Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJS23DSL402)

AY: 2024-25

Experiment 6

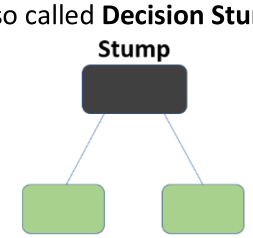
(AdaBoost)

Aim: Evaluate the performance of boosting algorithm (AdaBoost) with different base learners and hyperparameter tuning.

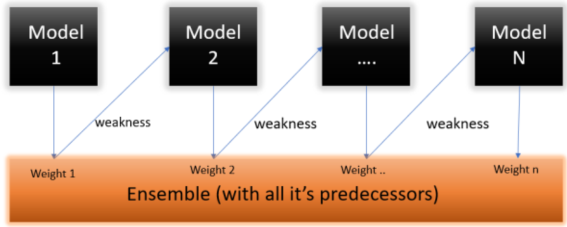
Theory:

Boosting algorithms improve the prediction power by **converting a number of weak learners to strong learners.** The principle behind boosting algorithms is first built a model on the training dataset, then a second model is built to rectify the errors present in the first model. This procedure is continued until and unless the errors are minimized, and the dataset is predicted correctly. Let's take an example to understand this, suppose you built a decision tree algorithm on the Titanic dataset and from there you get an accuracy of 80%. After this, you apply a different algorithm and check the accuracy and it comes out to be 75% for KNN and 70% for Linear Regression. The accuracy differs when we built a different model on the same dataset. But what if we use combinations of all these algorithms for making the final prediction? We'll get more accurate results by taking the average of results from these models. We can increase the prediction power in this way.

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called **Decision Stumps.**



Algorithm builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.



Step 1 – The Image is shown below is the actual representation of our dataset. Since the target column is binary it is a classification problem. First of all these data points will be assigned some weights. Initially, all the weights will be equal.

Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \ i = 1, 2, \ldots . n$$

Where N is the total number of datapoints. since we have 5 data points so the sample weights assigned will be 1/5.

Step 2 – We start by seeing how well "*Gender*" classifies the samples and will see how the variables (Age, Income) classifies the samples.

We'll create a decision stump for each of the features and then calculate the ***Gini Index*** of each tree. The tree with the lowest Gini Index will be our first stump.

Here in our dataset let's say ***Gender*** has the lowest gini index so it will be our first stump.

Step 3 – Calculate the **"Amount of Say"** or **"Importance"** or **"Influence"** for this classifier in classifying the datapoints using this formula:

$$\frac{1}{2} \log \frac{1 - Total\ Error}{Total\ Error}$$

The total error is nothing, but the summation of all the sample weights of misclassified data points.

Here in our dataset let's assume there is 1 wrong output, so our total error will be 1/5, and alpha(performance of the stump) will be:

$$Performance\ of\ the\ stump = \frac{1}{2}\log_e\left(\frac{1 - Total\ Error}{Total\ Error}\right)$$

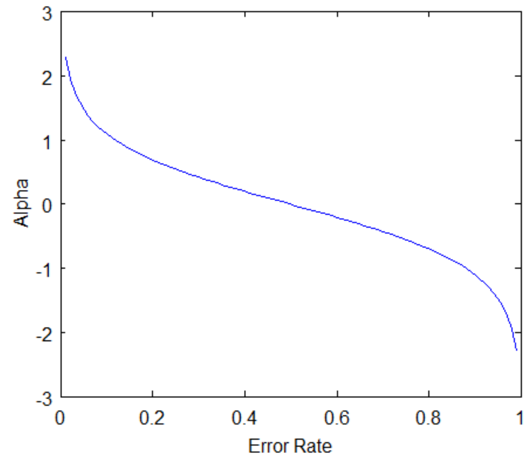
$$\alpha = \frac{1}{2}\log_e\left(\frac{1 - \frac{1}{5}}{\frac{1}{5}}\right)$$

$$\alpha = \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

$$\alpha = \frac{1}{2}\log_e(4) = \frac{1}{2} * (1.38)$$

$$\alpha = 0.69$$

Note: Total error will always be between 0 and 1.
0 Indicates perfect stump and 1 indicates horrible stump.



