|  |
| --- |
| Experiment No. 6 |
| Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: 21/08/2023 |
| Date of Submission: 04/09/2023 |

**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—create 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 Di
4. Use training set Di to derive a model Mi
5. Computer error(Mi), the error rate of Mi
6. Error(Mi)=∑wj\*err(Xj)
7. If Error(Mi)>0.5 then
8. Go back to step 3 and try again
9. endif
10. for each tuple in Di that was correctly classified do
11. Multiply the weight of the tuple by error(Mi)/(1-error(Mi)
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. wi=log((1-error(Mi))/error(Mi))//weight of the classifiers vote
4. C=Mi(X) // get class prediction for X from Mi
5. Add wi 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 pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Load the dataset (replace 'data.csv' with the path to your dataset)

data = pd.read\_csv('data.csv')

# Prepare the data: X (features) and y (target)

X = data.drop('income', axis=1)

y = data['income']

# 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)

# Create and train the Gradient Boosting model

gradient\_boosting = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

gradient\_boosting.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = gradient\_boosting.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

confusion = confusion\_matrix(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

# Print the results

print("Gradient Boosting Model Performance:")

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", confusion)

print("Classification Report:\n", report)

**Output:**

Gradient Boosting Model Performance:

Accuracy: 0.85

Confusion Matrix:

[[7180 545]

[1110 1416]]

Classification Report:

precision recall f1-score support

<=50K 0.87 0.93 0.90 7725

>50K 0.72 0.56 0.63 2526

**Conclusion:**

In summary, both boosting and random forest algorithms demonstrated comparable performance levels on the Adult Census Income Dataset, with slight variations in precision and recall. The choice between these models should be based on specific requirements and trade-offs, as both are viable options for income prediction in this context. To identify the most appropriate model for a given application or to achieve optimal results, further evaluation and hyperparameter tuning may be necessary.