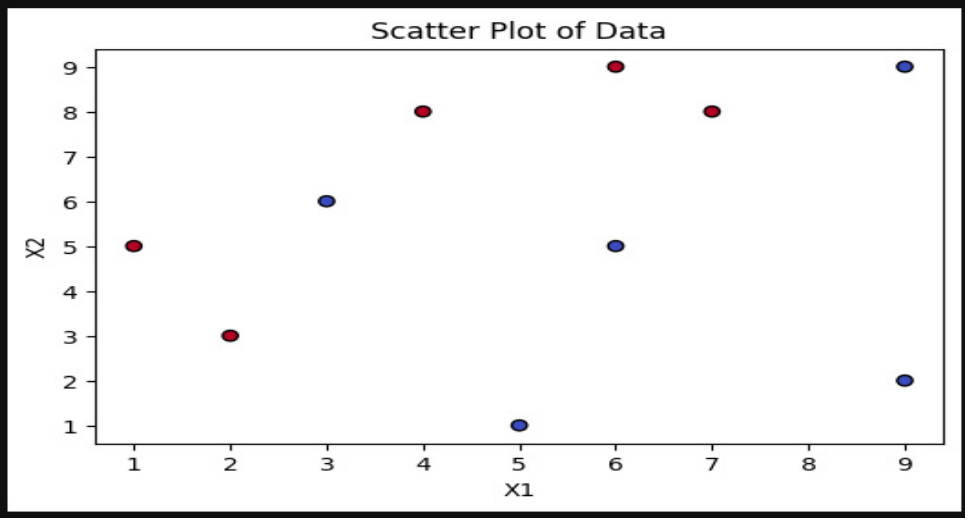
```
df['weights'] = 1/df.shape[0]
print("\nInitial Weights:\n", df)

x = df[['X1','X2']].values
y = df['label'].values

dt1 = DecisionTreeClassifier(max_depth=1)
dt1.fit(x, y)

plt.figure(figsize=(6,4))
plot_tree(dt1, filled=True,
feature_names=['X1','X2'], class_names=['0','1'])
plt.title("Decision Stump dt1")
plt.show() plot_decision_regions(x, y,

clf=dt1, legend=2) plt.title("Decision Region

dt1")
plt.show()

df['y_pred'] = dt1.predict(x)
print("\nAfter dt1 Predictions:\n",
df[['X1','X2','label','weights','y_pred']])

def calculate_alpha(error):
    return 0.5 * np.log((1-error)/error)

error1 = (df['label'] != df['y_pred']).mean()
alpha1 = calculate_alpha(error1)
print("\nError1 =", error1)
print("Alpha1 =", alpha1)
```

```
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)
print("\nUpdated Weights (Round 1):\n",
df[['X1','X2','label','weights','y_pred','updated_wei
ghts']])

# Normalized weights
df['normalized_weights'] =
df['updated_weights'] /
df['updated_weights'].sum()
print("\nNormalized Weights (Round 1):\n",
df[['X1','X2','label','normalized_weights']])
df['cumsum_upper'] =
np.cumsum(df['normalized_weights'])
df['cumsum_lower'] =
df['cumsum_upper'] -
df['normalized_weights']
print("\nCumsum Bounds (Round
1):\n",
df[['cumsum_lower','cumsum_upper']])

indices = []
for i in range(df.shape[0]):
    r = np.random.random()
```