From the graph above we can see that when there is no misclassification then we have no error (Total Error = 0), so the "amount of say (alpha)" will be a large number.
When the classifier predicts half right and half wrong then the Total Error = 0.5 and the importance (amount of say) of the classifier will be 0. If all the samples have been incorrectly classified then the error will be very high (approx. to 1) and hence our alpha value will be a negative integer.
Step 4 –We need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model. The wrong predictions will be given more weight whereas the correct predictions weights will be decreased. Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.
After finding the importance of the classifier and total error we need to finally update the weights and for this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

The amount of say (alpha) will be ***negative*** when the sample is **correctly classified**.
The amount of say (alpha) will be ***positive*** when the sample is **miss-classified.**
There are four correctly classified samples and 1 wrong, here the ***sample weight*** of that datapoint is *1/5* and the ***amount of say/performance of the stump*** of ***Gender*** is *0.69*.

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(-0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 0.502\ =\ 0.1004$$

For *wrongly classified* samples the updated weights will be:

$$New\ sample\ weight\ =\ \frac{1}{5}\ *\ \exp(0.69)$$

$$New\ sample\ weight\ =\ 0.2\ *\ 1.994\ =\ 0.3988$$

Note: See the sign of alpha when I am putting the values, the **alpha is negative** when the data point is correctly classified, and this *decreases the sample weight* from 0.2 to 0.1004. It is **positive** when there is **misclassification**, and this will *increase the sample weight* from 0.2 to 0.3988

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

We know that the total sum of the sample weights must be equal to 1 but here if we sum up all the new sample weights, we will get 0.8004. To bring this sum equal to 1 we will normalize these weights by dividing all the weights by the total sum of updated weights that is 0.8004. So, after normalizing the sample weights we get this dataset and now the sum is equal to 1.

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

Step 5 – Now we need to make a new dataset to see if the errors decreased or not. For this we will remove the "sample weights" and "new sample weights" column and then based on the "new sample weights" we will divide our data points into buckets.

Row No.	Gender	Age	Income	Illness	New Sample Weights	Buckets
1	Male	41	40000	Yes	0.1004/0.8004= 0.1254	0 to 0.1254
2	Male	54	30000	No	0.1004/0.8004= 0.1254	0.1254 to 0.2508
3	Female	42	25000	No	0.1004/0.8004= 0.1254	0.2508 to 0.3762
4	Female	40	60000	Yes	0.3988/0.8004= 0.4982	0.3762 to 0.8744
5	Male	46	50000	Yes	0.1004/0.8004= 0.1254	0.8744 to 0.9998

Step 6 – We are almost done, now what the algorithm does is selects random numbers from 0-1. Since incorrectly classified records have higher sample weights, the probability to select those records is very high. Suppose the 5 random numbers our algorithm take is 0.38,0.26,0.98,0.40,0.55.
Now we will see where these random numbers fall in the bucket and according to it, we'll make our new dataset shown below.

Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

This comes out to be our new dataset and we see the datapoint which was wrongly classified has been selected 3 times because it has a higher weight.

Step 9 – Now this act as our new dataset and we need to repeat all the above steps i.e.

1. Assign ***equal weights*** to all the datapoints
2. Find the stump that does the ***best job classifying*** the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index
3. Calculate the ***"Amount of Say"*** and ***"Total error"*** to update the previous sample weights.
4. Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.
Suppose with respect to our dataset we have constructed 3 decision trees (DT1, DT2, DT3) in a ***sequential manner.*** If we send our **test data** now it will pass through all the decision trees and finally, we will see which class has the majority, and based on that we will do predictions for our test dataset.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Synthetic dataset
Dataset 2: CreditcardFraud.csv: The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, the original features and more background information about the data are not provided. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

1. Implement Decision Tree classifier and Logistic Regression on Dataset 1 using K fold cross validation and compare the results with AdaBoost classifier with base learner as Decision tree and Logistic Regression.
2. Check if there is class imbalance problem in Dataset 2. Compare the results of decision tree classifier and AdaBoost classifier on Dataset 2 and write your analysis.
3. Implement AdaBoost with base learner as decision tree on dataset 2 using K fold cross validation. Perform Hyperparameter tuning using (a) different depth, (b) different learning rate and (c) grid search CV. Show your results using Boxplot.

