# EXPERIMENT : 8(a)

# A python program to implement ada boost

## 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.

## CODE

```
import pandas as pd import numpy as np import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, plot_tree from

mlxtend.plotting import plot_decision_regions import

matplotlib.pyplot as plt


# 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] display(df)
sns.scatterplot(x=df['X1'], y=df['X2'], hue=df['label'], s=80)
plt.title("Initial Data Distribution") plt.show()

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

```
x = df.iloc[:, 0:2].values y = df.iloc[:, 2].values dt1 =
DecisionTreeClassifier(max_depth=1)

dt1.fit(x, y)

 plt.figure(figsize=(6,4)) plot_tree(dt1) plt.title("Decision Tree 1 (Weak
Classifier)") plt.show()

plot_decision_regions(x, y, clf=dt1, legend=2) plt.title("Decision
Boundary of 1st Weak Classifier") plt.show()
df['y_pred'] = dt1.predict(x) display(df)

 def calculate_model_weight(error):

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

 alpha1 = calculate_model_weight(0.3)

 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 r: update_row_weights(r,
alpha1), axis=1) df['normalized_weights'] = df['updated_weights'] /
df['updated_weights'].sum()
display(df[['X1','X2','label','weights','y_pred','normalized_weight s']])

print("Sum of normalized weights:",

df['normalized_weights'].sum())

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','normalized_weight

s','cumsum_lower','cumsum_upper']])

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) print("Sampled indices
for next dataset:", index_values)


second_df = df.iloc[index_values, [0,1,2,3]] display(second_df)

x2 = second_df.iloc[:,0:2].values y2 =
second_df.iloc[:,2].values dt2 =
DecisionTreeClassifier(max_depth=1) dt2.fit(x2,
y2)

plt.figure(figsize=(6,4)) plot_tree(dt2)
plt.title("Decision Tree 2 (Weak Classifier)")
plt.show()


plot_decision_regions(x2, y2, clf=dt2, legend=2) plt.title("Decision
Boundary of 2nd Weak Classifier") plt.show()
```

