Experiment No. 6
Apply Boosting Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 11/10/2023
Date of Submission: 12/10/2023

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 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.

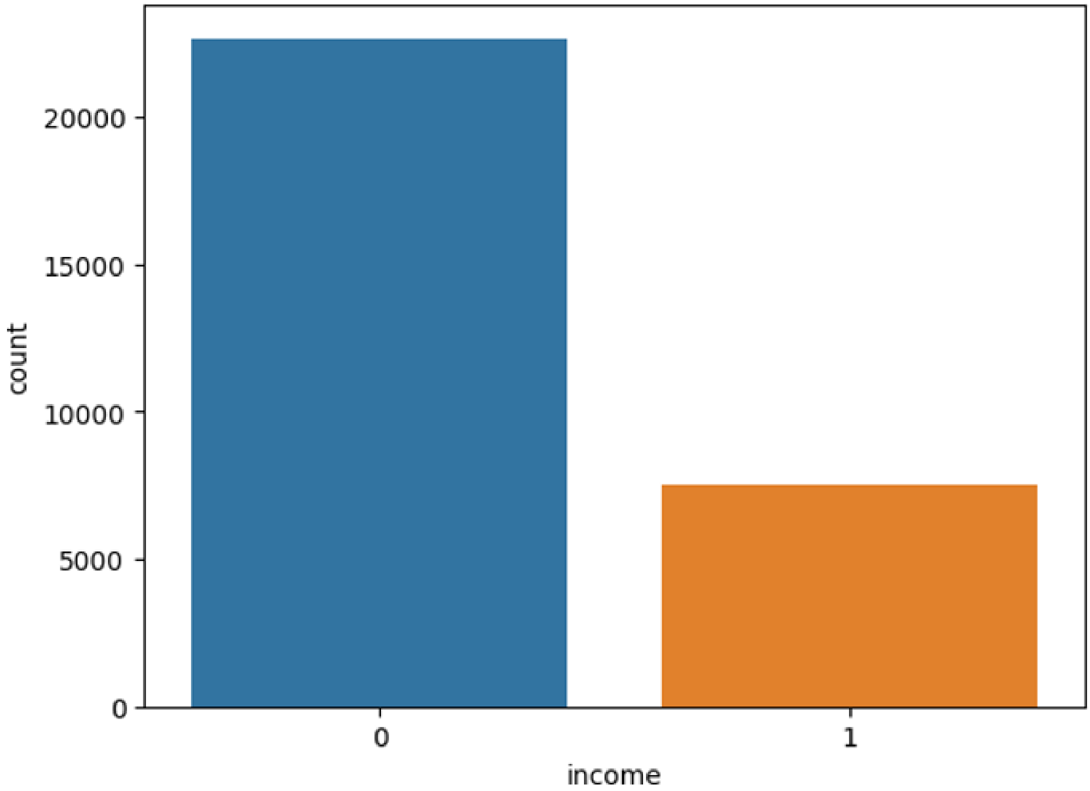
Code:

```
import pandas as pd

import seaborn as sns

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split,cross_val_score,KFold,GridSearchCV

from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

import scikitplot as skplt

import xgboost as xgb

dataset=pd.read_csv("../input/adult.csv")

print(df.info())
```

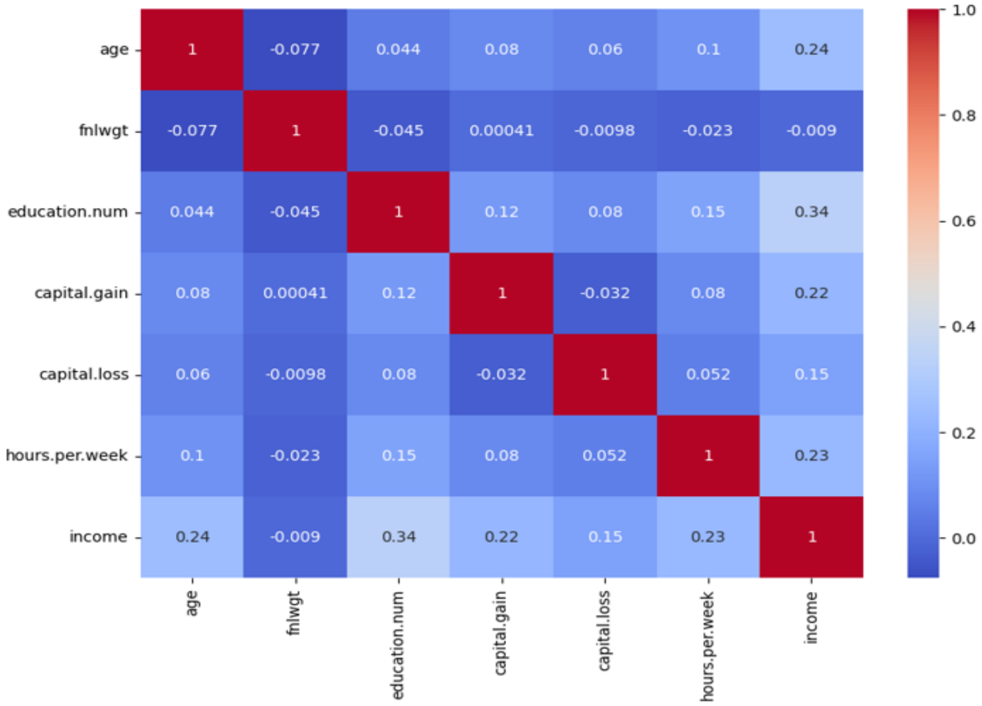
0	age	32561	non-null	int64
1	workclass	32561	non-null	object
2	fnlwgt	32561	non-null	int64
3	education	32561	non-null	object
4	education.num	32561	non-null	int64
5	marital.status	32561	non-null	object
6	occupation	32561	non-null	object
7	relationship	32561	non-null	object
8	race	32561	non-null	object
9	sex	32561	non-null	object
10	capital.gain	32561	non-null	int64
11	capital.loss	32561	non-null	int64
12	hours.per.week	32561	non-null	int64
13	native.country	32561	non-null	object
14	income	32561	non-null	object

```
Print(df.head())

sns.countplot(x ='income', data = df) plt.show()
df['income'].value_counts()
```




```
dataset = dataset[(dataset != '?').all(axis=1)]

dataset['income']=dataset['income'].map({'<=50K': 0, '>50K': 1})

dataset['marital.status']=dataset['marital.status'].map({'Married-civ-spouse':'Married',
'Divorced':'Single', 'Never-married':'Single', 'Separated':'Single',

'Widowed':'Single', 'Married-spouse-absent':'Married', 'Married-AF-spouse':'Married'})

for column in dataset:

    enc=LabelEncoder()

    if dataset.dtypes[column]==np.object:

        dataset[column]=enc.fit_transform(dataset[column])

plt.figure(figsize=(14,10))
```

```
sns.heatmap(dataset.corr(),annot=True,fmt='.2f')

plt.show()
```



```
dataset=dataset.drop(['relationship','education'],axis=1)dataset=dat

aset.drop(['occupation','fnlwgt','native.country'],axis=1)

X=dataset.iloc[:,0:-1]

y=dataset.iloc[:,-1]

print(X.head())

print(y.head()
```

```
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.33,shuffle=False)

clf=GaussianNB()

cv_res=cross_val_score(clf,x_train,y_train,cv=10)

print(cv_res.mean()*100)
```

