PRATIMA SURESH KUMAR

CIS 508 : MACHINE LEARNING IN BUSINESS

INDIVIDUAL ASSIGNMENT 2

FRAUD DETECTION

OBJECTIVE: Detecting if an insurance claim was fraudulent or not

STRATEGY USED:

- Using hyperparameter tuning of the two classifiers – Decision tree and Random Forest we try to determine the optimal set of parameters such that the model performs the best.
- Used random and grid search methods for hyperparameter tuning.
- Experimented k-cross folds cross-validation to train and test the model.

APPROACH:

Step 1: Read the test and train data. Understand the Understand the dataset by identifying the number of columns and rows and datatypes.

Python function used: .info() and .describe()

Step 2: Understand the datatype of each column and especially note down the categorical type columns.

Step 3: As most of ML algorithms cannot handle categorical variables, one hot encoding is used to encode the categorical features, except the target column that is to be predicted. The first step is to integrate training and test data to maintain uniformity in encoding categorical features.

Step 4: Construct a default decision tree classifier and random forest classifier for both data sets and obtain an accuracy score for the decision tree.

Perform hyperparameter tuning for the decision tree classifier. The below screenshot represents the specific code snippet for hyperparameter tuning for both classifiers for respective datasets.

4.1 Hyperparameter Tuning for Fraud Detection DS

4.1.1 Decision Tree Classifier

```
] #Hyperparameter tuning done for decision tree classifier
  parameters={'min_samples_split' : range(10,100,10),'max_depth': range(1,20,2)}
  clf_random = RandomizedSearchCV(clf,parameters,n_iter=15)
  clf_random.fit(X_train1, y_train)
  grid_parm=clf_random.best_params_
  print(grid_parm)
```

```
{'min_samples_split': 50, 'max_depth': 1}
accuracy Score (training) after hypertuning for Decision Tree:1.000000
Confusion Matrix after hypertuning for Decision Tree
[[12420     0]
 [    0   498]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12420
           1       1.00      1.00      1.00       498
```

4.1.2 Random Forest Classifier

```
11] #Hyperparameter tuning for random forest classifier
    rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=60)
    rfc_random.fit(X_train1, y_train)
    grid_parm_rfc=rfc_random.best_params_
    print(grid_parm_rfc)
```

```
{'min_samples_split': 10, 'max_depth': 19}
```

```
accuracy Score (training) after hypertuning for Random Forest:1.000000
Confusion Matrix after hypertuning for Random Forest:
[[12420     0]
 [    0   498]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12420
           1       1.00      1.00      1.00       498

    accuracy                           1.00     12918
   macro avg       1.00      1.00      1.00     12918
weighted avg       1.00      1.00      1.00     12918
```

4.2 Hyperparameter Tuning for Target Marketing Data set

4.2.1 Decision Tree Classifier

```
#Hyperparameter tuning done for decision tree classifier
parameters={'min_samples_split' : range(10,100,10),'max_depth': range(1,20,2)}
clf_random = RandomizedSearchCV(clf,parameters,n_iter=15)
clf_random.fit(X_train1, y_train)
grid_parm=clf_random.best_params_
print(grid_parm)
```

```
{'min_samples_split': 20, 'max_depth': 19}
```

```
accuracy Score (training) after hypertuning for Decision Tree:1.000000
Confusion Matrix after hypertuning for Decision Tree
[[39922     0]
 [    0  5289]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39922
           1       1.00      1.00      1.00      5289

    accuracy                           1.00     45211
   macro avg       1.00      1.00      1.00     45211
weighted avg       1.00      1.00      1.00     45211
```

4.2.2 Random Forest Classifier

```
[14] #Hyperparameter tuning for random forest classifier
     rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=60)
     rfc_random.fit(X_train1, y_train)
     grid_parm_rfc=rfc_random.best_params_
     print(grid_parm_rfc)

     {'min_samples_split': 70, 'max_depth': 15}
```

```
accuracy Score (training) after hypertuning for Random Forest:1.000000
Confusion Matrix after hypertuning for Random Forest:
[[39922     0]
 [    0  5289]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39922
           1       1.00      1.00      1.00      5289

    accuracy                           1.00     45211
   macro avg       1.00      1.00      1.00     45211
weighted avg       1.00      1.00      1.00     45211
```

```
accuracy Score (training) after hypertuning for Random Forest:1.000000
Confusion Matrix after hypertuning for Random Forest:
[[39922     0]
 [    0  5289]]
=== Classification Report ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39922
           1       1.00      1.00      1.00      5289

    accuracy                           1.00     45211
   macro avg       1.00      1.00      1.00     45211
weighted avg       1.00      1.00      1.00     45211
```