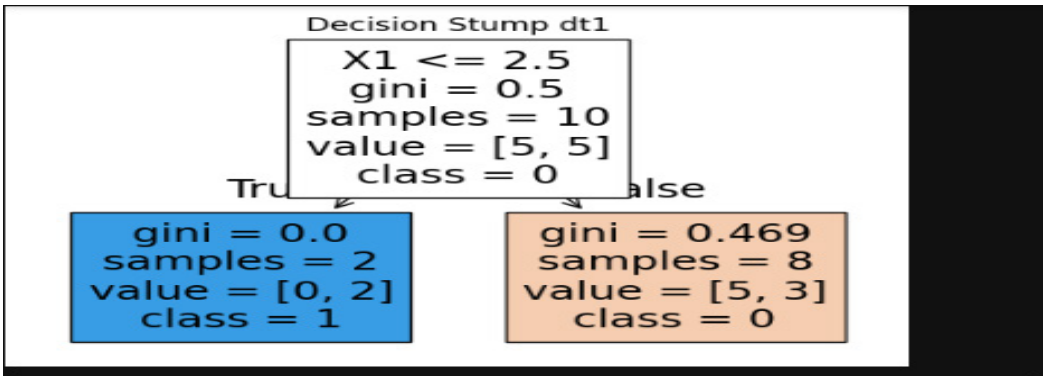
```
    for idx, row in df.iterrows():
        if row['cumsum_lower'] <= r <=
row['cumsum_upper']:
            indices.append(idx)
            break

second_df = df.iloc[indices].copy()
second_df['weights'] = 1/df.shape[0]
print("\nSecond Dataset (Weighted Sampling):\n",
second_df[['X1','X2','label','weights']])
x2 = second_df[['X1','X2']].values
y2 = second_df['label'].values
dt2 = DecisionTreeClassifier(max_depth=1)
dt2.fit(x2, y2)
plt.figure(figsize=(6,4))
plot_tree(dt2, filled=True,
feature_names=['X1','X2'], class_names=['0','1'])
plt.title("Decision Stump dt2")
plt.show()
plot_decision_regions(x2, y2, clf=dt2, legend=2)
plt.title("Decision Region dt2")
plt.show()
```

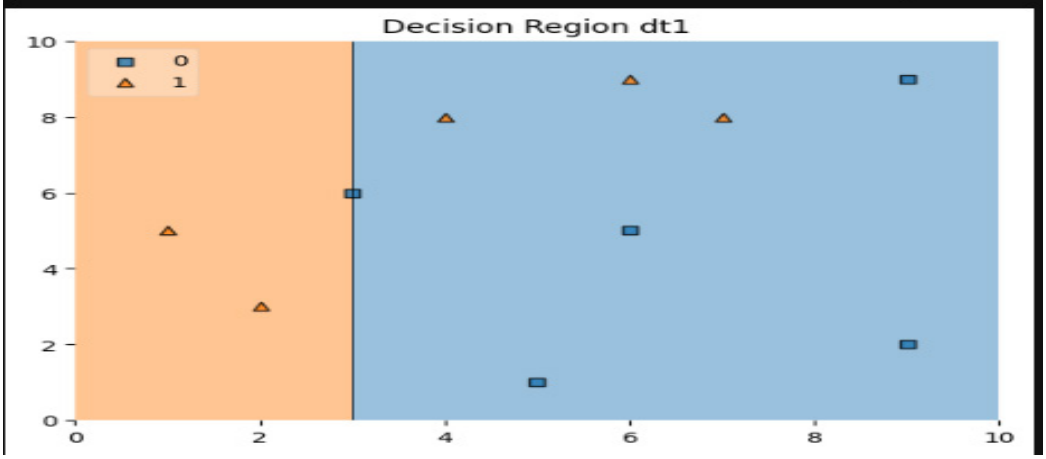
OUTPUT:

```
Original Data:
   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
```




```
Initial Weights:
   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
```





After dt1 Predictions:

	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

Error1 = 0.3
Alpha1 = 0.42364893019360184

Updated Weights (Round 1):

	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

Normalized Weights (Round 1):