Write Ups:
1. Write the algorithm of AdaBoost.

xqgsasgaz

April 11, 2025

0.1 Dataset 1

- Name : Smayan Kulkarni
- Roll no : D100
- SAP ID : 60009230142
- Colab Link : https://colab.research.google.com/github/SmayanKulkarni/AI-and-ML-Course/blob/master/ML-LAB/ML_Lab_6.ipynb

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix

# Generate Synthetic Dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,␣
↪random_state=42)

# Define Models
dt = DecisionTreeClassifier()
lr = LogisticRegression(max_iter=1000)
ada_dt = AdaBoostClassifier(estimator=DecisionTreeClassifier(),␣
↪n_estimators=50,algorithm='SAMME')
ada_lr = AdaBoostClassifier(estimator=LogisticRegression(max_iter=1000),␣
↪n_estimators=50,algorithm='SAMME')

# K-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Evaluate Models
```

```
models = {'Decision Tree': dt, 'Logistic Regression': lr, 'AdaBoost (DT)':␣
 ↪ada_dt, 'AdaBoost (LR)': ada_lr}

for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
    print(f"{name} Accuracy: {scores.mean():.4f}")
```

```
Decision Tree Accuracy: 0.8110
Logistic Regression Accuracy: 0.8030
AdaBoost (DT) Accuracy: 0.8070
AdaBoost (LR) Accuracy: 0.7930
```

0.2 Dataset 2

[ ]:
```
df2 = pd.read_csv('creditcard.csv')
```

[ ]:
```
df2
```

[ ]:
```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…            …         …          …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…            …         …         …         …  …        …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
```

```
2        0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3       -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4       -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…            …         …         …         …         …         …      …
284802   1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803   0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804  -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805  -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806   0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
…       …
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

[ ]:
```python
df2['Class'].value_counts()
```

[ ]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

Here we can see that there is a huge imbalance in the dataset. The number of fraudulent transactions is very small compared to the number of non-fraudulent transactions. This is a common issue in fraud detection datasets. Therefore now the model trained on this dataset with be biased towards the output being '0' which means that the transaction was not fraudulent and this may lead to quite a lot of incorrect classifications.

[ ]:
```python
ada = AdaBoostClassifier(algorithm='SAMME')
```

[ ]:
```python
X = df2.drop('Class', axis=1)
y = df2['Class']
```

[ ]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
ada.fit(X_train, y_train)
```

[ ]:
```
AdaBoostClassifier(algorithm='SAMME')
```

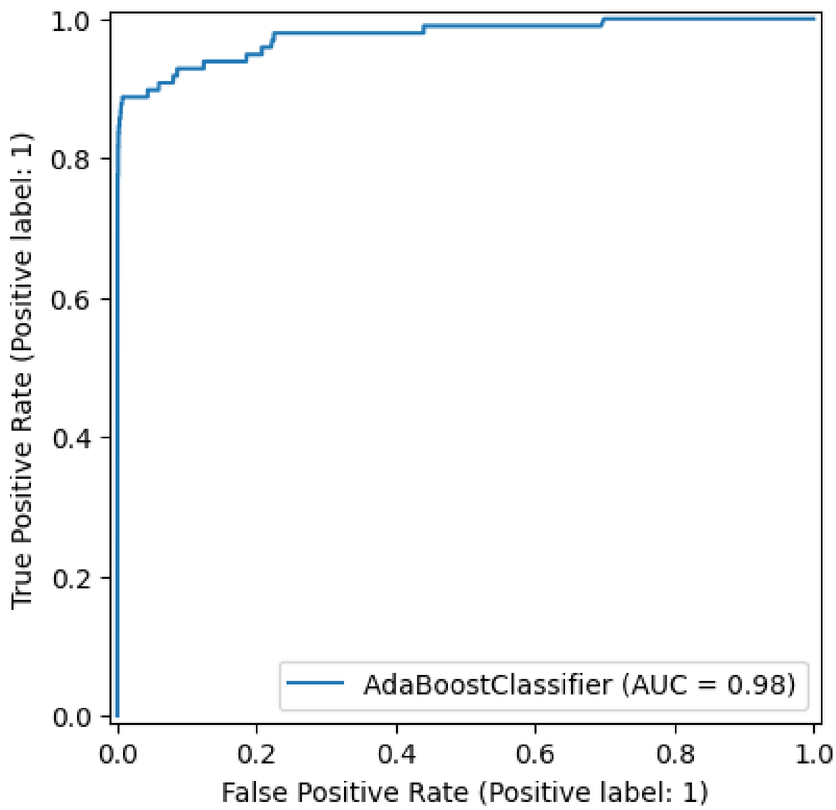
[ ]:
```
y_pred = ada.predict(X_test)
```

[ ]:
```
y_pred
```

[ ]:
```
array([1, 0, 0, …, 0, 0, 0])
```

[ ]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪RocCurveDisplay
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
RocCurveDisplay.from_estimator(ada, X_test, y_test)
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.79      0.68      0.73        98

    accuracy                           1.00     56962
   macro avg       0.89      0.84      0.87     56962
weighted avg       1.00      1.00      1.00     56962

[[56846    18]
 [   31    67]]
```



