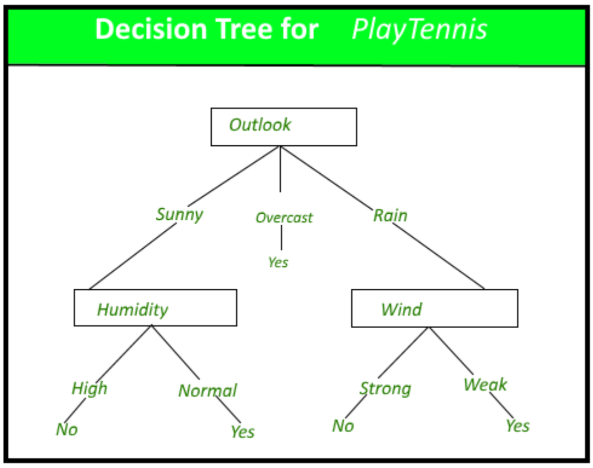
Experiment No. 3
Apply Decision Tree Algorithm on Adult Census Income Dataset and analyze the performance of the model
Date of Performance: 7-8-23
Date of Submission: 20-8-23

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:

Conclusion:

The Adult Census Income Dataset showed that the Decision Tree model performed well. It used one-hot encoding to handle categorical attributes and carried out the required data preprocessing to improve the efficacy of the model, including handling missing values, removing unnecessary tables, and dividing columns.

In order to improve the Decision Tree model's performance, hyperparameter tuning is essential since it gives the user control over the model's complexity by imposing certain parameters: We must refine the model using techniques like Gnd Search or Random Search in order to change hyperparameters such as max depth, min samples split, etc. in order to increase performance.

Attained an accuracy of 0.85, meaning that approximately 85% of the forecasts were accurate.

True positive = 9860, False positive = 481 on the confusion matrix. 1823 is a fake negative. Actual negative is 1646.

Precision: A precision of 0.84 indicates that around 0.77 of the occurrences that were anticipated as 0 were also projected to be 1.

Remember The model was able to catch the occurrences for 0 and 1 with a recall of 0.95 and 0.47, respectively.

F1 Score: In terms of the model's performance, the FI score of 0.90 represents the mean of accuracy and recall for 0 and 0.59 represents the mean of precision and recall for 1.

11__ml_exp3ipynb.py

```
# -*- coding: utf-8 -*-
"""11_.ML_Exp3ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1lE2yKcCkAVVyA01IS4cD8yhI-Wfd2Kh4
"""

# Commented out IPython magic to ensure Python compatibility.
# Import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# %matplotlib inline
# To ignore warning messages
import warnings
warnings.filterwarnings('ignore')
# Adult dataset path
adult_dataset_path = "/content/adult.csv"

# Function for loading adult dataset
def load_adult_data(adult_path=adult_dataset_path):
csv_path = os.path.join(adult_path)
return pd.read_csv(csv_path)

# Calling load adult function and assigning to a new variable df
df = load_adult_data()
# load top 3 rows values from adult dataset
df.head(3)

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())

df.info()

df.describe()

df.head()

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

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

```
df_check_missing_occupation

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

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

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

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

# 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()

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

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

# 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()

# look at column type
df.info()

# 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()

# 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(3)
```

```
y.head(3)

# 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()

# 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)

# 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))

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

!pip install my-package

!pip install pydotplus

# 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

!pip install graphviz

# 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)

# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()

# 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()

# 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)

# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()

# 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)

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

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

# 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)

# accuracy score
clf_gini.score(X_test,y_test)

#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))
# plotting tree with max_depth=3
dot_data = StringIO()
export_graphviz(clf_gini,
out_file=dot_data,feature_names=features,filled=True,rounded=True)
```

```
# classification metrics
from sklearn.metrics import classification_report,confusion_matrix
y_pred = clf_gini.predict(X_test)
print(classification_report(y_test, y_pred))
# confusion matrix
print(confusion_matrix(y_test,y_pred))
```