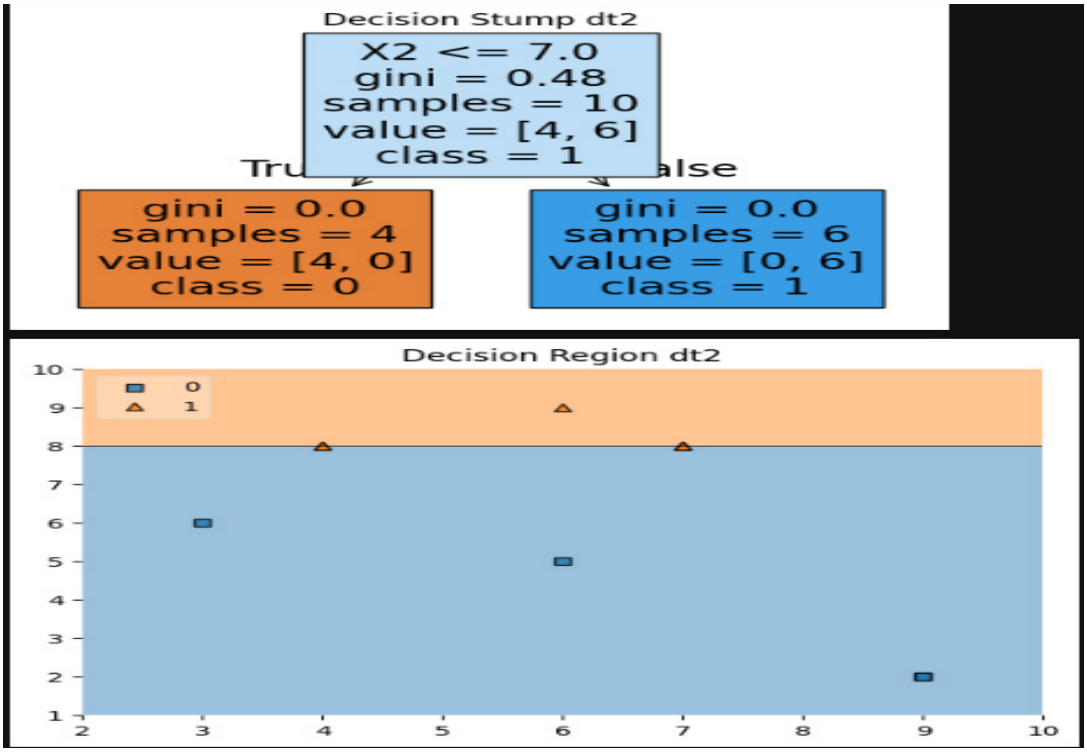
	X1	X2	label	normalized_weights
0	1	5	1	0.071429
1	2	3	1	0.071429
2	3	6	0	0.071429
3	4	8	1	0.166667
4	5	1	0	0.071429
5	6	9	1	0.166667
6	6	5	0	0.071429
7	7	8	1	0.166667
8	9	9	0	0.071429
9	9	2	0	0.071429

Cumsum Bounds (Round 1):

	cumsum_lower	cumsum_upper
0	0.000000	0.071429
1	0.071429	0.142857
2	0.142857	0.214286
3	0.214286	0.380952
4	0.380952	0.452381
5	0.452381	0.619048
6	0.619048	0.690476
7	0.690476	0.857143
8	0.857143	0.928571
9	0.928571	1.000000

Second Dataset (Weighted Sampling):

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



A PYTHON PROGRAM TO IMPLEMENT GRADIENT BOOSTING

Expt No. 8B

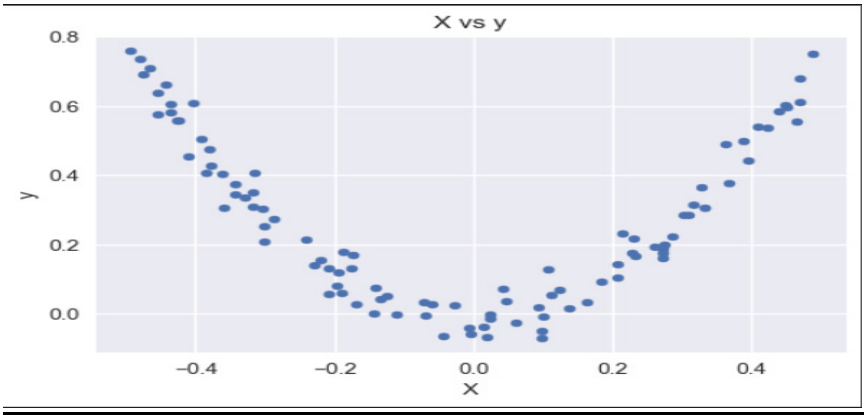
PROGRAM:

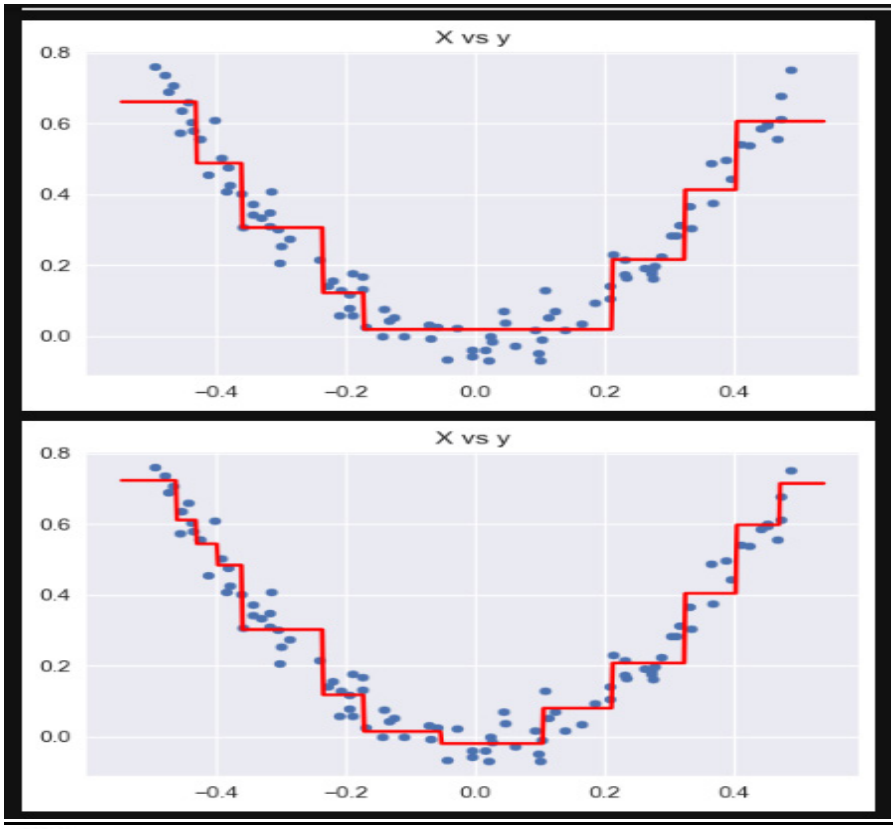
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

plt.style.use("seaborn-v0_8")
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({"X": X.reshape(-1), "y": y})

plt.figure(figsize=(6,4))
plt.scatter(df["X"], df["y"], s=25)
plt.title("X vs y")
plt.xlabel("X"); plt.ylabel("y")
plt.show()

def gradient_boost(X, y, n_estimators=5, lr=1.0, max_leaf_nodes=8,
random_state=42): rng = np.random.RandomState(random_state)

    X = np.asarray(X).reshape(-1, 1)
    y = np.asarray(y)

    F = np.full_like(y, y.mean(), dtype=float)
    trees = []
    xg = np.linspace(X.min() - 0.05, X.max() + 0.05, 600).reshape(-1, 1)
    curves = []
    stage_metrics = []
```

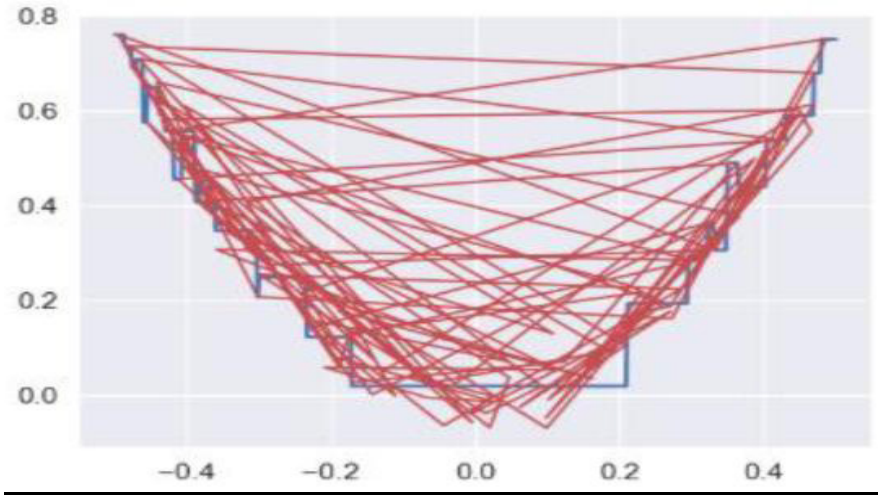
```
    for m in range(n_estimators):
        r =y-F
tree = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes,
random_state=rng.randint(0, 10**6))
        tree.fit(X, r)
        trees.append(tree)
        F += lr * tree.predict(X)

