

## **A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING**

**Ex.No.: 8**

**Date of Submission: 11/10/2024**

**AIM:-**

To implement a python program for Ada Boosting.

**ALGORITHM:-**

Step1: Import the necessary libraries(pandas as pd, numpy as np and plot\_decision\_regions from mlxtend.plotting)

Step2: Create a dataframe and fill values and labels in the data frame and display it.

Step3: Import seaborn as sns and plot a scatter plot with the data frame components as the parameters.

Step4: Add a new component to the data frame called “weights” which equals the inverse of the cumulative dimensions of the data frame and display it.

Step5: Import “DecisionTreeClassifier” from sklearn.tree and create an object.

Step6: Assign the variables “x” and “y” the range of values from the data frame.

Step7: Fit the first tree and then plot the tree using “plot\_tree” imported from sklearn.tree.

Step8: Plot the decision regions using the above trained tree as the classifier.

Step9: Introduce a new component in the dataframe called “y\_pred” to store the values predicted by the above use decision tree and display the decision tree.

Step10: Create a function which returns half the values of  $\log((1-\text{error})/(\text{error}))$  and calculate the weight of the decision tree.

Step11: Create a function to update the weights of the instances such that the weight is multiplied by  $\exp(-\alpha)$  if correctly classified and multiplied by  $\exp(\alpha)$  if misclassified.

Step12: Create a new component of the data frame called “updated\_weights” and apply the created function on the columns in the data frame and store the resulting values in the new component and display the data frame.

Step13: Add all the values in the “updated\_weights” component and add a new component called “normalized\_weights” which equals the division of each individual instance value by the sum of values of all instances and display the updated data frame.

Step14: Calculate the sum of the values of the “normalized\_values” component and display it.

Step15: Add a new component called “cumsum\_upper” the cumulative sum of the  
”normalized\_weights” values.

Step16: Add another component called “cumsum\_lower” which is the difference between the  
“cumsum\_upper” and “normalized\_weights” and display all the components of the data frame .

Step17: Follow the above 16 steps two more times for 2 new data frames and 2 new decision  
trees(second\_df,third\_df,dt2and dt3 respectively)

Step18: Compare the predicted values of all the decision trees.

Step19: Multiply alpha1, alpha2 and alpha3 by 1 and add all the values.

Step20: Find the sign of the resulting values from the previous step.

Step21: Multiply alpha1 by1, alpha2 and alpha3 by -1 and add the values and find the sign of the  
resulting value.

## **IMPLEMENTATION:-**

```
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]
```

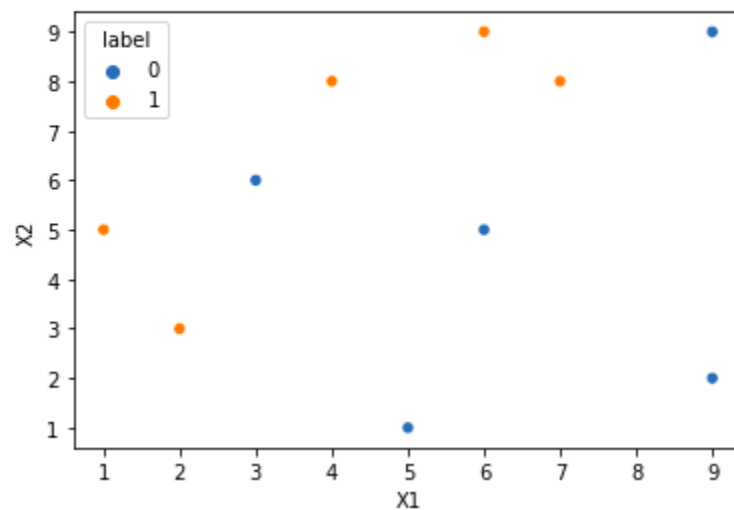
```
df['X2']=[5,3,6,8,1,9,5,8,9,2]
```

```
df['label']=[1,1,0,1,0,1,0,1,0,0]
```

```
df
```

|   | 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     |

```
import seaborn as sns
sns.scatterplot(x=df['X1'],y=df['X2'],hue=df['label'])
<AxesSubplot:xlabel='X1', ylabel='X2'>
```



