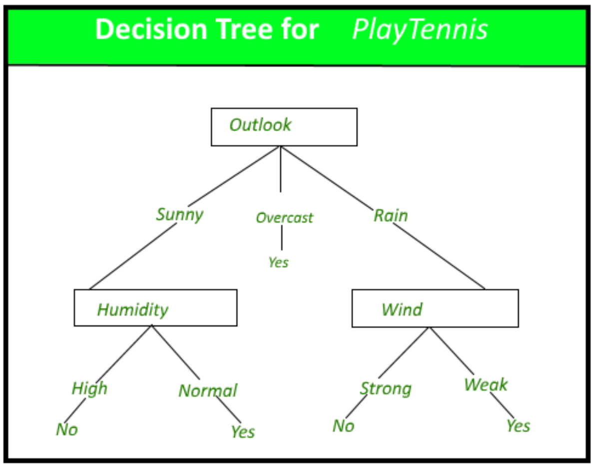
| |
|---|
| Experiment No. 3 |
| Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model |
| Date of Performance: 16/08/2023 |
| Date of Submission: 13/09/2023 |

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

**Objective:** To perform various feature engineering tasks, apply Decision Tree Algorithm on the given dataset and maximize the accuracy, Precision, Recall, F1 score. Improve the performance by performing different data engineering and feature engineering tasks.

**Theory:**

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.




**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 matplotlib.pyplot as plt
import seaborn as sb
import numpy as np
```

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

```
df.describe()
```

| | age | fnlwgt | education.num | capital.gain | capital.loss | hours.per.week |
|---|---|---|---|---|---|---|
| **count** | 32561.000000 | 3.256100e+04 | 32561.000000 | 32561.000000 | 32561.000000 | 32561.000000 |
| **mean** | 38.581647 | 1.897784e+05 | 10.080679 | 1077.648844 | 87.303830 | 40.437456 |
| **std** | 13.640433 | 1.055500e+05 | 2.572720 | 7385.292085 | 402.960219 | 12.347429 |
| **min** | 17.000000 | 1.228500e+04 | 1.000000 | 0.000000 | 0.000000 | 1.000000 |
| **25%** | 28.000000 | 1.178270e+05 | 9.000000 | 0.000000 | 0.000000 | 40.000000 |
| **50%** | 37.000000 | 1.783560e+05 | 10.000000 | 0.000000 | 0.000000 | 40.000000 |
| **75%** | 48.000000 | 2.370510e+05 | 12.000000 | 0.000000 | 0.000000 | 45.000000 |
| **max** | 90.000000 | 1.484705e+06 | 16.000000 | 99999.000000 | 4356.000000 | 99.000000 |

```
df.head(5)
```

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relationship | race | sex | capital.gain | ca |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 90 | ? | 77053 | HS-grad | 9 | Widowed | ? | Not-in-family | White | Female | 0 | |
| **1** | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial | Not-in-family | White | Female | 0 | |
| **2** | 66 | ? | 186061 | Some-college | 10 | Widowed | ? | Unmarried | Black | Female | 0 | |
| **3** | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unmarried | White | Female | 0 | |
| **4** | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty | Own-child | White | Female | 0 | |

```
print ("Rows : " ,df.shape[0])
print ("Columns : " ,df.shape[1])
print ("\nFeatures : \n" ,df.columns.tolist())
print ("\nMissing values : ", df.isnull().sum().values.sum())
print ("\nUnique values : \n",df.nunique())
```

```
Rows :  32561
Columns :  15

Features :
 ['age', 'workclass', 'fnlwgt', 'education', 'education.num', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'capit

Missing values :  0

Unique values :
 age                  73
workclass             9
fnlwgt            21648
education            16
education.num        16
marital.status        7
occupation           15
relationship          6
race                  5
sex                   2
capital.gain        119
capital.loss         92
hours.per.week       94
native.country       42
income                2
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
```

```
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 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
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
# checking "?" total values present in particular 'workclass' feature
df_check_missing_workclass = (df['workclass']=='?').sum()
df_check_missing_workclass
```

```
1836
```

```
# checking "?" total values present in particular 'occupation' feature
df_check_missing_occupation = (df['occupation']=='?').sum()
df_check_missing_occupation
```