        Fg = np.full(xg.shape[0], y.mean(), dtype=float)
        for t in trees:
            Fg += lr * t.predict(xg)

        curves.append(Fg)

        mse = float(np.mean((y - F) ** 2))
        mae = float(np.mean(np.abs(y - F)))
        stage_metrics.append([m+1, mse, mae])

        plt.figure(figsize=(6,4))
        plt.scatter(X[:,0], y, s=25)
        plt.plot(xg[:,0], Fg, linewidth=2, color="red")
        plt.title("X vs y")
        plt.show()

return F, curves, xg, pd.DataFrame(stage_metrics, columns=["Stage",
"MSE", "MAE"])

final_pred, curves, xg, metrics = gradient_boost(X, y, n_estimators=5,
lr=1.0)

plt.figure(figsize=(6,4))
plt.scatter(df["X"], df["y"], s=25)
plt.plot(xg[:,0], curves[-1], linewidth=2, color="red")
plt.title("X vs y")
plt.xlabel("X"); plt.ylabel("y")
plt.show()

plt.figure(figsize=(6,4))
plt.scatter(df["X"], df["y"], s=20)
for c in curves:
plt.plot(xg[:,0], c, color="red", linewidth=1, alpha=0.8)
plt.title("All Boosting Stages (combined)")
```

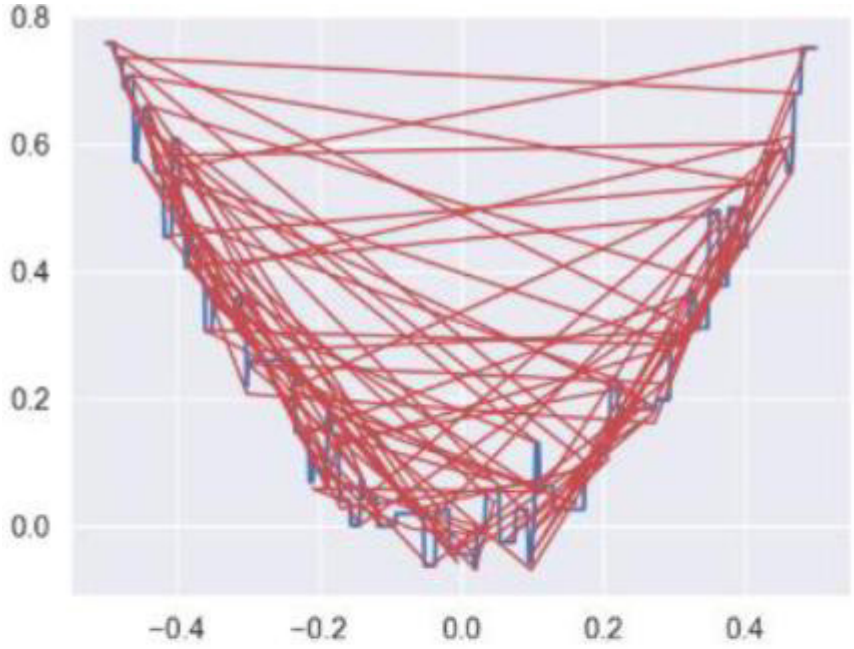
```
plt.xlabel("X"); plt.ylabel("y")
plt.show()

plt.figure(figsize=(6,4))
plt.scatter(df["X"], df["y"], s=20)
for c in curves:
    plt.plot(xg[:,0], c, color="red", linewidth=1)
plt.xlabel("X"); plt.ylabel("y")
plt.show()

plt.figure(figsize=(6,4))
plt.scatter(df["X"], df["y"], s=20)
for c in curves:
    plt.plot(xg[:,0], c, color="red", linewidth=1)
plt.xlabel("X"); plt.ylabel("y")
plt.show()

df_out = df.copy()
df_out["y_pred_final"] = final_pred
df_out["residual_final"] = df["y"] - df_out["y_pred_final"]

print("\nTABLE : Predictions & Residuals\n")
print(df_out.head(15))