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

```
df
```

|   | 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
```

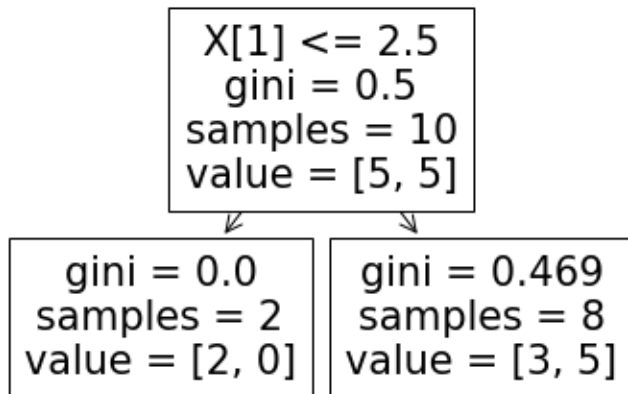
```
dt1.fit(x,y)
```

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

```
from sklearn.tree import plot_tree
```

```
plot_tree(dt1)
```

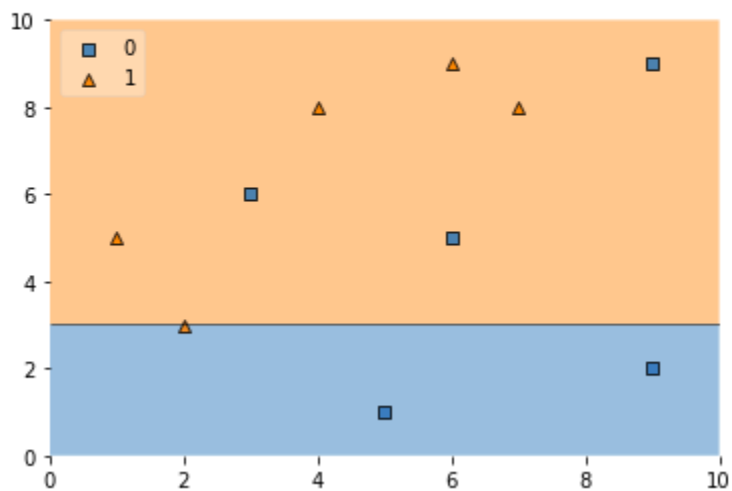
```
[Text(0.5, 0.75, 'X[1] <= 2.5\ngini = 0.5\nsamples = 10\nvalue = [5, 5]'),  
Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),  
Text(0.75, 0.25, 'gini = 0.469\nsamples = 8\nvalue = [3, 5]')]
```



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

```
J :
```

```
<AxesSubplot:>
```



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

```
df
```

|   | 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     | 1      |
| 3 | 4  | 8  | 1     | 0.1     | 1      |
| 4 | 5  | 1  | 0     | 0.1     | 0      |
| 5 | 6  | 9  | 1     | 0.1     | 1      |
| 6 | 6  | 5  | 0     | 0.1     | 1      |
| 7 | 7  | 8  | 1     | 0.1     | 1      |
| 8 | 9  | 9  | 0     | 0.1     | 1      |
| 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)
alpha1
```

```
0.42364893019360184
```

```
# Step -4 Update 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

|   | X1 | X2 | label | weights | y_pred | updated_weights |
|---|----|----|-------|---------|--------|-----------------|
| 0 | 1  | 5  | 1     | 0.1     | 1      | 0.065508        |
| 1 | 2  | 3  | 1     | 0.1     | 1      | 0.065508        |
| 2 | 3  | 6  | 0     | 0.1     | 1      | 0.152653        |
| 3 | 4  | 8  | 1     | 0.1     | 1      | 0.065508        |
| 4 | 5  | 1  | 0     | 0.1     | 0      | 0.065508        |
| 5 | 6  | 9  | 1     | 0.1     | 1      | 0.065508        |
| 6 | 6  | 5  | 0     | 0.1     | 1      | 0.152653        |
| 7 | 7  | 8  | 1     | 0.1     | 1      | 0.065508        |
| 8 | 9  | 9  | 0     | 0.1     | 1      | 0.152653        |
| 9 | 9  | 2  | 0     | 0.1     | 0      | 0.065508        |

df['updated\_weights'].sum()

0.9165153319682015

df['normalized\_weights']=df['updated\_weights']/df['updated\_weights'].sum()

df

|   | X1 | X2 | label | weights | y_pred | updated_weights | normalized_weights |
|---|----|----|-------|---------|--------|-----------------|--------------------|
| 0 | 1  | 5  | 1     | 0.1     | 1      | 0.065508        | 0.071475           |
| 1 | 2  | 3  | 1     | 0.1     | 1      | 0.065508        | 0.071475           |
| 2 | 3  | 6  | 0     | 0.1     | 1      | 0.152653        | 0.166559           |
| 3 | 4  | 8  | 1     | 0.1     | 1      | 0.065508        | 0.071475           |
| 4 | 5  | 1  | 0     | 0.1     | 0      | 0.065508        | 0.071475           |
| 5 | 6  | 9  | 1     | 0.1     | 1      | 0.065508        | 0.071475           |
| 6 | 6  | 5  | 0     | 0.1     | 1      | 0.152653        | 0.166559           |
| 7 | 7  | 8  | 1     | 0.1     | 1      | 0.065508        | 0.071475           |
| 8 | 9  | 9  | 0     | 0.1     | 1      | 0.152653        | 0.166559           |
| 9 | 9  | 2  | 0     | 0.1     | 0      | 0.065508        | 0.071475           |