Step 5: Obtained AUC values after initializing the decision tree classifier and random classifier with the parameters obtained post-hyper tuning, for both data sets. Also performed cross-validation using cv= 10 and scoring =" roc_auc"

```
#get cross-validation report
clf_cv_score = cross_val_score(clf, X_train1, y_train, cv=10, scoring="roc_auc")
```

```
=== All AUC Scores ===
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]


=== Mean AUC Score ===
Mean AUC Score - Decision Tree:  1.0
```

I obtained the same result of score 1 for both data sets for both the classifiers.

PRATIMA SURESH KUMAR

OBSERVATIONS

Unique Observations during Hyper parameter Tuning: Fraud Detection Data Set

Before Hyper parameter tuning

```
accuracy Score (training) for RandomForest:0.999923
Confusion Matrix for Random Forest:
[[12420     0]
 [    1   497]]
```

Hyper Parameter Tuning

```
#Hyperparameter tuning for random forest classifier
rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=40)
rfc_random.fit(X_train1, y_train)
grid_parm_rfc=rfc_random.best_params_
print(grid_parm_rfc)
```

```
{'min_samples_split': 10, 'max_depth': 17}
```

After Hyper parameter tuning

```
accuracy Score (training) after hypertuning for Random Forest:1.000000
Confusion Matrix after hypertuning for Random Forest:
[[12420     0]
 [    0   498]]
```

There is improvement in the accuracy of the model post hyper parameter tuning of the Random Forest classifier for the Fraud Detection dataset.

