| |
|---|
| Experiment No. 4 |
| Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: |
| Date of Submission: |

**Aim:** Apply Random Forest Algorithm on Adult Census Income Dataset and analyze the performance of the model.

**Objective:** Able to perform various feature engineering tasks, apply Random Forest Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score.

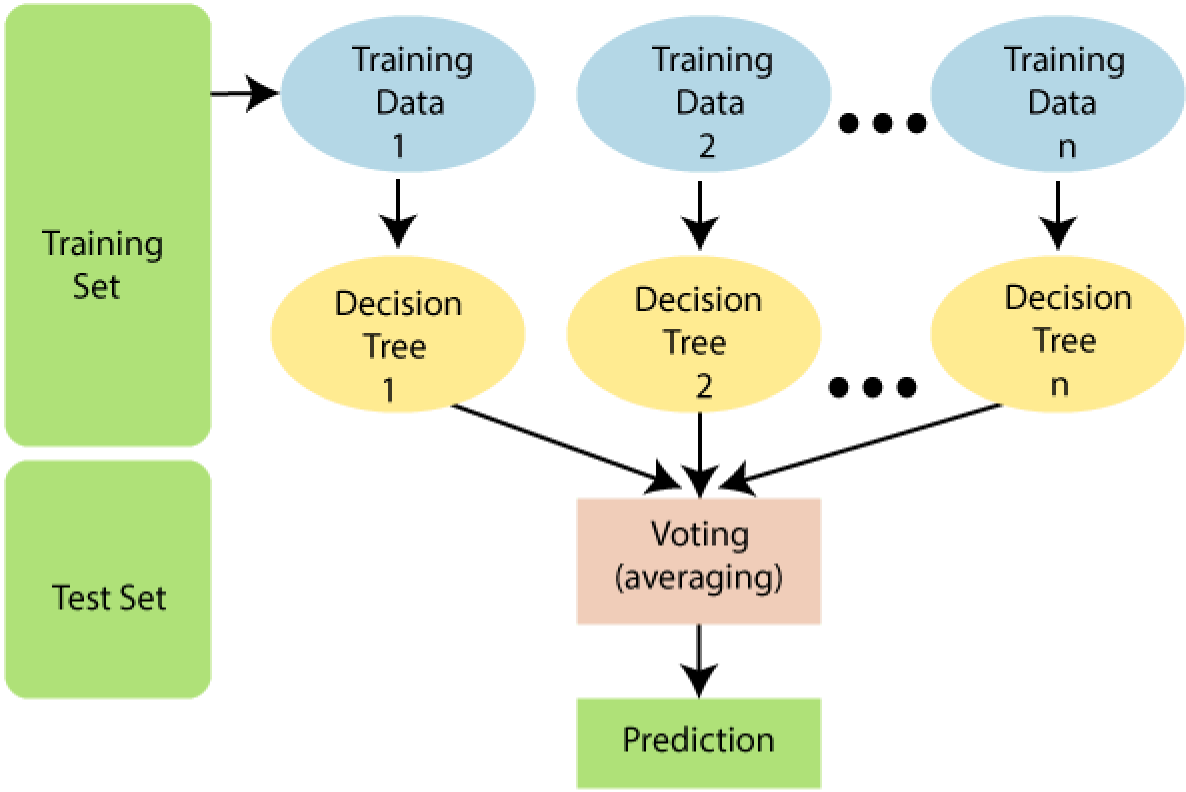
**Theory:**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:




**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.

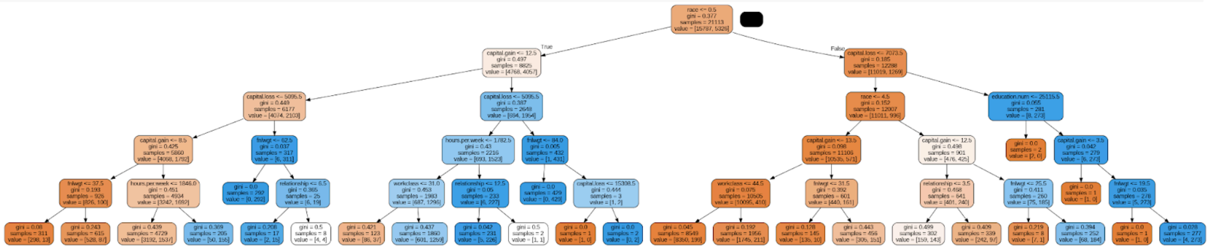
native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holland-Netherlands.

