Experiment No. 6
Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:4/10/23
Date of Submission:11/10/23

Aim: Apply Boosting algorithm on Adult Census Income Dataset and analyze the performance of the model.

Objective: Apply Boosting algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

Theory:

Suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnosis they have made. The final diagnosis is then a combination of the weighted diagnosis. This is the essence behind boosting.

Algorithm: Adaboost- A boosting algorithm—creates an ensemble of classifiers. Each one

gives a weighted vote.

Input:

- D , a set of d class labelled training tuples
- k, the number of rounds (one classifier is generated per round)
- a classification learning scheme

Output: A composite model

Method

1. Initialize the weight of each tuple in D is 1/d
2. For i=1 to k do // for each round
3. Sample D with replacement according to the tuple weights to obtain D_i
4. Use training set D_i to derive a model M_i
5. Computer error(M_i), the error rate of M_i
6. Error(M_i)=$\sum w_j * err(X_j)$
7. If Error(M_i)>0.5 then
8. Go back to step 3 and try again
9. endif
10. for each tuple in D_i that was correctly classified do
11. Multiply the weight of the tuple by error(Mi)/(1-error(M_i)
12. Normalize the weight of each tuple
13. end for

To use the ensemble to classify tuple X

1. Initialize the weight of each class to 0
2. for i=1 to k do // for each classifier
3. $w_i=\log((1-error(M_i))/error(M_i))$//weight of the classifiers vote
4. $C=M_i(X)$ // get class prediction for X from M_i
5. Add w_i to weight for class C
6. end for
7. Return the class with the largest weight.

Dataset:

Predict whether income exceeds $50K/yr based on census data. Also known as "Adult" dataset.
Attribute Information:
Listing of attributes:

>50K, <=50K.

age: continuous.
workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.

fnlwgt: continuous.
education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.

education-num: continuous.

marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.

occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.

race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.

sex: Female, Male.

capital-gain: continuous.

capital-loss: continuous.

hours-per-week: continuous.

native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad &Tobago, Peru, Hong, Holand-Netherlands.

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = pd.read_csv('adult.csv')



# Handling missing values (if any)
data = data.dropna()

# Encode categorical variables using one-hot encoding
data = pd.get_dummies(data, columns=['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.cou

# Split the data into features and target variable
X = data.drop('income_>50K', axis=1)
y = data['income_>50K']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Initialize the AdaBoostClassifier
ada_classifier = AdaBoostClassifier(n_estimators=50, random_state=42)

# Fit the model to the training data
ada_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = ada_classifier.predict(X_test)



# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.86
```

```
# Create a confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(confusion)
```

```
Confusion Matrix:
[[4693  283]
 [ 629  908]]
```

```
# Generate a classification report
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4976
           1       0.76      0.59      0.67      1537

    accuracy                           0.86      6513
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513
```

Conclusion:

1. Comment on the accuracy, confusion matrix, precision, recall and F1 score obtained.

   Accuracy: Accuracy measures the overall correctness of the model's predictions. It's the ratio of correctly predicted instances to the total number of instances. In this case, the accuracy score tells us how well the model performs in predicting income levels.

   High accuracy suggests that the model is making correct predictions for most of the data.

   Low accuracy indicates that the model may have difficulty making accurate predictions.

   Accuracy is 0.86

   Confusion Matrix: The confusion matrix provides a more detailed view of the model's performance by breaking down predictions into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

   Confusion Matrix:

   [[4693 283]

   [629 908]]

   Precision: Precision is the ratio of true positives to the total predicted positives (true positives + false positives). It measures the model's ability to make accurate positive predictions.

   High precision means that when the model predicts ">50K" income, it is often correct.

   Low precision indicates that the model has a high rate of false positives when predicting ">50K" income.

   precision is 0.88

   Recall (Sensitivity): Recall is the ratio of true positives to the total actual positives (true positives + false negatives). It measures the model's ability to capture all positive instances.

   Recall 0.94

   F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall, especially when dealing with imbalanced datasets.

A high F1 score indicates a good balance between precision and recall.

A low F1 score suggests that one of the two metrics (precision or recall) dominates the other.

High recall means that the model is effective at identifying ">50K" income instances.

Low recall suggests that the model misses many ">50K" income instances.

F1 score is 0.91

2. Compare the results obtained by applying boosting and random forest algorithm on the Adult Census Income Dataset

AdaBoost (Boosting Algorithm):

Accuracy: AdaBoost tends to produce a relatively high accuracy because it focuses on improving the performance of weak learners by sequentially giving more weight to misclassified samples. However, the exact accuracy can vary based on hyperparameters and data preprocessing.

Precision and Recall: AdaBoost typically achieves a good balance between precision and recall. It's capable of identifying both positive and negative cases effectively.
F1 Score: AdaBoost's F1 score is generally competitive and balanced.
Computational Complexity: AdaBoost can be computationally expensive, especially with a large number of weak learners. It may take longer to train compared to Random Forest.

Random Forest (Ensemble Algorithm):
Accuracy: Random Forest is known for its high accuracy, as it builds multiple decision trees and aggregates their predictions. It's robust and less prone to overfitting compared to a single decision tree.
Precision and Recall: Random Forest typically provides good precision and recall, making it suitable for both binary and multiclass classification tasks.
F1 Score: The F1 score of Random Forest is usually competitive and balanced, similar to AdaBoost.
Computational Complexity: Random Forest can be faster to train than AdaBoost since it parallelism tree construction. However, it may require more memory due to the ensemble of decision trees.