| |
|---|
| 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:**

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification_report, accuracy_score


df = pd.read_csv('adult.csv')


print(df.head())
```

```
   age  workclass  fnlwgt     education  educational-num      marital-status  \
0   25    Private  226802          11th                7       Never-married
1   38    Private   89814       HS-grad                9  Married-civ-spouse
2   28  Local-gov  336951    Assoc-acdm               12  Married-civ-spouse
3   44    Private  160323  Some-college               10  Married-civ-spouse
4   18          ?  103497  Some-college               10       Never-married

          occupation relationship   race  gender  capital-gain  capital-loss  \
0  Machine-op-inspct    Own-child  Black    Male             0             0
1    Farming-fishing      Husband  White    Male             0             0
2    Protective-serv      Husband  White    Male             0             0
3  Machine-op-inspct      Husband  Black    Male          7688             0
4                  ?    Own-child  White  Female             0             0

   hours-per-week native-country income
0              40  United-States  <=50K
1              50  United-States  <=50K
2              40  United-States   >50K
3              40  United-States   >50K
4              30  United-States  <=50K
```

```
df.columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
                          'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
              'hours-per-week', 'native-country', 'income']





df.replace(' ?', pd.NA, inplace=True)

df.dropna(inplace=True)


categorical_columns = ['workclass', 'education', 'marital-status', 'occupation','relationship', 'race', 'sex', 'native-country', 'income']

label_encoders = {}

for col in categorical_columns:

   le = LabelEncoder()

   df[col] = le.fit_transform(df[col])

   label_encoders[col] = le
```

```
print(df)
```

```
       age  workclass  fnlwgt  education  education-num  marital-status  \
0       25          4  226802          1              7               4
1       38          4   89814         11              9               2
2       28          2  336951          7             12               2
3       44          4  160323         15             10               2
4       18          0  103497         15             10               4
...    ...        ...     ...        ...            ...             ...
48837   27          4  257302          7             12               2
48838   40          4  154374         11              9               2
48839   58          4  151910         11              9               6
48840   22          4  201490         11              9               4
48841   52          5  287927         11              9               2

       occupation  relationship  race  sex  capital-gain  capital-loss  \
0               7             3     2    1             0             0
1               5             0     4    1             0             0
2              11             0     4    1             0             0
3               7             0     2    1          7688             0
4               0             3     4    0             0             0
...           ...           ...   ...  ...           ...           ...
48837          13             5     4    0             0             0
48838           7             0     4    1             0             0
48839           1             4     4    0             0             0
48840           1             3     4    1             0             0
48841           4             5     4    0         15024             0
```

```
X = df.drop('income', axis=1)

y = df['income']

def bootstrap_evaluate(X, y, n_bootstraps=100):

   accuracies = []

   for i in range(n_bootstraps):

       indices = np.random.randint(0, len(X), len(X))


       bootstrap_X, bootstrap_y = X.iloc[indices], y.iloc[indices]


        X_train, X_test, y_train, y_test = train_test_split(bootstrap_X,
bootstrap_y, test_size=0.2, random_state=42)
```

```
        rf = RandomForestClassifier(n_estimators=100, random_state=42)
        rf.fit(X_train, y_train)

        y_pred = rf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

    mean_accuracy = np.mean(accuracies)
    std_accuracy = np.std(accuracies)

    return mean_accuracy, std_accuracy

mean_acc, std_acc = bootstrap_evaluate(X, y, n_bootstraps=30)

X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(X, y, test_size=0.2, random_state=42)

rf_final = RandomForestClassifier(n_estimators=100, random_state=42)
rf_final.fit(X_train_final, y_train_final)

y_pred_final = rf_final.predict(X_test_final)

print(f"Final Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test_final, y_pred_final)}")
```

```
print(f"Classification    Report:\n{classification_report(y_test_final,
y_pred_final)}")

print(f"Mean Accuracy (Bootstrapping): {mean_acc}")

print(f"Standard Deviation of Accuracy (Bootstrapping): {std_acc}")
```

Conclusion:

Accuracy, confusion matrix, precision, recall, and F1 score obtained from applying the Random Forest algorithm on the Adult Census Income Dataset is

- Accuracy: 0.864
- Precision:
  - Class 0 (<=50K income): 0.89
  - Class 1 (>50K income): 0.74
- Recall:
  - Class 0: 0.93
  - Class 1: 0.64
- F1-score:
  - Class 0: 0.91
  - Class 1: 0.69

Both the Random Forest and Boosting algorithms were applied to predict income levels from the Adult Census Income Dataset. The Random Forest model achieved an accuracy of 0.864, demonstrating strong predictive capability with precision of 0.89 for <=50K incomes and 0.74 for >50K incomes. However, it showed lower recall for >50K incomes at 0.64, indicating some difficulty in identifying higher income brackets. In contrast, Boosting using bootstrapping yielded a mean accuracy of 0.935 with low standard deviation (0.0035), showcasing higher overall predictive accuracy and stability across different dataset samples. While Random Forest excelled in precision metrics, particularly for lower income classes, Boosting demonstrated superior average accuracy and robustness, making it potentially more reliable for broader predictive tasks in income classification scenarios.