```
second_df['y_pred'] = dt2.predict(x2)
display(second_df) alpha2 =
calculate_model_weight(0.1) print("Alpha 2
=", alpha2)
query = np.array([1,5]).reshape(1,2)
print("\nQuery [1,5] predictions:")
print("dt1:", dt1.predict(query)) print("dt2:",
dt2.predict(query))
query2 = np.array([9,9]).reshape(1,2)
print("\nQuery [9,9] predictions:") print("dt1:",
dt1.predict(query2)) print("dt2:",
dt2.predict(query2)) OUTPUT 10:
```

Query [1,5] predictions:

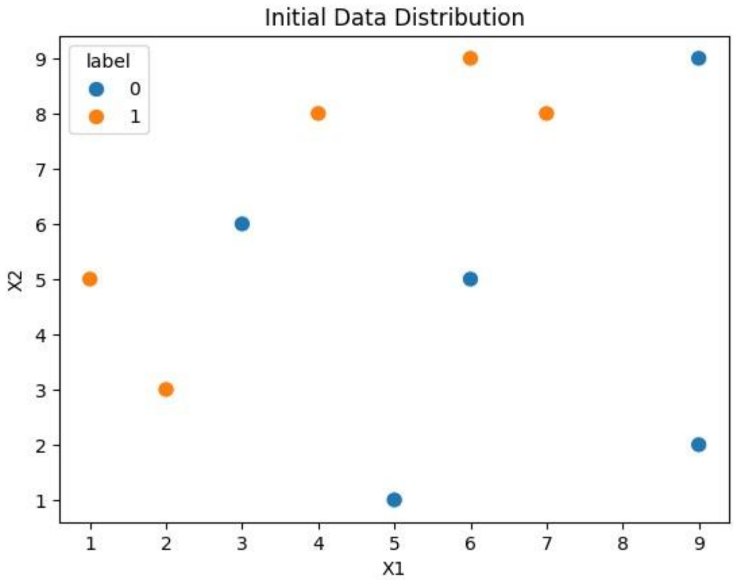
dt1: [1] dt2: [0]

Query [9,9] predictions:

dt1: [0] dt2: [1]

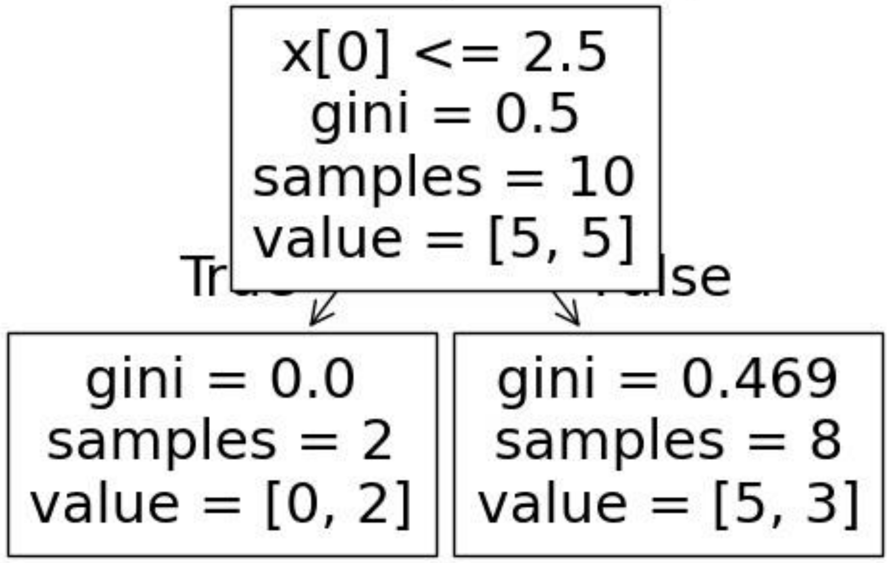
# OUTPUT :

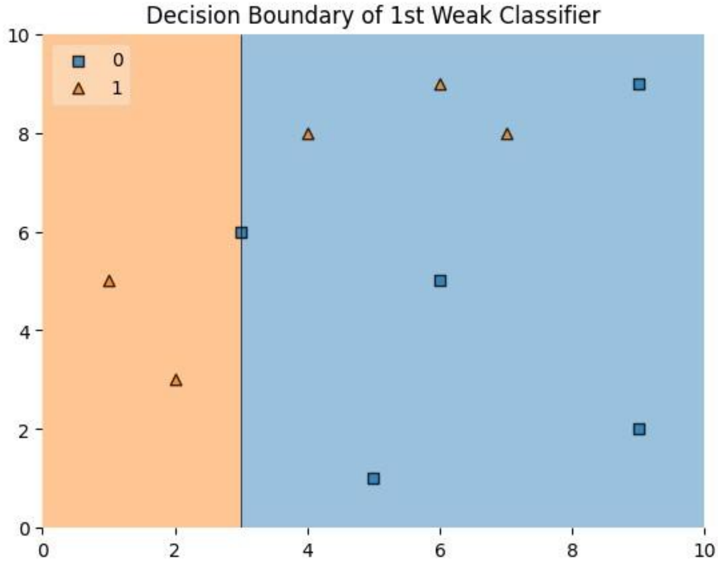
| | X1 | X2 | label |
|---|---|---|---|
| 0 | 1 | 5 | 1 |
| 1 | 2 | 3 | 1 |
| 2 | 3 | 6 | 0 |
| 3 | 4 | 8 | 1 |
| 4 | 5 | 1 | 0 |
| 5 | 6 | 9 | 1 |
| 6 | 6 | 5 | 0 |
| 7 | 7 | 8 | 1 |
| 8 | 9 | 9 | 0 |
| 9 | 9 | 2 | 0 |



| | 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 |

Decision Tree 1 (Weak Classifier)






| | 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 |

| | X1 | X2 | label | weights | y_pred | normalized_weights |
|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 1 | 0.1 | 1 | 0.071429 |
| 1 | 2 | 3 | 1 | 0.1 | 1 | 0.071429 |
| 2 | 3 | 6 | 0 | 0.1 | 0 | 0.071429 |
| 3 | 4 | 8 | 1 | 0.1 | 0 | 0.166667 |
| 4 | 5 | 1 | 0 | 0.1 | 0 | 0.071429 |
| 5 | 6 | 9 | 1 | 0.1 | 0 | 0.166667 |
| | X1 | X2 | label | weights | y_pred | normalized_weights |
| 6 | 6 | 5 | 0 | 0.1 | 0 | 0.071429 |
| 7 | 7 | 8 | 1 | 0.1 | 0 | 0.166667 |
| 8 | 9 | 9 | 0 | 0.1 | 0 | 0.071429 |
| 9 | 9 | 2 | 0 | 0.1 | 0 | 0.071429 |

Sum of normalized weights: 0.9999999999999999