[ ]:
```
tree = DecisionTreeClassifier()
```

[ ]:
```
tree.fit(X_train, y_train)
```

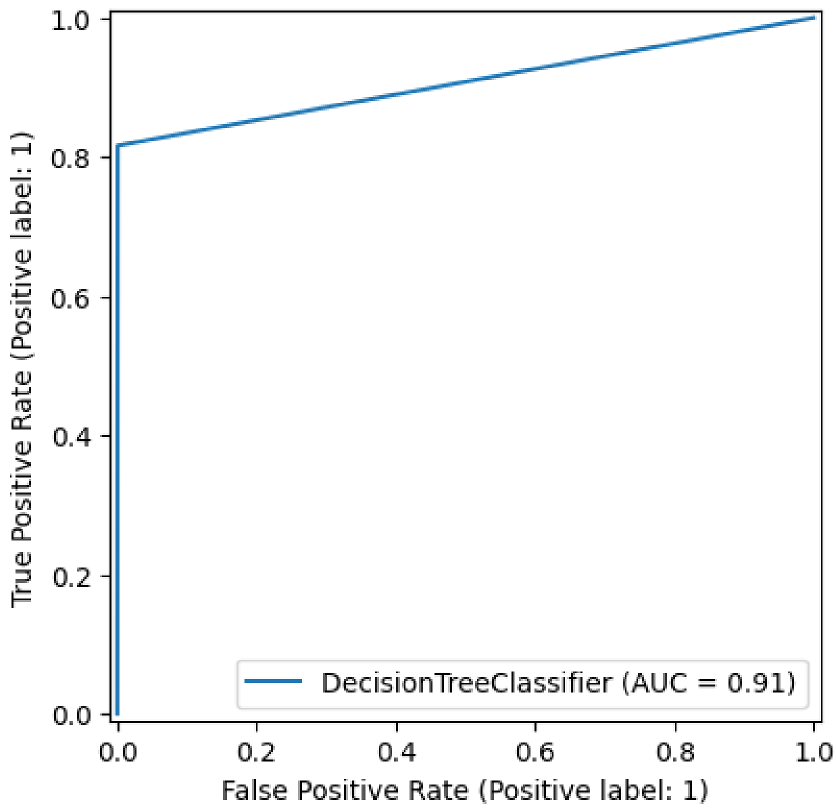
[ ]:
```
DecisionTreeClassifier()
```

[ ]:
```
tree_pred = tree.predict(X_test)
```

[ ]:
```
print(classification_report(y_test, tree_pred))
print(confusion_matrix(y_test, tree_pred))
RocCurveDisplay.from_estimator(tree, X_test, y_test)
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.73      0.82      0.77        98

    accuracy                           1.00     56962
   macro avg       0.87      0.91      0.89     56962
weighted avg       1.00      1.00      1.00     56962
```

```
[[56835    29]
 [   18    80]]
```



From the above, we can see that the AdaBoost classifier has a better performance than the Decision Tree classifier. The AdaBoost classifier has a higher precision and recall, which means it is better at identifying fraudulent transactions. The confusion matrix also shows that the AdaBoost classifier has fewer false positives and false negatives compared to the Decision Tree classifier.m

Adaboost with base learning as Decision Tree Classifier

[ ]:
```
ada_tree = AdaBoostClassifier(estimator=DecisionTreeClassifier(),algorithm='SAMME')
```

