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.

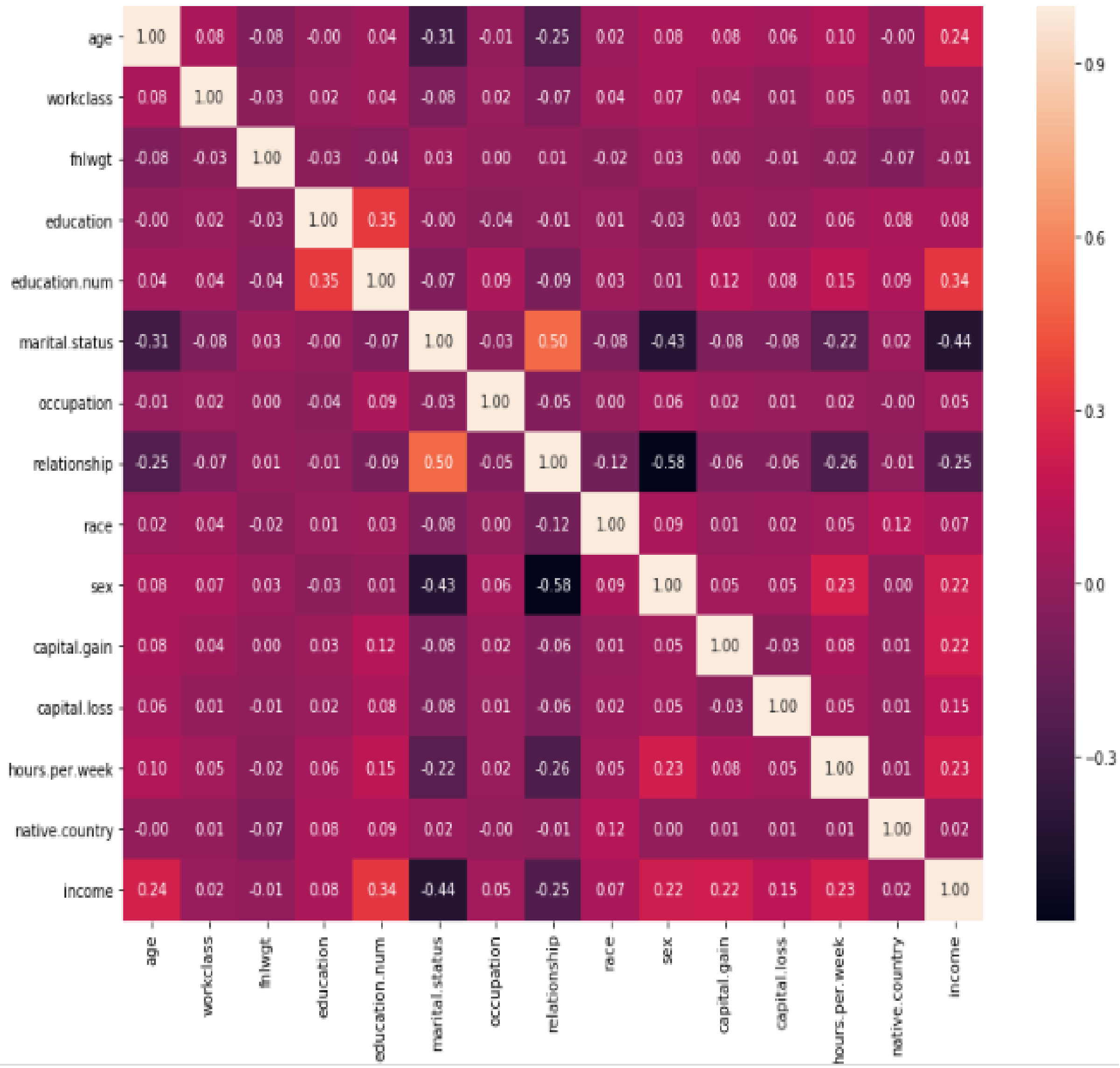
hours-per-week: continuous.

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")

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=dataset.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)
```

76.68322339606843

```
clf=DecisionTreeClassifier()

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

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

74.31939201845867

```
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)

cv_res.tail()
```

	train-error-mean	train-error-std	test-error-mean	test-error-std
833	0.129033	0.000570	0.137223	0.002801
834	0.128934	0.000633	0.137322	0.002658
835	0.128984	0.000493	0.137272	0.002618
836	0.128897	0.000578	0.137173	0.002636
837	0.129033	0.000625	0.136876	0.002336

```
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

import pandas as pd

confusion = confusion_matrix(y_test, pred)

df_confusion = pd.DataFrame (confusion, columns=['Predicted No', 'Predicted Yes'], index=['Actual No', 'Actual Yes'])

from sklearn.metrics import classification_report

print(classification_report (y_test, pred))

86.35037617073545

Conclusion:

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

   Accuracy Obtained is **86.35037617073545**

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

           0       0.89      0.94      0.91      4942
           1       0.76      0.63      0.69      1571

    accuracy                           0.86      6513
   macro avg       0.83      0.78      0.80      6513
weighted avg       0.86      0.86      0.86      6513
```

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

   The classification Report of AdaBoost and Random Forest is as follows:

```
AdaBoost:
Accuracy: 0.8635037617073545
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      4942
           1       0.76      0.63      0.69      1571

    accuracy                           0.86      6513
   macro avg       0.83      0.78      0.80      6513
weighted avg       0.86      0.86      0.86      6513


Random Forest:
Accuracy: 0.8602794411177644
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.93      0.91      4942
           1       0.75      0.63      0.69      1571

    accuracy                           0.86      6513
   macro avg       0.82      0.78      0.80      6513
weighted avg       0.86      0.86      0.86      6513
```

- Both techniques are very efficient for obtaining high accuracy. However, the algorithm we need to use depends on the factors such as the specific needs of the application, the available data, and the trade-offs between model complexity and interpretability. AdaBoost is known for its focus on misclassified samples, while Random Forest typically provides robust generalization.
- In this experiment, it can be concluded that **AdaBoost** has accuracy **86.35%,** while **Random Forest** has **86.02%** which is lesser than AdaBoost.