## OUTPUT 7:

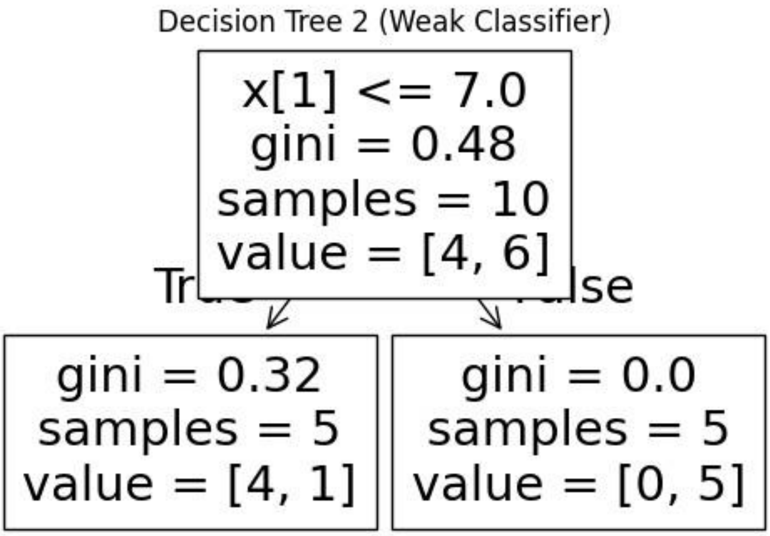
| | X1 | X2 | label | weights | y_pred | normalized_weights | cumsum_lower | cumsum_upper |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 5 | 1 | 0.1 | 1 | 0.071429 | 0.000000 | 0.071429 |
| 1 | 2 | 3 | 1 | 0.1 | 1 | 0.071429 | 0.071429 | 0.142857 |
| 2 | 3 | 6 | 0 | 0.1 | 0 | 0.071429 | 0.142857 | 0.214286 |
| 3 | 4 | 8 | 1 | 0.1 | 0 | 0.166667 | 0.214286 | 0.380952 |

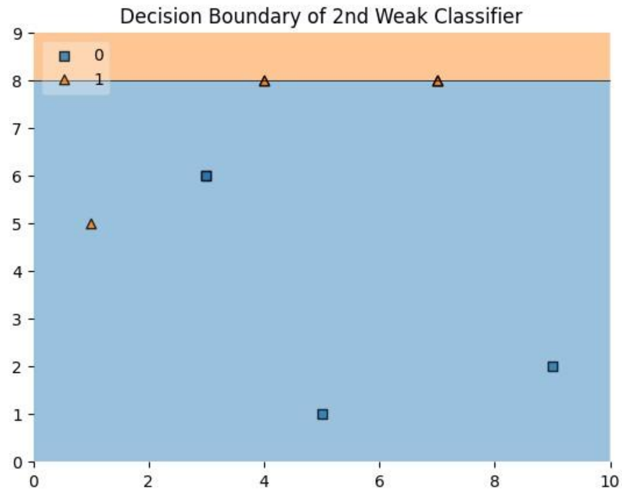
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | 5 | 1 | 0 | 0.1 | 0 | 0.071429 | 0.380952 | 0.452381 |
| 5 | 6 | 9 | 1 | 0.1 | 0 | 0.166667 | 0.452381 | 0.619048 |
| | X1 | X2 | label | weights | y_pred | normalized_weights | cumsum_lower | cumsum_upper |
| 6 | 6 | 5 | 0 | 0.1 | 0 | 0.071429 | 0.619048 | 0.690476 |
| 7 | 7 | 8 | 1 | 0.1 | 0 | 0.166667 | 0.690476 | 0.857143 |
| 8 | 9 | 9 | 0 | 0.1 | 0 | 0.071429 | 0.857143 | 0.928571 |
| 9 | 9 | 2 | 0 | 0.1 | 0 | 0.071429 | 0.928571 | 1.000000 |

Sampled indices for next dataset: [3, 2, 0, 2, 7, 9, 3, 7, 4, 7]

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

| | | | | |
|---|---|---|---|---|
| 7 | 7 | 8 | 1 | 0.1 |




| | X1 | X2 | label | weights | y_pred |
|---|---|---|---|---|---|
| 3 | 4 | 8 | 1 | 0.1 | 1 |
| 2 | 3 | 6 | 0 | 0.1 | 0 |
| 0 | 1 | 5 | 1 | 0.1 | 0 |
| 2 | 3 | 6 | 0 | 0.1 | 0 |
| 7 | 7 | 8 | 1 | 0.1 | 1 |

| | | | | | |
|---|---|---|---|---|---|
| 9 | 9 | 2 | 0 | 0.1 | 0 |
| 3 | 4 | 8 | 1 | 0.1 | 1 |
| 7 | 7 | 8 | 1 | 0.1 | 1 |
| 4 | 5 | 1 | 0 | 0.1 | 0 |
| 7 | 7 | 8 | 1 | 0.1 | 1 |

Alpha 2 = 1.0986122886681098

**RESULT:**

Thus a python program to implement ada boost is written and the output is verified.