```
1843
```

```
# checking "?" values, how many are there in the whole dataset
df_missing = (df=='?').sum()
df_missing
```

```
age                  0
workclass         1836
fnlwgt               0
education            0
education.num        0
marital.status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital.gain         0
capital.loss         0
hours.per.week       0
native.country     583
income               0
dtype: int64
```

```
percent_missing = (df=='?').sum() * 100/len(df)
percent_missing
```

```
age               0.000000
workclass         5.638647
fnlwgt            0.000000
education         0.000000
education.num     0.000000
marital.status    0.000000
occupation        5.660146
relationship      0.000000
race              0.000000
sex               0.000000
capital.gain      0.000000
capital.loss      0.000000
hours.per.week    0.000000
native.country    1.790486
income            0.000000
dtype: float64
```

```
# find total number of rows which doesn't contain any missing value as '?'
df.apply(lambda x: x !='?',axis=1).sum()
```

```
age               32561
workclass         30725
fnlwgt            32561
education         32561
education.num     32561
```

```
marital.status    32561
occupation        30718
relationship      32561
race              32561
sex               32561
capital.gain      32561
capital.loss      32561
hours.per.week    32561
native.country    31978
income            32561
dtype: int64
```

```
# dropping the rows having missing values in workclass
df = df[df['workclass'] !='?']
df.head()
```

| | age | workclass | fnlwgt | education | education.num | marital.status | occupation | relationship | race | sex | capital.gain | ca |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 82 | Private | 132870 | HS-grad | 9 | Widowed | Exec-managerial | Not-in-family | White | Female | 0 | |
| **3** | 54 | Private | 140359 | 7th-8th | 4 | Divorced | Machine-op-inspct | Unmarried | White | Female | 0 | |
| **4** | 41 | Private | 264663 | Some-college | 10 | Separated | Prof-specialty | Own-child | White | Female | 0 | |
| **5** | 34 | Private | 216864 | HS-grad | 9 | Divorced | Other-service | Unmarried | White | Female | 0 | |
| **6** | 38 | Private | 150601 | 10th | 6 | Separated | Adm-clerical | Unmarried | White | Male | 0 | |

```
# select all categorical variables
df_categorical = df.select_dtypes(include=['object'])
# checking whether any other column contains '?' value
df_categorical.apply(lambda x: x=='?',axis=1).sum()
```

```
workclass           0
education           0
marital.status      0
occupation          7
relationship        0
race                0
sex                 0
native.country    556
income              0
dtype: int64
```

```
# dropping the "?"s from occupation and native.country
df = df[df['occupation'] !='?']
df = df[df['native.country'] !='?']
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             30162 non-null  int64 
 1   workclass       30162 non-null  object
 2   fnlwgt          30162 non-null  int64 
 3   education       30162 non-null  object
 4   education.num   30162 non-null  int64 
 5   marital.status  30162 non-null  object
 6   occupation      30162 non-null  object
 7   relationship    30162 non-null  object
 8   race            30162 non-null  object
 9   sex             30162 non-null  object
 10  capital.gain    30162 non-null  int64 
 11  capital.loss    30162 non-null  int64 
 12  hours.per.week  30162 non-null  int64 
 13  native.country  30162 non-null  object
 14  income          30162 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
from sklearn import preprocessing
# encode categorical variables using label Encoder
# select all categorical variables
df_categorical = df.select_dtypes(include=['object'])
df_categorical.head()
```

| | workclass | education | marital.status | occupation | relationship | race | sex | native.country | income |
|---|---|---|---|---|---|---|---|---|---|
| **1** | Private | HS-grad | Widowed | Exec-managerial | Not-in-family | White | Female | United-States | <=50K |
| **3** | Private | 7th-8th | Divorced | Machine-op-inspct | Unmarried | White | Female | United-States | <=50K |
| **4** | Private | Some-college | Separated | Prof-specialty | Own-child | White | Female | United-States | <=50K |
| **5** | Private | HS-grad | Divorced | Other-service | Unmarried | White | Female | United-States | <=50K |
| **6** | Private | 10th | Separated | Adm-clerical | Unmarried | White | Male | United-States | <=50K |