**Code & Output:**

```
!pip install pydotplus
```

```python
# Importing required packages for visualization

from IPython.display import Image

from six import StringIO

from sklearn.tree import export_graphviz

import pydotplus,graphviz

# Putting features

features = list(df.columns[1:])

features

['fnlwgt',

 'education.num',

 'capital.gain',

 'capital.loss',

 'hours.per.week',

 'workclass',

 'education',

 'marital.status',

 'occupation',

 'relationship',

 'race',

 'sex',

 'native.country',

 'income']


# plotting tree with max_depth=3

dot_data = StringIO()
```

```
export_graphviz(dt_default, out_file=dot_data,

                feature_names=features, filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```



```
from sklearn.model_selection import KFold

from sklearn.model_selection import GridSearchCV

# specify number of folds for k-fold CV

n_folds =

# parameters to build the model on

parameters = {'max_depth': range(1, 40)}

# instantiate the model

dtree = DecisionTreeClassifier(criterion = "gini",

                               random_state = 100)

# fit tree on training data

tree = GridSearchCV(dtree, parameters,

                    cv=n_folds,

                   scoring="accuracy")

tree.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
# scores of GridSearch CV

scores = tree.cv_results_

pd.DataFrame(scores).head()
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_max_depth | params | split0_test_score | split1_test_score | split2_test_ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.014912 | 0.002731 | 0.003536 | 0.000278 | 1 | {'max_depth': 1} | 0.747810 | 0.747810 | 0.7 |
| 1 | 0.020321 | 0.000216 | 0.003579 | 0.000050 | 2 | {'max_depth': 2} | 0.812219 | 0.818612 | 0.8 |
| 2 | 0.029969 | 0.003706 | 0.005022 | 0.001609 | 3 | {'max_depth': 3} | 0.828558 | 0.834241 | 0.8 |
| 3 | 0.034974 | 0.000750 | 0.003977 | 0.000556 | 4 | {'max_depth': 4} | 0.832583 | 0.840871 | 0.8 |
| 4 | 0.043431 | 0.001287 | 0.005024 | 0.000871 | 5 | {'max_depth': 5} | 0.834241 | 0.844897 | 0.8 |

| s | split0_test_score | split1_test_score | split2_test_score | split3_test_score | split4_test_score | mean_test_score | std_test_score | rank_test_score |
|---|---|---|---|---|---|---|---|---|
| ': } | 0.747810 | 0.747810 | 0.747573 | 0.747750 | 0.747750 | 0.747738 | 0.000087 | 39 |
| ': } | 0.812219 | 0.818612 | 0.820507 | 0.825675 | 0.822833 | 0.819969 | 0.004538 | 16 |
| ': } | 0.828558 | 0.834241 | 0.834478 | 0.836570 | 0.837518 | 0.834273 | 0.003115 | 12 |
| ': } | 0.832583 | 0.840871 | 0.842529 | 0.842729 | 0.842255 | 0.840193 | 0.003860 | 9 |
| ': } | 0.834241 | 0.844897 | 0.847265 | 0.842729 | 0.847466 | 0.843319 | 0.004858 | 7 |

```
from sklearn.model_selection import KFold

from sklearn.model_selection import GridSearchCV

# specify number of folds for k-fold CV

n_folds =

# parameters to build the model on

parameters = {'min_samples_leaf': range(5, 200, 20)}

# instantiate the model

dtree = DecisionTreeClassifier(criterion = "gini",
```

```
                                random_state = 100)

# fit tree on training data

tree = GridSearchCV(dtree, parameters,

                    cv=n_folds,

                    scoring="accuracy")

tree.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```
# scores of GridSearch CV

scores = tree.cv_results_

pd.DataFrame(scores).head()
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_min_samples_leaf | params | split0_test_score | split1_test_score |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.302561 | 0.047643 | 0.010234 | 0.003623 | 5 | {'min_samples_leaf': 5} | 0.825716 | 0.827848 |
| 1 | 0.226468 | 0.059930 | 0.013695 | 0.003549 | 25 | {'min_samples_leaf': 25} | 0.841819 | 0.851291 |
| 2 | 0.119051 | 0.014305 | 0.008670 | 0.002440 | 45 | {'min_samples_leaf': 45} | 0.843003 | 0.849159 |
| 3 | 0.109432 | 0.024206 | 0.007160 | 0.001666 | 65 | {'min_samples_leaf': 65} | 0.841108 | 0.852711 |
| 4 | 0.071941 | 0.003895 | 0.004989 | 0.000820 | 85 | {'min_samples_leaf': 85} | 0.838030 | 0.849159 |

| split0_test_score | split1_test_score | split2_test_score | split3_test_score | split4_test_score | mean_test_score | std_test_score | rank_test_score |
|---|---|---|---|---|---|---|---|
| 0.825716 | 0.827848 | 0.819560 | 0.826149 | 0.818806 | 0.823616 | 0.003696 | 10 |
| 0.841819 | 0.851291 | 0.839451 | 0.842018 | 0.849360 | 0.844788 | 0.004651 | 6 |
| 0.843003 | 0.849159 | 0.846555 | 0.851018 | 0.851729 | 0.848293 | 0.003194 | 1 |
| 0.841108 | 0.852711 | 0.845371 | 0.851492 | 0.838465 | 0.845830 | 0.005589 | 2 |
| 0.838030 | 0.849159 | 0.845371 | 0.851492 | 0.842018 | 0.845214 | 0.004834 | 3 |

