Experiment No. 5
Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance:
Date of Submission:

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 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 & OUTPUT:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostClassifier
# Load and preprocess the dataset
df = pd.read_csv('adult_dataset.csv')
df.dropna(inplace=True)

# Encode categorical features
label_encoders = {}
categorical_features = ['workclass', 'education', 'marital.status', 'occupation', 'relationship',
'race', 'sex', 'native.country']
for feature in categorical_features:
    le = LabelEncoder()
    df[feature] = le.fit_transform(df[feature])
    label_encoders[feature] = le
```

```
X = df.drop('income', axis=1)
y = df['income']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
df['income'] = LabelEncoder().fit_transform(df['income'])
# Initialize and train AdaBoost
ada_classifier = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),
n_estimators=50, random_state=42)
ada_classifier.fit(X_train, y_train)
y_pred_ada = ada_classifier.predict(X_test)
# Initialize and train Gradient Boosting
gb_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
max_depth=3, random_state=42)
gb_classifier.fit(X_train, y_train)
y_pred_gb = gb_classifier.predict(X_test)
# Initialize and train XGBoost
xgb_classifier = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42)
xgb_classifier.fit(X_train, y_train)
y_pred_xgb = xgb_classifier.predict(X_test)
# Initialize and train LightGBM
lgb_classifier = lgb.LGBMClassifier(n_estimators=100, learning_rate=0.1, max_depth=3,
random_state=42)
lgb_classifier.fit(X_train, y_train)
y_pred_lgb = lgb_classifier.predict(X_test)

catboost_classifier = CatBoostClassifier(n_estimators=100, learning_rate=0.1, depth=3,
random_state=42, verbose=0)
catboost_classifier.fit(X_train, y_train)
y_pred_catboost = catboost_classifier.predict(X_test)
# Compare performance
def print_comparison(name, y_true, y_pred):
```

```
    print(f"{name}")
    print(f'Accuracy: {accuracy_score(y_true, y_pred):.2f}')
    print(classification_report(y_true, y_pred))
    print("-" * 50)

print_comparison("AdaBoost", y_test, y_pred_ada)
print_comparison("Gradient Boosting", y_test, y_pred_gb)
print_comparison("XGBoost", y_test, y_pred_xgb)
print_comparison("LightGBM", y_test, y_pred_lgb)
print_comparison("CatBoost", y_test, y_pred_catboost)
```

```
AdaBoost
Accuracy: 0.86
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      4976
           1       0.77      0.58      0.66      1537

    accuracy                           0.86      6513
   macro avg       0.82      0.76      0.78      6513
weighted avg       0.85      0.86      0.85      6513

--------------------------------------------------
Gradient Boosting
Accuracy: 0.87
              precision    recall  f1-score   support

           0       0.88      0.95      0.92      4976
           1       0.80      0.58      0.67      1537

    accuracy                           0.87      6513
   macro avg       0.84      0.77      0.79      6513
```

weighted avg 0.86 0.87 0.86 6513

XGBoost

Accuracy: 0.86

	precision	recall	f1-score	support
0	0.88	0.95	0.91	4976
1	0.79	0.57	0.66	1537
accuracy			0.86	6513
macro avg	0.83	0.76	0.79	6513
weighted avg	0.86	0.86	0.85	6513

LightGBM

Accuracy: 0.86

	precision	recall	f1-score	support
0	0.88	0.95	0.91	4976
1	0.79	0.57	0.67	1537
accuracy			0.86	6513
macro avg	0.84	0.76	0.79	6513
weighted avg	0.86	0.86	0.86	6513

CatBoost

Accuracy: 0.86

	precision	recall	f1-score	support
0	0.87	0.95	0.91	4976

1	0.78	0.55	0.64	1537
accuracy			0.86	6513
macro avg	0.83	0.75	0.78	6513
weighted avg	0.85	0.86	0.85	6513

Conclusion:

After applying the Boosting algorithm, specifically AdaBoost, on the Adult Census Income Dataset, the accuracy, precision, recall, and F1 score were evaluated. AdaBoost improved the overall performance by combining several weak classifiers to create a stronger ensemble, with a weighted voting system. The confusion matrix provided detailed insights into the true positives, true negatives, false positives, and false negatives, enabling precise evaluations of each metric. The F1 score balanced both precision and recall, reflecting the trade-off between the two.

When comparing AdaBoost with the Random Forest algorithm, Random Forest generally provided higher accuracy due to its inherent ability to handle variance by averaging decisions from a diverse set of trees. However, AdaBoost can often outperform Random Forest when the dataset contains noisy data or imbalanced classes, as it focuses on misclassified instances during training. Both models have their strengths: AdaBoost tends to be better in reducing bias, while Random Forest excels in reducing variance.