```
# apply label encoder to df_categorical
le = preprocessing.LabelEncoder()
df_categorical = df_categorical.apply(le.fit_transform)
df_categorical.head()
```

| | workclass | education | marital.status | occupation | relationship | race | sex | native.country | income |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 2 | 11 | 6 | 3 | 1 | 4 | 0 | 38 | 0 |
| **3** | 2 | 5 | 0 | 6 | 4 | 4 | 0 | 38 | 0 |
| **4** | 2 | 15 | 5 | 9 | 3 | 4 | 0 | 38 | 0 |
| **5** | 2 | 11 | 0 | 7 | 4 | 4 | 0 | 38 | 0 |
| **6** | 2 | 0 | 5 | 0 | 4 | 4 | 1 | 38 | 0 |

```
# Next, Concatenate df_categorical dataframe with original df (dataframe)
# first, Drop earlier duplicate columns which had categorical values
df = df.drop(df_categorical.columns,axis=1)
df = pd.concat([df,df_categorical],axis=1)
df.head()
```

| | age | fnlwgt | education.num | capital.gain | capital.loss | hours.per.week | workclass | education | marital.status | occupation | relatio |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 82 | 132870 | 9 | 0 | 4356 | 18 | 2 | 11 | 6 | 3 | |
| **3** | 54 | 140359 | 4 | 0 | 3900 | 40 | 2 | 5 | 0 | 6 | |
| **4** | 41 | 264663 | 10 | 0 | 3900 | 40 | 2 | 15 | 5 | 9 | |
| **5** | 34 | 216864 | 9 | 0 | 3770 | 45 | 2 | 11 | 0 | 7 | |
| **6** | 38 | 150601 | 6 | 0 | 3770 | 40 | 2 | 0 | 5 | 0 | |

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             30162 non-null  int64
 1   fnlwgt          30162 non-null  int64
 2   education.num   30162 non-null  int64
 3   capital.gain    30162 non-null  int64
 4   capital.loss    30162 non-null  int64
 5   hours.per.week  30162 non-null  int64
 6   workclass       30162 non-null  int64
 7   education       30162 non-null  int64
 8   marital.status  30162 non-null  int64
 9   occupation      30162 non-null  int64
 10  relationship    30162 non-null  int64
 11  race            30162 non-null  int64
 12  sex             30162 non-null  int64
 13  native.country  30162 non-null  int64
 14  income          30162 non-null  int64
dtypes: int64(15)
memory usage: 3.7 MB
```

```
# convert target variable income to categorical
df['income'] = df['income'].astype('category')
# check df info again whether everything is in right format or not
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30162 entries, 1 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             30162 non-null  int64
```

```
 1   fnlwgt          30162 non-null  int64
 2   education.num   30162 non-null  int64
 3   capital.gain    30162 non-null  int64
 4   capital.loss    30162 non-null  int64
 5   hours.per.week  30162 non-null  int64
 6   workclass       30162 non-null  int64
 7   education       30162 non-null  int64
 8   marital.status  30162 non-null  int64
 9   occupation      30162 non-null  int64
 10  relationship    30162 non-null  int64
 11  race            30162 non-null  int64
 12  sex             30162 non-null  int64
 13  native.country  30162 non-null  int64
 14  income          30162 non-null  category
dtypes: category(1), int64(14)
memory usage: 3.5 MB
```

```
# Importing train_test_split
from sklearn.model_selection import train_test_split
# Putting independent variables/features to X
X = df.drop('income',axis=1)
# Putting response/dependent variable/feature to y
y = df['income']
```

```
X.head(5)
```

| | age | fnlwgt | education.num | capital.gain | capital.loss | hours.per.week | workclass | education | marital.status | occupation | relatio |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 82 | 132870 | 9 | 0 | 4356 | 18 | 2 | 11 | 6 | 3 | |
| **3** | 54 | 140359 | 4 | 0 | 3900 | 40 | 2 | 5 | 0 | 6 | |
| **4** | 41 | 264663 | 10 | 0 | 3900 | 40 | 2 | 15 | 5 | 9 | |
| **5** | 34 | 216864 | 9 | 0 | 3770 | 45 | 2 | 11 | 0 | 7 | |
| **6** | 38 | 150601 | 6 | 0 | 3770 | 40 | 2 | 0 | 5 | 0 | |

```
y.head(3)
```

```
1    0
3    0
4    0
Name: income, dtype: category
Categories (2, int64): [0, 1]
```

```
# Splitting the data into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,random_state=99)
X_train.head()
```