```
df['normalized_weights'].sum()
```

```
|:
1.0
```

```
df['cumsum_upper'] = np.cumsum(df['normalized_weights'])
```

```
df['cumsum_lower']=df['cumsum_upper'] - df['normalized_weights']
```

```
df[['X1','X2','label','weights','y_pred','updated_weights','cumsum_lower','cumsum_upper']]
```



|   | X1 | X2 | label | weights | y_pred | updated_weights | cumsum_lower | cumsum_upper |
|---|----|----|-------|---------|--------|-----------------|--------------|--------------|
| 0 | 1  | 5  | 1     | 0.1     | 1      | 0.065508        | 0.000000     | 0.071475     |
| 1 | 2  | 3  | 1     | 0.1     | 1      | 0.065508        | 0.071475     | 0.142950     |
| 2 | 3  | 6  | 0     | 0.1     | 1      | 0.152653        | 0.142950     | 0.309508     |
| 3 | 4  | 8  | 1     | 0.1     | 1      | 0.065508        | 0.309508     | 0.380983     |
| 4 | 5  | 1  | 0     | 0.1     | 0      | 0.065508        | 0.380983     | 0.452458     |
| 5 | 6  | 9  | 1     | 0.1     | 1      | 0.065508        | 0.452458     | 0.523933     |
| 6 | 6  | 5  | 0     | 0.1     | 1      | 0.152653        | 0.523933     | 0.690492     |
| 7 | 7  | 8  | 1     | 0.1     | 1      | 0.065508        | 0.690492     | 0.761967     |
| 8 | 9  | 9  | 0     | 0.1     | 1      | 0.152653        | 0.761967     | 0.928525     |
| 9 | 9  | 2  | 0     | 0.1     | 0      | 0.065508        | 0.928525     | 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
```

```
[6, 6, 0, 6, 7, 5, 1, 8, 4, 6]
```

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

```
second_df
```

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

```
dt2 = DecisionTreeClassifier(max_depth=1)
```

```
x = second_df.iloc[:,0:2].values
```

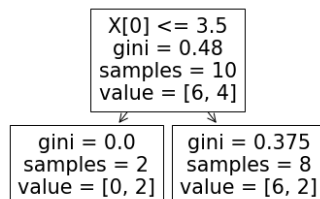
```
y = second_df.iloc[:,2].values
```

```
dt2.fit(x,y)
```

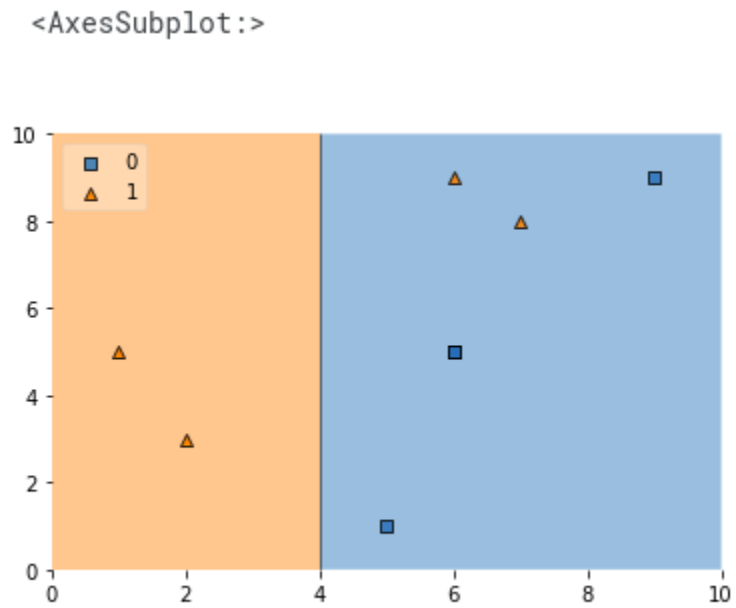
```
DecisionTreeClassifier(max_depth=1)
```

```
plot_tree(dt2)
```

```
[Text(0.5, 0.75, 'X[0] <= 3.5\ngini = 0.48\nsamples = 10\nvalue = [6, 4]'),
 Text(0.25, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.75, 0.25, 'gini = 0.375\nsamples = 8\nvalue = [6, 2]')]
```