```
from sklearn.model_selection import KFold

from sklearn.model_selection import GridSearchCV

# specify number of folds for k-fold CV

n_folds = 5

# parameters to build the model on

parameters = {'min_samples_split': range(5, 200, 20)}

# instantiate the model

dtree = DecisionTreeClassifier(criterion = "gini",

                               random_state = 100)

# fit tree on training data

tree = GridSearchCV(dtree, parameters,

                    cv=n_folds,

                    scoring="accuracy")

tree.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

```
# scores of GridSearch CV

scores = tree.cv_results_

pd.DataFrame(scores).head()
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_min_samples_split | params | split0_test_score | split1_test_score |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.257015 | 0.030095 | 0.011569 | 0.004530 | 5 | {'min_samples_split': 5} | 0.811982 | 0.811035 |
| 1 | 0.251020 | 0.036221 | 0.009931 | 0.002677 | 25 | {'min_samples_split': 25} | 0.825006 | 0.825243 |
| 2 | 0.283793 | 0.051051 | 0.013075 | 0.003601 | 45 | {'min_samples_split': 45} | 0.835188 | 0.839687 |
| 3 | 0.131714 | 0.039597 | 0.005965 | 0.001252 | 65 | {'min_samples_split': 65} | 0.839451 | 0.845844 |
| 4 | 0.105095 | 0.010136 | 0.005290 | 0.000514 | 85 | {'min_samples_split': 85} | 0.846081 | 0.853895 |

| split0_test_score | split1_test_score | split2_test_score | split3_test_score | split4_test_score | mean_test_score | std_test_score | rank_test_score |
|---|---|---|---|---|---|---|---|
| 0.811982 | 0.811035 | 0.818376 | 0.811701 | 0.808385 | 0.812296 | 0.003296 | 10 |
| 0.825006 | 0.825243 | 0.830215 | 0.822596 | 0.827570 | 0.826126 | 0.002581 | 9 |
| 0.835188 | 0.839687 | 0.830215 | 0.827333 | 0.838702 | 0.834225 | 0.004783 | 8 |
| 0.839451 | 0.845844 | 0.837556 | 0.833728 | 0.843913 | 0.840098 | 0.004360 | 7 |
| 0.846081 | 0.853895 | 0.838977 | 0.837281 | 0.845334 | 0.844314 | 0.005898 | 6 |

```
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'criterion': ["entropy", "gini"]
}
n_folds = 5
# Instantiate the grid search model
dtree = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator = dtree, param_grid = param_grid,
                           cv = n_folds, verbose = 1)
# Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
▸ DecisionTreeClassifier
```

```python
# cv results

cv_results = pd.DataFrame(grid_search.cv_results_)

cv_results
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_criterion | param_max_depth | param_min_samples_leaf | param_min_samples_s |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.066055 | 0.002414 | 0.006103 | 0.000330 | entropy | 5 | 50 | |
| 1 | 0.066297 | 0.002458 | 0.007829 | 0.003156 | entropy | 5 | 50 | |
| 2 | 0.063863 | 0.002771 | 0.006069 | 0.000250 | entropy | 5 | 100 | |
| 3 | 0.069619 | 0.007983 | 0.006503 | 0.000399 | entropy | 5 | 100 | |
| 4 | 0.110682 | 0.007796 | 0.006779 | 0.000537 | entropy | 10 | 50 | |
| 5 | 0.088213 | 0.019038 | 0.004649 | 0.000906 | entropy | 10 | 50 | |
| 6 | 0.072638 | 0.003297 | 0.005654 | 0.002214 | entropy | 10 | 100 | |
| 7 | 0.068670 | 0.001294 | 0.004287 | 0.000213 | entropy | 10 | 100 | |
| 8 | 0.041203 | 0.000853 | 0.004231 | 0.000213 | gini | 5 | 50 | |
| 9 | 0.043932 | 0.002999 | 0.004368 | 0.000620 | gini | 5 | 50 | |
| 10 | 0.041142 | 0.000680 | 0.004011 | 0.000175 | gini | 5 | 100 | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 0.040627 | 0.001732 | 0.003843 | 0.000085 | gini | 5 | 100 |
| 12 | 0.066317 | 0.001318 | 0.004181 | 0.000077 | gini | 10 | 50 |
| 13 | 0.069441 | 0.002989 | 0.004656 | 0.000756 | gini | 10 | 50 |
| 14 | 0.065258 | 0.003734 | 0.004463 | 0.000590 | gini | 10 | 100 |
| 15 | 0.065308 | 0.003595 | 0.004518 | 0.000568 | gini | 10 | 100 |