[ ]:
```
cross = KFold(n_splits=6, shuffle=True, random_state=2384)
```

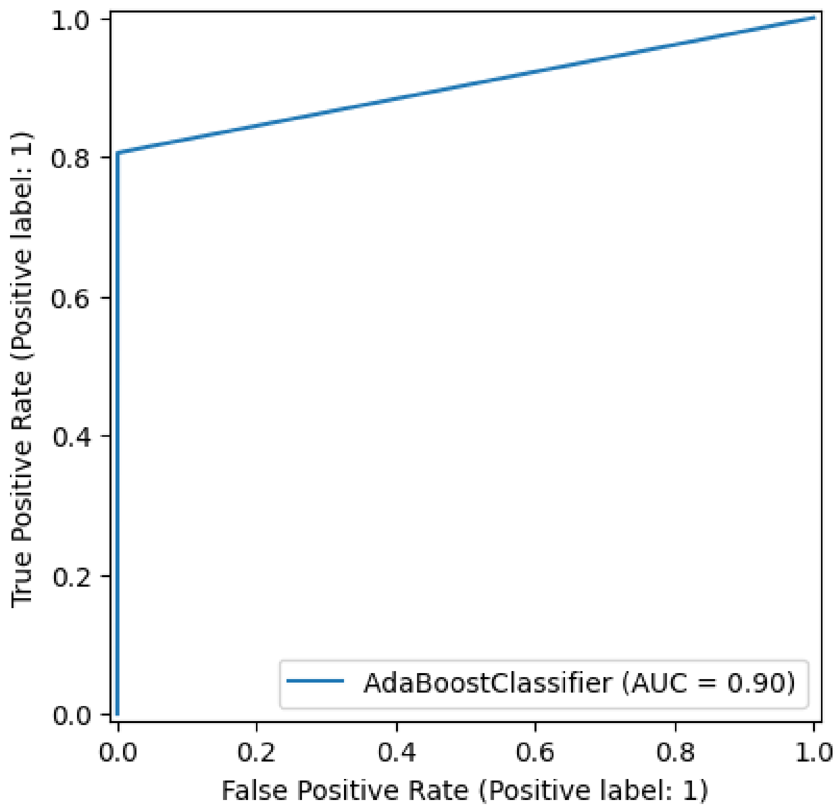
[ ]:
```
ada_tree.fit(X_train, y_train)
```

[ ]:
```
AdaBoostClassifier(algorithm='SAMME', estimator=DecisionTreeClassifier())
```

[ ]:
```
ada_tree_preds = ada_tree.predict(X_test)
```

[ ]:
```
print(classification_report(y_test, ada_tree_preds))
print(confusion_matrix(y_test, ada_tree_preds))
RocCurveDisplay.from_estimator(ada_tree, X_test, y_test)
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.72      0.81      0.76        98

    accuracy                           1.00     56962
   macro avg       0.86      0.90      0.88     56962
weighted avg       1.00      1.00      1.00     56962

[[56834    30]
 [   19    79]]
```



[ ]:
```
cross_val_score(ada_tree, X, y, cv=cross, scoring='accuracy').mean()
```

[ ]: 0.9991397672863834

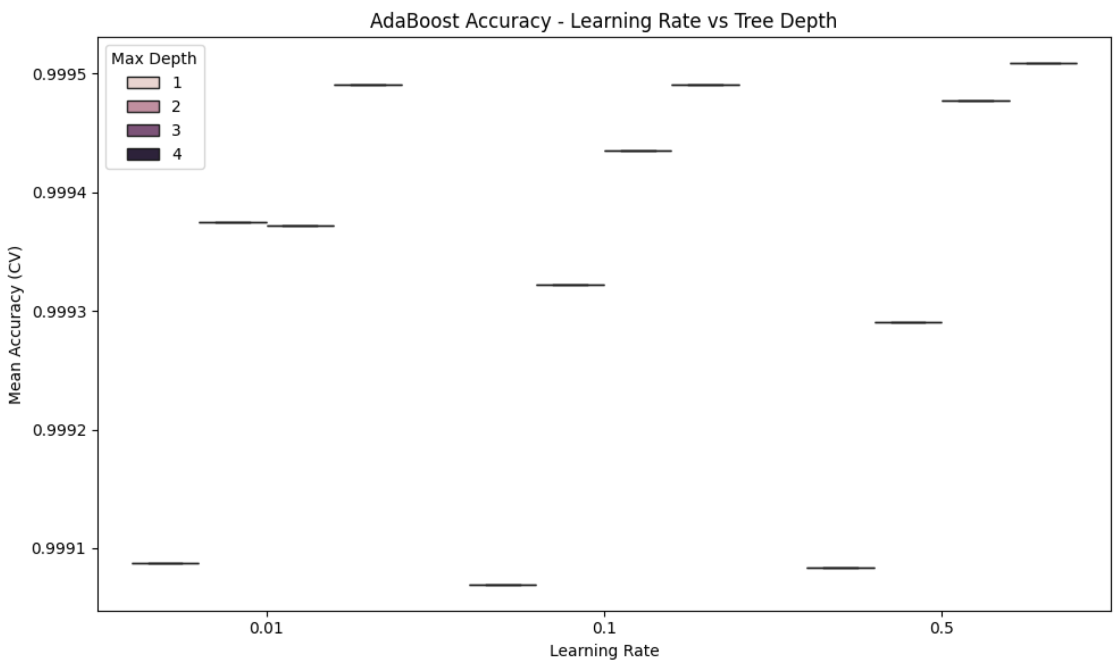
[ ]:
```python
from sklearn.model_selection import GridSearchCV
param_grid = {
    'estimator__max_depth': [1, 2, 3,4],
    'learning_rate': [0.01, 0.1, 0.5]
}

grid_search = GridSearchCV(estimator=ada_tree,
                           param_grid=param_grid,
                           cv=cross,
                           scoring='accuracy',
                           n_jobs=-1,
                           return_train_score=True)

grid_search.fit(X, y)
```

```
/home/smayan/Desktop/AI-ML-DS/AI-and-ML-Course/.conda/lib/python3.11/site-
packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in
cast
  _data = np.array(data, dtype=dtype, copy=copy,
```