print("\nTABLE : Stage Metrics (MSE, MAE per boosting stage)\n")
print(metrics)
```

OUTPUT:

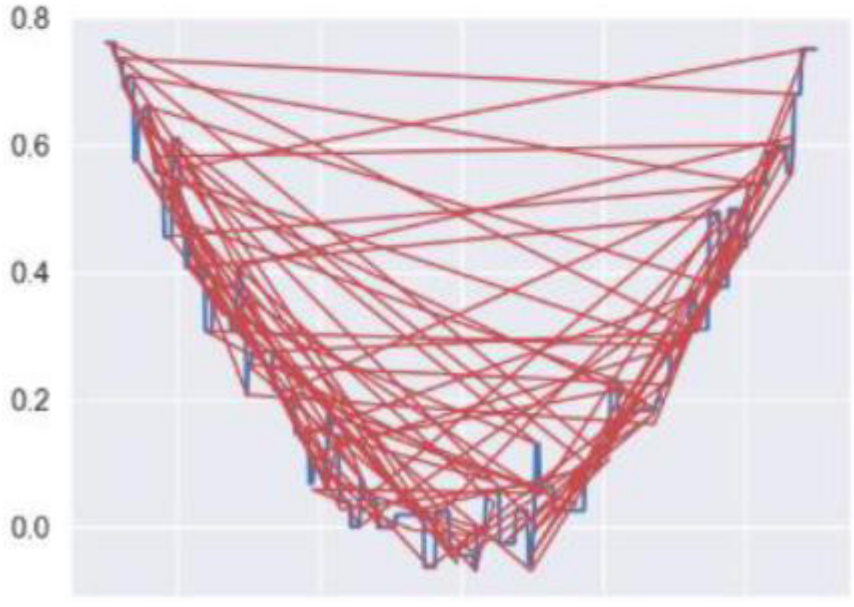


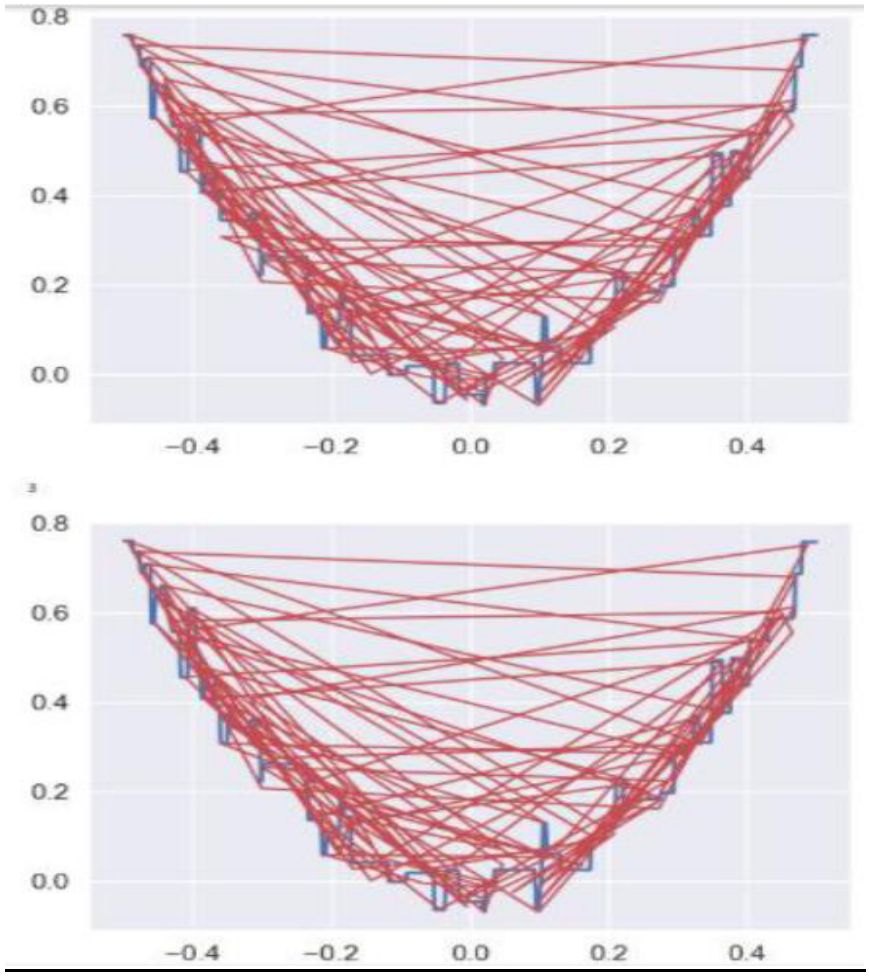

X vs y
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4
X vs y
0.8
0.6
0.4
0.2
0.0
−0.4
−0.2
0.0
0.2
0.4


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



1




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