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



```
second_df['y_pred'] = dt2.predict(x)
```

```
second_df
```

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

```
alpha2 = calculate_model_weight(0.1)
```

alpha2

1.0986122886681098

# Step 4 - Update weights

```
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

|   | X1 | X2 | label | weights | y_pred | updated_weights |
|---|----|----|-------|---------|--------|-----------------|
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.033622        |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.033622        |
| 0 | 1  | 5  | 1     | 0.1     | 1      | 0.033622        |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.033622        |
| 7 | 7  | 8  | 1     | 0.1     | 0      | 0.297427        |
| 5 | 6  | 9  | 1     | 0.1     | 0      | 0.297427        |
| 1 | 2  | 3  | 1     | 0.1     | 1      | 0.033622        |
| 8 | 9  | 9  | 0     | 0.1     | 0      | 0.033622        |
| 4 | 5  | 1  | 0     | 0.1     | 0      | 0.033622        |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.033622        |

```
second_df['nomalized_weights'].sum()
```

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']]
```

|   | 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     |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.038922           | 0.038922     | 0.077843     |
| 0 | 1  | 5  | 1     | 0.1     | 1      | 0.038922           | 0.077843     | 0.116765     |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 0.038922           | 0.116765     | 0.155687     |
| 7 | 7  | 8  | 1     | 0.1     | 0      | 0.344313           | 0.155687     | 0.500000     |
| 5 | 6  | 9  | 1     | 0.1     | 0      | 0.344313           | 0.500000     | 0.844313     |
| 1 | 2  | 3  | 1     | 0.1     | 1      | 0.038922           | 0.844313     | 0.883235     |
| 8 | 9  | 9  | 0     | 0.1     | 0      | 0.038922           | 0.883235     | 0.922157     |
| 4 | 5  | 1  | 0     | 0.1     | 0      | 0.038922           | 0.922157     | 0.961078     |
| 6 | 6  | 5  | 0     | 0.1     | 0      | 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
```

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

```
dt3 = DecisionTreeClassifier(max_depth=1)
```

```
X = second_df.iloc[:,0:2].values
```

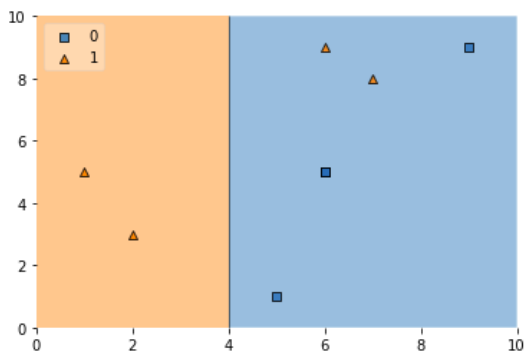
```
y = second_df.iloc[:,2].values
```

```
dt3.fit(X,y)
```

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

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

<AxesSubplot:>



```
third_df['y_pred'] = dt3.predict(X)
```

```
third_df
```

```
alpha3 = calculate_model_weight(0.7)
```

```
alpha3
```

```
-0.4236489301936017
```

```
print(alpha1,alpha2,alpha3)
```

```
0.42364893019360184 1.0986122886681098 -0.4236489301936017
```

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

```
dt1.predict(query)
```

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

```
dt2.predict(query)
```

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

```
dt3.predict(query)
```

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

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

```
1.09861228866811
```

```
np.sign(1.09)
```

```
.
```

```
1.0
```

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

```
dt1.predict(query)
```

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

```
dt2.predict(query)
```

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

```
dt3.predict(query)
```

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

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

```
-0.2513144282809062
```

```
np.sign(-0.25)
```

```
'1' -1.0
```

### **RESULT:-**

Thus the python program to implement Adaboosting has been executed successfully and the results have been verified and analyzed.