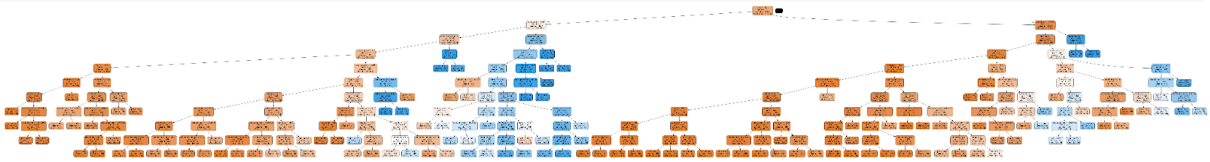
```
# printing the optimal accuracy score and hyperparameters
print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.8510400232064759
DecisionTreeClassifier(max_depth=10,min_samples_leaf=50,min_samples_spl
it=50)
```

```
# model with optimal hyperparameters
clf_gini = DecisionTreeClassifier(criterion = "gini",
                                  random_state = 100,
                                  max_depth=10,
                                  min_samples_leaf=50,
                                  min_samples_split=50)
clf_gini.fit(X_train, y_train)
```

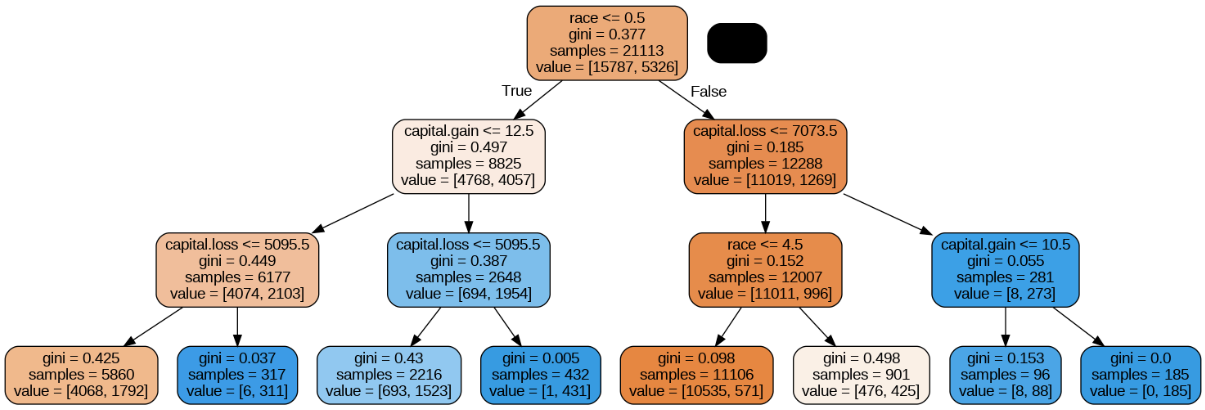
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, min_samples_leaf=50, min_samples_split=50,
                       random_state=100)
```

```
# accuracy score

clf_gini.score(X_test,y_test)

0.850922753895458


# plotting the tree

dot_data = StringIO()

export_graphviz(clf_gini,
out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```



```
# tree with max_depth = 3

clf_gini = DecisionTreeClassifier(criterion = "gini",

                                  random_state = 100,

                                  max_depth=3,

                                  min_samples_leaf=50,

                                  min_samples_split=50)

clf_gini.fit(X_train, y_train)

# score

print(clf_gini.score(X_test,y_test))

0.8393192617968837



# plotting tree with max_depth=3

dot_data = StringIO()
```

```
export_graphviz(clf_gini,
out_file=dot_data,feature_names=features,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```




```
# classification metrics

from sklearn.metrics import classification_report,confusion_matrix

y_pred = clf_gini.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      6867
           1       0.77      0.47      0.59      2182

    accuracy                           0.84      9049
   macro avg       0.81      0.71      0.74      9049
weighted avg       0.83      0.84      0.82      9049
```

**Conclusion:**

Accuracy is the proportion of correctly predicted instances out of the total instances. Here, it is 0.84, meaning the model correctly predicted 84% of the cases. Performance on class 1 (minority class) is not as strong, with lower recall and F1-score.

Confusion matrix as follows:

- **True Positives (TP)** for class 1 = Recall × Support = 0.47 × 2182 ≈ **1026**
- **False Negatives (FN)** for class 1 = Support - TP = 2182 - 1026 ≈ **1156**
- **True Negatives (TN)** for class 0 = Recall × Support = 0.96 × 6867 ≈ **6593**
- **False Positives (FP)** for class 0 = Support - TN = 6867 - 6593 ≈ **274**