| | age | fnlwgt | education.num | capital.gain | capital.loss | hours.per.week | workclass | education | marital.status | occupation | rel |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **24351** | 42 | 289636 | 9 | 0 | 0 | 46 | 2 | 11 | 2 | 13 | |
| **15626** | 37 | 52465 | 9 | 0 | 0 | 40 | 1 | 11 | 4 | 7 | |
| **4347** | 38 | 125933 | 14 | 0 | 0 | 40 | 0 | 12 | 2 | 9 | |
| **23972** | 44 | 183829 | 13 | 0 | 0 | 38 | 5 | 9 | 4 | 0 | |
| **26843** | 35 | 198841 | 11 | 0 | 0 | 35 | 2 | 8 | 0 | 12 | |

```
# Importing decision tree classifier from sklearn library
from sklearn.tree import DecisionTreeClassifier
# Fitting the decision tree with default hyperparameters, apart from
# max_depth which is 5 so that we can plot and read the tree.
dt_default = DecisionTreeClassifier(max_depth=5)
dt_default.fit(X_train,y_train)
```

```
▾     DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

```
# check the evaluation metrics of our default model
# Importing classification report and confusion matrix from sklearn metrics
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
# making predictions
y_pred_default = dt_default.predict(X_test)
# Printing classifier report after prediction
print(classification_report(y_test,y_pred_default))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      6867
           1       0.78      0.52      0.63      2182

    accuracy                           0.85      9049
   macro avg       0.82      0.74      0.77      9049
weighted avg       0.84      0.85      0.84      9049
```

```
# Printing confusion matrix and accuracy
print(confusion_matrix(y_test,y_pred_default))
print(accuracy_score(y_test,y_pred_default))
```

```
[[6553  314]
 [1039 1143]]
0.8504807161012267
```

```
!pip install my-package
```

```
Collecting my-package
  Downloading my_package-0.0.0-py3-none-any.whl (2.0 kB)
Installing collected packages: my-package
Successfully installed my-package-0.0.0
```

```
!pip install pydotplus
```

```
Requirement already satisfied: pydotplus in /usr/local/lib/python3.10/dist-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.10/dist-packages (from pydotplus) (3.1.1)
```

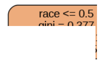
```
# Importing required packages for visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus,graphviz
# Putting features
features = list(df.columns[1:])
features
```

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

```
!pip install graphviz
```

```
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (0.20.1)
```

```
# 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())
```




```
# GridSearchCV to find optimal max_depth
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 = {'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
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'max_depth': range(1, 40)}, scoring='accuracy')
    estimator: DecisionTreeClassifier
    DecisionTreeClassifier(random_state=100)
        DecisionTreeClassifier
        DecisionTreeClassifier(random_state=100)
```

```
# 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 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.018425 | 0.001211 | 0.005323 | 0.000191 | 1 | {'max_depth': 1} | 0.747810 | 0.747810 |
| **1** | 0.035381 | 0.016654 | 0.007002 | 0.003257 | 2 | {'max_depth': 2} | 0.812219 | 0.818612 |
| **2** | 0.048566 | 0.017279 | 0.008126 | 0.003305 | 3 | {'max_depth': 3} | 0.828558 | 0.834241 |
| **3** | 0.040141 | 0.018838 | 0.003873 | 0.000468 | 4 | {'max_depth': 4} | 0.832583 | 0.840871 |
| **4** | 0.073534 | 0.036241 | 0.009387 | 0.008625 | 5 | {'max_depth': 5} | 0.834241 | 0.844897 |

```
# GridSearchCV to find optimal max_depth
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_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)
# 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 |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.129880 | 0.012816 | 0.008649 | 0.003510 | 5 | {'min_samples_leaf': 5} | 0.825716 | |