CODE

Fraud detection Code

```
# To upload our datasets from our working directory we need to mount
our drive contents to the colab environment.
# For the code to do so you can search “mount” in code snippets or use
the code given below.
# Our entire drive contents are now mounted on colab at the location
“/gdrive”.
from google.colab import drive
drive.mount('/gdrive')
#Change current working directory to gdrive
%cd /gdrive
!pip install vecstack
from vecstack import stacking
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score #works
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
#from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from collections import Counter #for Smote,

import warnings
warnings.filterwarnings("ignore")
trainfile = r'/gdrive/My Drive/Colab
Notebooks/week3_bank/TrainInsuranceFraud.csv'
train_data = pd.read_csv(trainfile)
testfile = r'/gdrive/My Drive/Colab
Notebooks/week3_bank/TestInsuranceFraud.csv'
test_data = pd.read_csv(testfile)
print(train_data.shape)
print(test_data.shape)
print(train_data.head())
# Finding categorical columns is your DataFrame
categorical_columns = train_data.select_dtypes(include=['object',
'category']).columns
print(categorical_columns)
```

```
categorical_columns = test_data.select_dtypes(include=['object',
'category']).columns
print(categorical_columns)
#Do one-hot encoding of categorical variables

categoricalFeatures = [
    'MONTH', 'DAYOFWEEK', 'MAKE', 'ACCIDENTAREA', 'DAYOFWEEKCLAIMED',
    'MONTHCLAIMED', 'SEX', 'MARITALSTATUS', 'AGEOFPOLICYHOLDER',
    'POLICEREPORTFILED', 'WITNESSPRESENT', 'AGENTTYPE',
    'NUMBEROFSUPPLIMENTS', 'ADDRESSCHANGE_CLAIM', 'NUMBEROFCARS',
    'BASEPOLICY','FAULT', 'POLICYTYPE', 'VEHICLECATEGORY',
'VEHICLEPRICE',
       'DAYS_POLICY_ACCIDENT', 'DAYS_POLICY_CLAIM',
'PASTNUMBEROFCLAIMS',
       'AGEOFVEHICLE'
]

#Combine Train and test for one Hot Encoding
combined_Data = pd.concat([train_data,test_data], keys=[0,1])

#Do one Hot encoding for categorical features
combined_Data["FRAUDFOUND"]=combined_Data["FRAUDFOUND"].map({"Yes":1,"N
o":0})
combined_Data =
pd.get_dummies(combined_Data,columns=categoricalFeatures)
print(combined_Data['FRAUDFOUND'])

#Separate Train data and test data
X_train = combined_Data.xs(0)
X_test = combined_Data.xs(1)
y_train=X_train["FRAUDFOUND"]
y_test=X_test["FRAUDFOUND"]
X_train1=X_train.iloc[:, :-1].copy()
X_test1=X_test.iloc[:, :-1].copy()

#CONSTRUCT DEFAULT DECISION TREE AND OBTAIN RESPECTIVE ACCURACY
clf = DecisionTreeClassifier()
clf.fit(X_train1,y_train)
clf_predict=clf.predict(X_test1)
print("accuracy Score (training) for Decision
Tree:{0:6f}".format(clf.score(X_test1,y_test)))
print("Confusion Matrix for Decision Tree")
print(confusion_matrix(y_test,clf_predict))
#Hyperparameter tuning done for decision tree classifier
parameters={'min_samples_split' : range(10,100,10),'max_depth':
range(1,20,2)}
clf_random = RandomizedSearchCV(clf,parameters,n_iter=15)
clf_random.fit(X_train1, y_train)
```

```
grid_parm=clf_random.best_params_
print(grid_parm)

#Using the parameters obtained from HyperParameterTuning in the
DecisionTreeClassifier
clf = DecisionTreeClassifier(**grid_parm)
clf.fit(X_train1,y_train)
clf_predict = clf.predict(X_test1)

#Obtain accuracy ,confusion matrix,classification report and AUC values
for the result above.
print("accuracy Score (training) after hypertuning for Decision
Tree:{0:6f}".format(clf.score(X_test1,y_test)))
print("Confusion Matrix after hypertuning for Decision Tree")
print(confusion_matrix(y_test,clf_predict))
print("=== Classification Report ===")
print(classification_report(y_test,clf_predict))

#get cross-validation report
clf_cv_score = cross_val_score(clf, X_train1, y_train, cv=10,
scoring="roc_auc")
print("=== All AUC Scores ===")
print(clf_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Decision Tree: ",clf_cv_score.mean())
#Construct Random Forest Model

rfc = RandomForestClassifier()
rfc.fit(X_train1, y_train)
rfc_predict=rfc.predict(X_test1)
print("accuracy Score (training) for
RandomForest:{0:6f}".format(rfc.score(X_test1,y_test)))
print("Confusion Matrix for Random Forest:")
print(confusion_matrix(y_test,rfc_predict))
#Hyperparameter tuning for random forest classifier
rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=40)
rfc_random.fit(X_train1, y_train)
grid_parm_rfc=rfc_random.best_params_
print(grid_parm_rfc)
#Construct Random Forest with best parameters
rfc= RandomForestClassifier(**grid_parm_rfc)
rfc.fit(X_train1,y_train)
rfc_predict = rfc.predict(X_test1)
print("accuracy Score (training) after hypertuning for Random
Forest:{0:6f}".format(rfc.score(X_test1,y_test)))
print("Confusion Matrix after hypertuning for Random Forest:")
print(confusion_matrix(y_test,rfc_predict))
```

```
print("=== Classification Report ===")
print(classification_report(y_test,rfc_predict))
#get cross-validation report
rfc_cv_score = cross_val_score(rfc, X_train1, y_train, cv=10,
scoring="roc_auc")
print("=== All AUC Scores ===")
print(rfc_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ",rfc_cv_score.mean())
```

