**Ex. No.: 8 a.**

**Date: 11/10/24**

**A PYTHON PROGRAM TO IMPLEMENT ADA BOOSTING**

**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

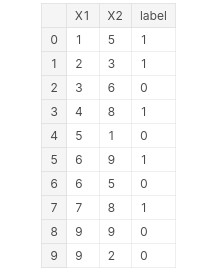
Compute the accuracy of the AdaBoost model on the testing set using accuracy\_score().

Step 7: Output Results

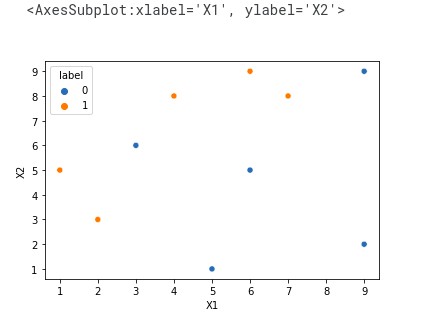
Print or plot the final accuracy and possibly other evaluation metrics.

**PROGRAM:**

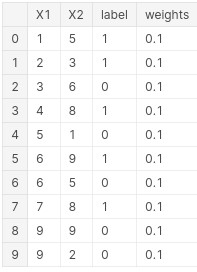
import pandas as pd import numpy as np from mlxtend.plotting import plot\_decision\_regions df = pd.DataFrame() df['X1']=[1,2,3,4,5,6,6,7,9,9] df['X2']=[5,3,6,8,1,9,5,8,9,2] df['label']=[1,1,0,1,0,1,0,1,0,0]



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



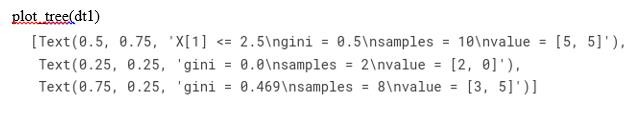
df['weights']=1/df.shape[0]

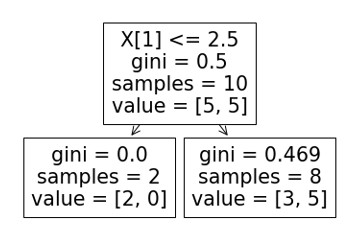


from sklearn.tree import DecisionTreeClassifier

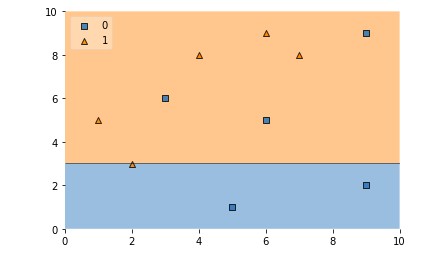
dt1 = DecisionTreeClassifier(max\_depth=1) x = df.iloc[:,0:2].values y = df.iloc[:,2].values # Step 2 - Train 1st Model dt1.fit(x,y)

from sklearn.tree import plot\_tree





**plot\_decision\_regions (x,yclf=dt1, legend=2)**



dfl'y pred'] = dt1.predict(x)



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']:

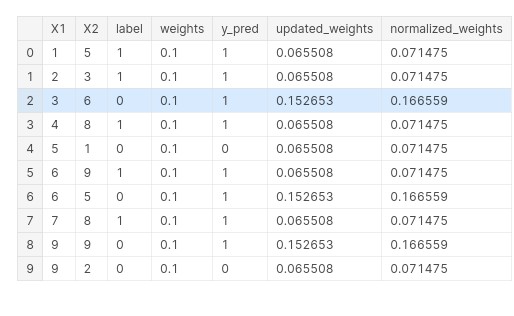
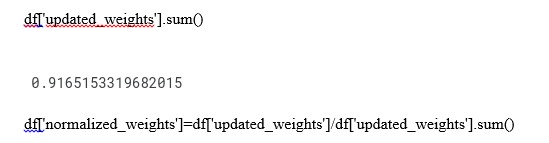
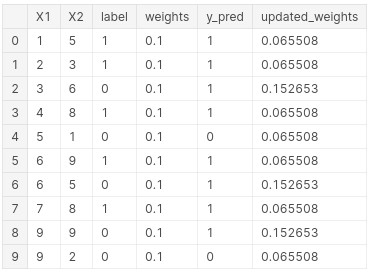
return row['weights']\* np.exp(-alpha)

else:

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

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

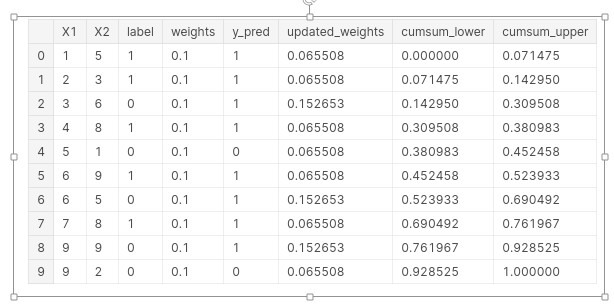
df['normalized\_weights'].sum()





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



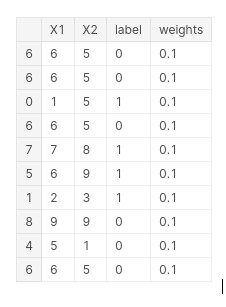
def create\_new\_dataset(df): indices= [] for i in range(df.shape[0]): a = np.random.random() for index,row in df.iterrows(): if row['cumsum\_upper']>a and a>row['cumsum\_lower']:

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



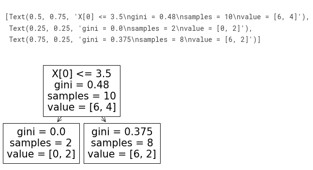
dt2 = DecisionTreeClassifier(max\_depth=1)

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

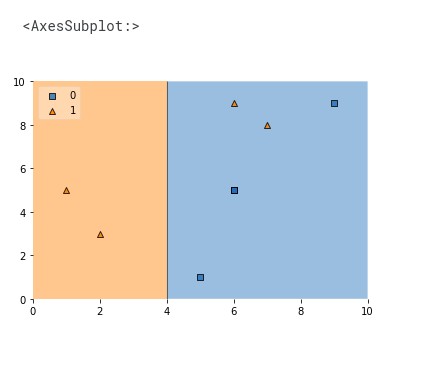
dt2.fit(x,y)



plot\_tree(dt2)

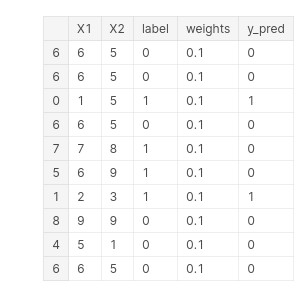


plot\_decision\_regions(x, y, clf=dt2, legend=2)



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

alpha2 = calculate\_model\_weight(0.1)



alpha2

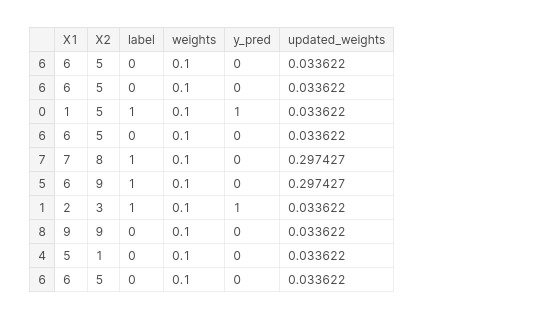


**# Step 4 - Update weights** def update\_row\_weights(row,alpha=1.09): if row['label'] == row['y\_pred']:

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

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

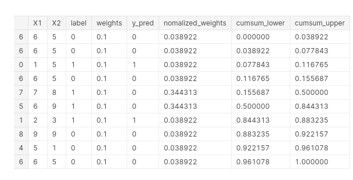
second\_df['updated\_weights'] = second\_df.apply(update\_row\_weights,axis=1) second\_df second\_df['nomalized\_weights'].sum()

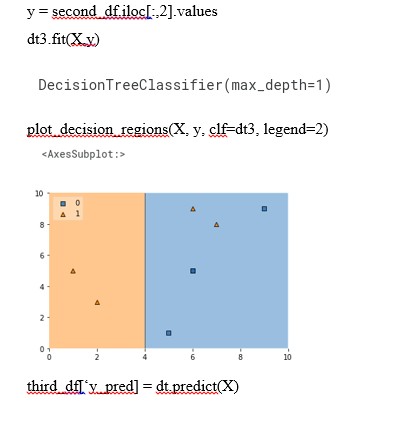
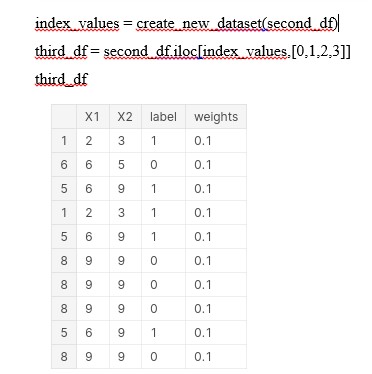


second\_df['nomalized\_weights'].sum()



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\_ upper']]





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.