**APYTHONPROGRAMTOIMPLEMENTADABOOSTING**

**Ex.No.:8**

**Date of Experiment: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 of (1-error)/(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:-

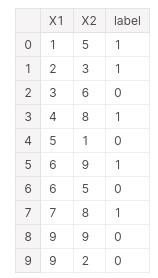
import pandas as pd

import numpy as np

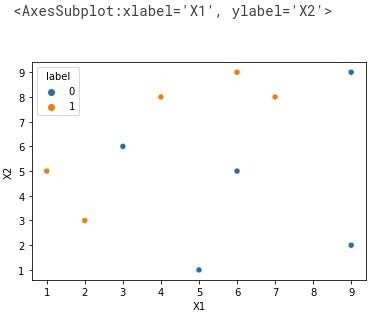
from mlxtend.plotting import plot\_decision\_regions

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

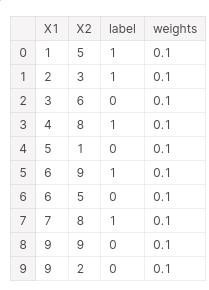


import seaborn as sns sns.scatterplot(x=df['X1'],y=df['X2'],hue=df['label'])



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

df



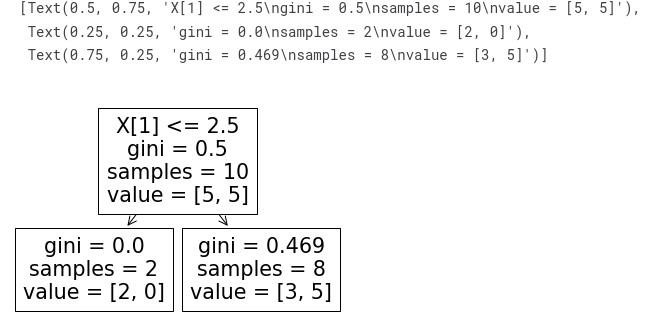
from sklearn.tree import DecisionTreeClassifier

dt1 = DecisionTreeClassifier(max\_depth=1) x = df.iloc[:,0:2].values y = df.iloc[:,2].values

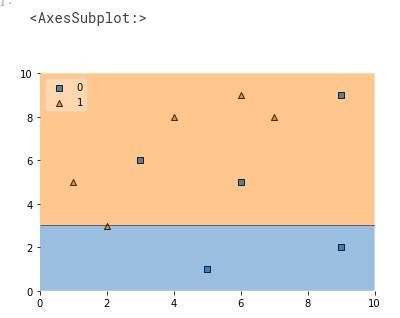
# Step 2 - Train 1st Model dt1.fit(x,y)



from sklearn.tree import plot\_tree plot\_tree(dt1)

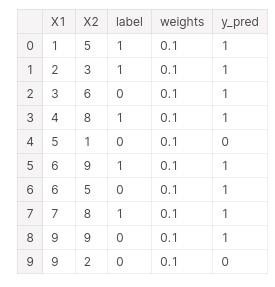


plot\_decision\_regions(x,y,clf=dt1,legend=2)



df['y\_pred'] = dt1.predict(x)

df



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



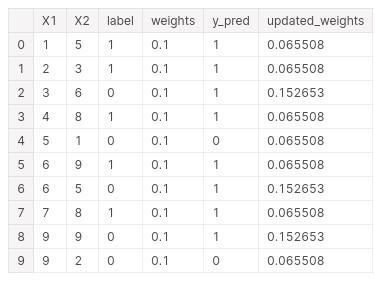
# Step -4 Update weights def update\_row\_weights(row,alpha=0.423): if row['label'] == row['y\_pred']:

return row['weights']\* np.exp(-alpha)

else: return row['weights']\*

np.exp(alpha)

df['updated\_weights'] = df.apply(update\_row\_weights,axis=1) df

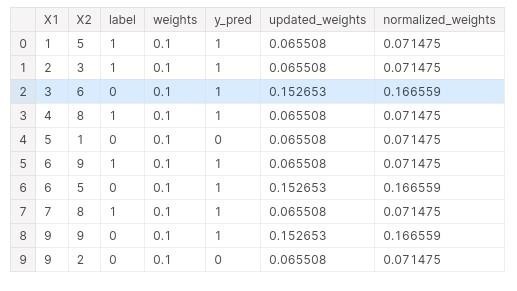


df['updated\_weights'].sum()



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

df



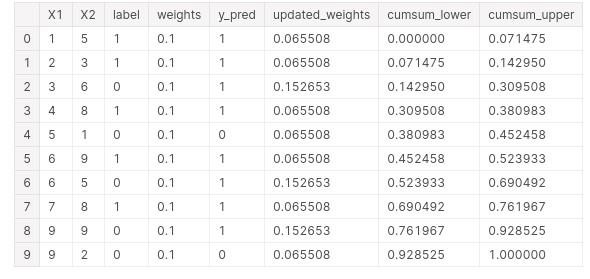
df['normalized\_weights'].sum()



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']]



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)