Target Marketing

```
# To upload our datasets from our working directory we need to mount
our drive contents to the colab environment.
# For the code to do so you can search "mount" in code snippets or use
the code given below.
# Our entire drive contents are now mounted on colab at the location
"/gdrive".
from google.colab import drive
drive.mount('/gdrive')
#Change current working directory to gdrive
%cd /gdrive
!pip install vecstack
from vecstack import stacking
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score #works
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
#from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from collections import Counter #for Smote,

import warnings
warnings.filterwarnings("ignore")
trainfile = r'/gdrive/My Drive/Colab
Notebooks/week3_bank/TargetMarketingTrain.csv'
train_data = pd.read_csv(trainfile)
testfile = r'/gdrive/My Drive/Colab
Notebooks/week3_bank/TargetMarketingTest.csv'
```

```
test_data = pd.read_csv(testfile)
print(train_data.shape)
print(test_data.shape)
print(train_data.head())
print(test_data.head())
# Finding categorical columns is your DataFrame
categorical_columns = train_data.select_dtypes(include=['object',
'category']).columns
print(categorical_columns)

#Do one-hot encoding of categorical variables

categoricalFeatures = ['job', 'marital', 'education', 'default',
'housing', 'loan', 'contact',
       'month', 'poutcome']
#Combine Train and test for one Hot Encoding
combined_Data = pd.concat([train_data,test_data], keys=[0,1])

#Do one Hot encoding for categorical features
combined_Data["y"]=combined_Data["y"].map({"yes":1,"no":0})
combined_Data =
pd.get_dummies(combined_Data,columns=categoricalFeatures)
print(combined_Data["y"])

#Separate Train data and test data
X_train = combined_Data.xs(0)
X_test = combined_Data.xs(1)
y_train=X_train["y"]
y_test=X_test["y"]
X_train1=X_train.iloc[:, :-1].copy()
X_test1=X_test.iloc[:, :-1].copy()
#CONSTRUCT DEFAULT DECISION TREE AND OBTAIN RESPECTIVE ACCURACY
clf = DecisionTreeClassifier()
clf.fit(X_train1,y_train)
clf_predict=clf.predict(X_test1)
print("accuracy Score (training) for Decision
Tree:{0:6f}".format(clf.score(X_test1,y_test)))
print("Confusion Matrix for Decision Tree")
print(confusion_matrix(y_test,clf_predict))
#Hyperparameter tuning done for decision tree classifier
parameters={'min_samples_split' : range(10,100,10),'max_depth':
range(10,20,15)}
clf_random = RandomizedSearchCV(clf,parameters,n_iter=15)
clf_random.fit(X_train1, y_train)
grid_parm=clf_random.best_params_
print(grid_parm)
#Using the parameters obtained from HyperParameterTuning in the
DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier(**grid_parm)
clf.fit(X_train1,y_train)
clf_predict = clf.predict(X_test1)

#Obtain accuracy ,confusion matrix,classification report and AUC values
for the result above.
print("accuracy Score (training) after hypertuning for Decision
Tree:{0:6f}".format(clf.score(X_test1,y_test)))
print("Confusion Matrix after hypertuning for Decision Tree")
print(confusion_matrix(y_test,clf_predict))
print("=== Classification Report ===")
print(classification_report(y_test,clf_predict))

#get cross-validation report
clf_cv_score = cross_val_score(clf, X_train1, y_train, cv=10,
scoring="roc_auc")
print("=== All AUC Scores ===")
print(clf_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Decision Tree: ",clf_cv_score.mean())
#Construct Random Forest Model

rfc = RandomForestClassifier()
rfc.fit(X_train1, y_train)
rfc_predict=rfc.predict(X_test1)
print("accuracy Score (training) for
RandomForest:{0:6f}".format(rfc.score(X_test1,y_test)))
print("Confusion Matrix for Random Forest:")
print(confusion_matrix(y_test,rfc_predict))
#Hyperparameter tuning for random forest classifier
rfc_random = RandomizedSearchCV(rfc,parameters,n_iter=40)
rfc_random.fit(X_train1, y_train)
grid_parm_rfc=rfc_random.best_params_
print(grid_parm_rfc)
#Construct Random Forest with best parameters
rfc= RandomForestClassifier(**grid_parm_rfc)
rfc.fit(X_train1,y_train)
rfc_predict = rfc.predict(X_test1)
print("accuracy Score (training) after hypertuning for Random
Forest:{0:6f}".format(rfc.score(X_test1,y_test)))
print("Confusion Matrix after hypertuning for Random Forest:")
print(confusion_matrix(y_test,rfc_predict))
print("=== Classification Report ===")
print(classification_report(y_test,rfc_predict))
#get cross-validation report
rfc_cv_score = cross_val_score(rfc, X_train1, y_train, cv=29,
scoring="roc_auc")
```

```
print("=== All AUC Scores ===")
print(rfc_cv_score)
print('\n')
print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ",rfc_cv_score.mean())
```