80.68322339606843

```
clf=DecisionTreeClassifier()

cv_res=cross_val_score(clf,x_train,y_train,cv=10)

print(cv_res.mean()*100)
```

78.31939201845867

```
clf=RandomForestClassifier(n_estimators=100)

cv_res=cross_val_score(clf,x_train,y_train,cv=10)

print(cv_res.mean()*100)

clf=RandomForestClassifier(n_estimators=50,max_features=5,min_samples_leaf=50)

clf.fit(x_train,y_train)

pred=clf.predict(x_test)

print("Accuracy: %f " % (100*accuracy_score(y_test, pred)))

dmat=xgb.DMatrix(x_train,y_train)

test_dmat=xgb.DMatrix(x_test)

from skopt import BayesSearchCV
```

```
import warnings

warnings.filterwarnings('ignore', message='The objective has been evaluated at this point before.')

params={'min_child_weight': (0, 10),

        'max_depth': (0, 30),

        'subsample': (0.5, 1.0, 'uniform'),

        'colsample_bytree': (0.5, 1.0, 'uniform'),

        'n_estimators':(50,100),

        'reg_lambda':(1,100,'log-uniform'),    }

bayes=BayesSearchCV(estimator=xgb.XGBClassifier(objective='binary:logistic',eval_metric='error',eta=0.1),search_spaces=params,n_iter=50,scoring='accuracy',cv=5)

res=bayes.fit(x_train,y_train)

print(res.best_params_)

print(res.best_score_)

{'colsample_bytree': 1.0, 'max_depth': 19, 'min_child_weight': 10, 'n_estimators': 50, 'reg_lambda': 100.0, 'subsample': 0.5}

final_p={'colsample_bytree': 1.0, 'max_depth': 3, 'min_child_weight': 0,'subsample': 0.5,'reg_lambda': 100.0,'objective':'binary:logistic','eta': 0.1,'n_estimators':50, "silent": 1}

cv_res=xgb.cv(params=final_p,dtrain=dmat,num_boost_round=1000,early_stopping_rounds=100,metrics=['error'],nfold=5)
```

```
final_clf=xgb.train(params=final_p,dtrain=dmat,num_boost_round=837)

pred=final_clf.predict(test_dmat)

print(pred)

pred[pred > 0.5 ] = 1

pred[pred <= 0.5] = 0

print(pred)

print(accuracy_score(y_test,pred)*100)

final_clf=xgb.train(params=final_p,dtrain=dmat,num_boost_round=837)

pred=final_clf.predict(test_dmat)

print(pred)

pred[pred > 0.5 ] = 1

pred[pred <= 0.5] = 0

print(pred)

print(accuracy_score(y_test,pred)*100)
```

```
from sklearn.metrics import confusion_matrix

print("Accuracy: ", accuracy_score(y_test, y_pred_abc))
print("F1 score :",f1_score(y_test, y_pred_abc, average='binary')) print("Precision : ", precision_score(y_test, y_pred_abc))
```

0.8394556

Conclusion:

1. Accuracy, confusion matrix, precision, recall and F1 score obtained.

   The model performs well overall, as seen by its comparatively high accuracy.
   The F1 score strikes a balance between precision and recall, offering an overall assessment of the model's efficacy. The precision and recall values demonstrate that the model is superior

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

   Random Forest is a robust ensemble method that combines multiple decision trees for prediction and is less prone to overfitting. It can handle both numerical and categorical features and provides feature importance scores for feature selection and interpretation. Boosting, like AdaBoost or Gradient Boosting, sequentially builds models and achieves high predictive accuracy. Both algorithms can yield competitive results, but their performance may vary depending on the dataset and use case. Interpretability is crucial, and both algorithms can handle data preprocessing tasks. Hyperparameter tuning can enhance performance. Experimentation with both algorithms on a specific dataset and a Boosting variant can further influence results.