[ ]:
```
GridSearchCV(cv=KFold(n_splits=6, random_state=2384, shuffle=True),
             estimator=AdaBoostClassifier(algorithm='SAMME',
                                          estimator=DecisionTreeClassifier()),
             n_jobs=-1,
             param_grid={'estimator__max_depth': [1, 2, 3, 4],
                         'learning_rate': [0.01, 0.1, 0.5]},
             return_train_score=True, scoring='accuracy')
```

[ ]:
```python
results = pd.DataFrame(grid_search.cv_results_)
```

[ ]:
```python
plt.figure(figsize=(10, 6))
sns.boxplot(x='param_learning_rate', y='mean_test_score',␣
 ↪hue='param_estimator__max_depth', data=results)
plt.title('AdaBoost Accuracy - Learning Rate vs Tree Depth')
plt.xlabel('Learning Rate')
plt.ylabel('Mean Accuracy (CV)')
plt.legend(title='Max Depth')
plt.tight_layout()
plt.show()
```


AdaBoost Accuracy - Learning Rate vs Tree Depth
Max Depth
1
2
3
4
0.9995
0.9994
0.9993
0.9992
0.9991
Mean Accuracy (CV)
0.01
0.1
0.5
Learning Rate

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: ML

Semester: IV

AY: 2024-25

Experiment No: 6

AIM: Evaluate the performance of boosting Algorithm (AdaBoost) with different base learners and hyperparameter tuning

Write-Up:

AdaBoost is an ensemble learning technique that combines multiple weak classifiers to create a strong classifier. It works iteratively by focusing on hard to classify data points.

⇒ Algorithm :-

I) Initialize weights
→ Assign equal weights to all training data points.

$$w_i = \frac{1}{N}$$

II) Train weak classifiers :-
→ train a weak classifier on dataset using current weights
→ Evaluate using weighted error rate:

$$\text{Error} = \sum_{i=1}^{N} w_i \cdot I(\text{prediction} \neq y_i)$$

where I is an indicator function that equals 1 for misclassified points and 0 otherwise.

III] Compute Classifier Importance:

→ Calculate importance of weak classifier based on error rate (α)

$$\alpha = \frac{1}{2} \ln \left(\frac{1 - \text{Error}}{\text{Error}} \right)$$

IV] Update weights:

→ Increase weights for misclassified points and decrease for correctly classified points.

$$w_i = w_i \cdot e^{\pm \alpha}$$

$\{+\alpha$: misclassified ; $-\alpha$: correctly classified$\}$

V] Combine weak classifiers

$$G(x) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m f_m(x) \right)$$

where $f_m(x)$ is the prediction of the m^{th} classifier and α_m its weight

VI] Repeat until Stopping Criteria:-

→ Continue training weak classifiers until predefined number of iteration is reached or error becomes negligible.