```
# GridSearchCV to find optimal min_samples_split
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
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             param_grid={'min_samples_split': range(5, 200, 20)},
             scoring='accuracy')
estimator: DecisionTreeClassifier
DecisionTreeClassifier(random_state=100)
DecisionTreeClassifier
DecisionTreeClassifier(random_state=100)
```

```
# 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 | spli |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.165695 | 0.022669 | 0.010115 | 0.004717 | 5 | {'min_samples_split': 5} | 0.811982 | |
| **1** | 0.194452 | 0.047568 | 0.008605 | 0.005247 | 25 | {'min_samples_split': 25} | 0.825006 | |
| **2** | 0.125775 | 0.006459 | 0.006552 | 0.000275 | 45 | {'min_samples_split': 45} | 0.835188 | |
| **3** | 0.128474 | 0.019117 | 0.006556 | 0.000719 | 65 | {'min_samples_split': 65} | 0.839451 | |
| **4** | 0.120904 | 0.010535 | 0.006089 | 0.000474 | 85 | {'min_samples_split': 85} | 0.846081 | |

```
# 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**

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['entropy', 'gini'],
```

```
# cv results
cv_results = pd.DataFrame(grid_search.cv_results_)
```

```
             verbose=1)
```

```
cv_results
```

| | mean_fit_time | std_fit_time | mean_score_time | std_score_time | param_criterion | param_max_depth | param_min_samples_leaf | param_mi |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.109761 | 0.027735 | 0.007429 | 0.003987 | entropy | 5 | 50 | |
| **1** | 0.077615 | 0.021292 | 0.007246 | 0.001316 | entropy | 5 | 50 | |
| **2** | 0.059589 | 0.004072 | 0.005806 | 0.000180 | entropy | 5 | 100 | |
| **3** | 0.064347 | 0.011536 | 0.005823 | 0.000098 | entropy | 5 | 100 | |
| **4** | 0.091334 | 0.007191 | 0.005694 | 0.000245 | entropy | 10 | 50 | |
| **5** | 0.070263 | 0.013116 | 0.003854 | 0.000769 | entropy | 10 | 50 | |
| **6** | 0.058181 | 0.003006 | 0.004055 | 0.000751 | entropy | 10 | 100 | |

**Conclusion:**

1. Discuss about how categorical attributes have been dealt with during data pre-processing.

➔

In data preprocessing, categorical attributes are typically converted into numerical representations to make them suitable for machine learning algorithms which is achieved through methods such as label encoding. Label encoding assigns a unique integer to each category. Additionally, we also choose to drop a column containing categorical values since it doesn't provide meaningful information for the analysis.

2. Discuss the hyper-parameter tunning done based on the decision tree obtained.

➔

i) Max Depth: By limiting the decision tree's depth, this parameter keeps it from overcomplicating and overfitting the training set of data.

ii) Min Samples Split: This setting establishes the minimal amount of samples needed in a node to be subject to additional splitting. It aids in reducing the tree's tendency to specified choices depending on a constrained set of circumstances.

ii) Min Samples Leaf: This setting determines how many samples must be present in a leaf node. This can stop the tree from forming nodes with the same properties as min samples split extremely few occasions.

iv) Criteria: This option specifies the method for calculating a split's quality. Common criteria include "entropy" and "impurity".

3. Comment on the accuracy, confusion matrix, precision, recall and F1 score obtained.

➔

- Accuracy is a measure of overall correctness and is relatively high at approximately 85.05%, indicating that a significant portion of predictions is accurate.
- The confusion matrix provides more detailed information about the model's performance. It shows that there are 6553 true negatives, 1143 true positives, 1039 false positives, and 314 false negatives. This information helps in understanding how the model performs with respect to each class.
- Precision is the ratio of true positives to the total predicted positives (true positives + false positives). The precision for class 1 (positive class) is approximately 0.78, indicating that when the model predicts a positive outcome, it is correct about 78% of the time.
- Recall (or sensitivity) is the ratio of true positives to the total actual positives (true positives + false negatives). The recall for class 1 is approximately 0.52, which means the model correctly identifies about 52% of all actual positive instances.
- F1-score is the harmonic mean of precision and recall and provides a balance between the two. The F1-score for class 1 is approximately 0.63, reflecting the trade-off between precision and recall.