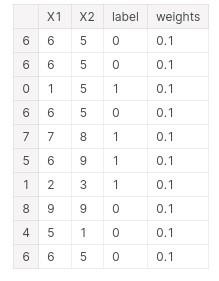
return indices

index\_values = create\_new\_dataset(df) index\_values



second\_df = df.iloc[index\_values,[0,1,2,3]]

second\_df

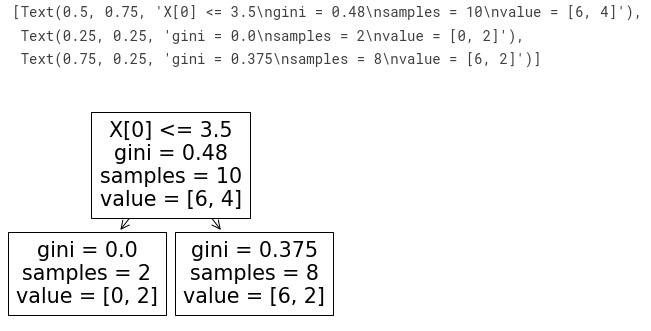


dt2 = DecisionTreeClassifier(max\_depth=1) x = second\_df.iloc[:,0:2].values y = second\_df.iloc[:,2].values

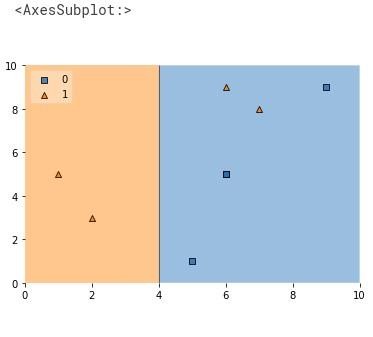
dt2.fit(x,y)



plot\_tree(dt2)

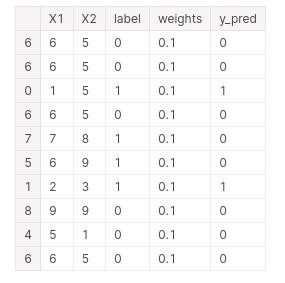


plot\_decision\_regions(x, y, clf=dt2, legend=2)



second\_df['y\_pred'] = dt2.predict(x)

second\_df



alpha2 = calculate\_model\_weight(0.1) alpha2



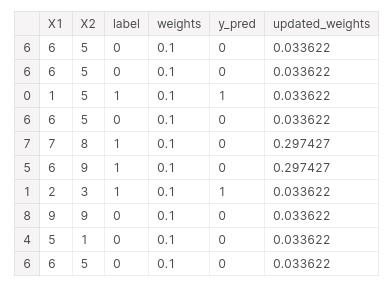
# Step 4 - Update weights def update\_row\_weights(row,alpha=1.09): if row['label'] == row['y\_pred']: return row['weights'] \* np.exp(-alpha)

else: return row['weights'] \*

np.exp(alpha)

second\_df['updated\_weights'] = second\_df.apply(update\_row\_weights,axis=1)

second\_df



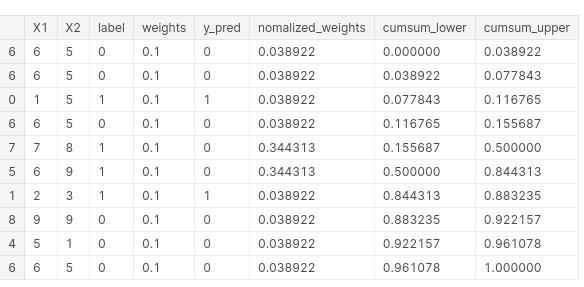
second\_df['nomalized\_weights'].sum()



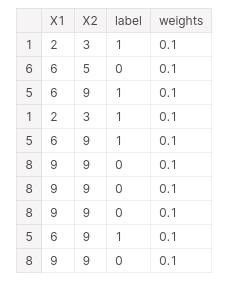
second\_df['cumsum\_upper'] = np.cumsum(second\_df['nomalized\_weights']) second\_df['cumsum\_lower'] = second\_df['cumsum\_upper'] - second\_df['nomalized\_weights']

second\_df[['X1','X2','label','weights','y\_pred','nomalized\_weights','cumsum\_lower','cumsum\_upp

er']]



index\_values = create\_new\_dataset(second\_df) third\_df = second\_df.iloc[index\_values,[0,1,2,3]] third\_df



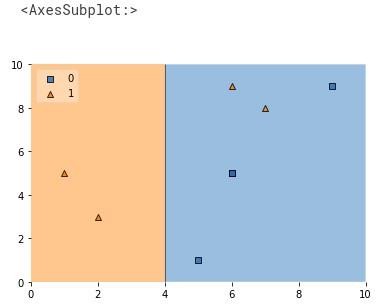
dt3 = DecisionTreeClassifier(max\_depth=1)

X = second\_df.iloc[:,0:2].values y = second\_df.iloc[:,2].values

dt3.fit(X,y)



plot\_decision\_regions(X, y, clf=dt3, legend=2)



third\_df[‘y\_pred] = dt3.predict(X)

third\_df



alpha3 = calculate\_model\_weight(0.7) alpha3



print(alpha1,alpha2,alpha3)



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

dt1.predict(query)



dt2.predict(query)



dt3.predict(query)



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



np.sign(1.09)



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

dt1.predict(query)



dt2.predict(query)



dt3.predict(query)



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



np.sign(-0.25)



# RESULT:-

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