ADS-Code_Supplement

November 5, 2024

[2]:
```
# # 1 Preliminary work
# Import the required libraries
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split as TTS
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score as ACC
from matplotlib import pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import GridSearchCV
```

[6]:
```
#Import data
filePath = r"E:\UIBE\train.csv"
dataBase = pd.read_csv(filePath, encoding='utf-8')
#View data dimensions
print(dataBase.shape)
```

```
(8631, 18)
```

[8]:
```
# # 2 Data cleansing and organization
### The original data had character data, missing values, so the data needed to␣
 ↪be processed.
#View data
print(dataBase.head())
#View the amount of missing value data
nullInfo = dataBase.isnull().sum()
print(nullInfo)
```

```
                                                                    \
0  Returning_Visitor      136.00  0.000000   0.0    2         138.0   Mar
1  Returning_Visitor       34.25  0.007310   0.0    2          68.5   Nov
2  Returning_Visitor        0.00  0.013636   0.0    2           0.0   Dec
3  Returning_Visitor        0.00  0.007018   0.0    2           0.0   Nov
4  Returning_Visitor        0.00  0.200000   0.0    4           0.0  June

                                                                    \
0  0.009697      54   2631.100000     2     3.0  False  0.0   1   1.0     2.0
1  0.013520     385  14505.727250     3     5.0  False  0.0   3   3.0     2.0
```

```
2  0.025515     111   1624.750000     2     0.0  False  0.0   1   2.0     0.0
3  0.027875      57   1091.005876     2     0.0  False  0.0   6   2.0     0.0
4  0.200000       1      0.000000     3     0.0  False  0.0   9   2.0     0.0
```

```
0     0
1     0
2     0
3     0
4     0
             1
           2
              0
             0
              0
          0
               0
              1
            0
          0
              0
             1
               2
              0
               0
              1
             1
              0
dtype: int64
```

[10]:
```
# Use plurality to populate data
for I in nullInfo.keys():
    if nullInfo[I] != 0:
        dataBase[I] = dataBase[I].fillna(dataBase[I].mode()[0])
# Look at the data again
nullInfo = dataBase.isnull().sum()
print(nullInfo)
```

```
             0
           0
              0
             0
              0
          0
               0
              0
            0
          0
```

```
                0
               0
                 0
                0
                 0
               0
              0
                0
dtype: int64
```

```
C:\Users\WilliamNiu\AppData\Local\Temp\ipykernel_6908\997320758.py:4:
FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is
deprecated and will change in a future version. Call
result.infer_objects(copy=False) instead. To opt-in to the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  dataBase[I] = dataBase[I].fillna(dataBase[I].mode()[0])
```

[12]:
```
#Category encoding of character type data
strColumns = ['   ', ' ', '    ', ' ']
for I in strColumns:
    setList = list(set(dataBase[I].values.tolist()))
    dataBase[I] = [setList.index(W) for W in dataBase[I]]
print(dataBase.head())
```

```
                                                    \
0     0      136.00  0.000000   0.0    2         138.0   9  0.009697      54
1     0       34.25  0.007310   0.0    2          68.5   8  0.013520     270
2     0        0.00  0.013636   0.0    2           0.0   0  0.025515     111
3     0        0.00  0.007018   0.0    2           0.0   8  0.027875      57
4     0        0.00  0.200000   0.0    4           0.0  10  0.200000       1


0   2631.100000     2     3.0   0  0.0   1   1.0     2.0     0
1  14505.727250     3     5.0   0  0.0   3   3.0     2.0     0
2   1624.750000     2     0.0   0  0.0   1   2.0     0.0     0
3   1091.005876     2     0.0   0  0.0   6   2.0     0.0     0
4      0.000000     3     0.0   0  0.0   9   2.0     0.0     0
```

[14]:
```
# # 3 Modeling with Random Forests
# Remove feature X and label Y
X = dataBase.drop(['   '], axis=1)
Y = dataBase[['   ']].values.ravel()
print(X)
print(Y)
```

```
                                                  \
0        0  136.000000  0.000000   0.0    2         138.0   9  0.009697
1        0   34.250000  0.007310   0.0    2          68.5   8  0.013520
2        0    0.000000  0.013636   0.0    2           0.0   0  0.025515
```

```
3         0    0.000000  0.007018   0.0    2           0.0   8  0.027875
4         0    0.000000  0.200000   0.0    4           0.0  10  0.200000
…    …          …        …    …  …          …   ..       …
8626      0    0.000000  0.062500   0.0    2           0.0   5  0.118750
8627      0    0.000000  0.066667   0.0    2           0.0   2  0.100000
8628      0    0.000000  0.016667   0.0    2          30.0   0  0.026389
8629      0  399.333333  0.003673   0.0    2           0.0   9  0.026757
8630      3   34.000000  0.000000   0.0    4           0.0   8  0.009524


0          54   2631.100000     2     3.0   0  0.0   1   1.0     2.0
1         270  14505.727250     3     5.0   0  0.0   3   3.0     2.0
2         111   1624.750000     2     0.0   0  0.0   1   2.0     0.0
3          57   1091.005876     2     0.0   0  0.0   6   2.0     0.0
4           1      0.000000     3     0.0   0  0.0   9   2.0     0.0
…       …            …   …     …   ..   …   ..   …      …
8626        4     86.360000     1     0.0   1  0.0   4   3.0     0.0
8627        6    105.000000     3     0.0   0  0.0   4   3.0     0.0
8628       23    625.300000     3     3.0   0  0.0   3   2.0     0.0
8629       46   1814.583333     2     0.0   0  0.0   2   2.0     4.0
8630       18    271.166667     2     0.0   0  0.0   9   2.0     3.0

[8631 rows x 17 columns]
[0 0 0 … 0 0 0]
```

[16]:
```
#Divided according to the ratio of training set:test set = 7:3
xTrain, xTest, yTrain, yTest = TTS(X, Y, test_size=0.3, random_state=114)#
print(xTrain.shape)
print(xTest.shape)
```

```
(6041, 17)
(2590, 17)
```

[18]:
```
# ## 3.1 Modeling of Default Parameters
RF = RandomForestClassifier(random_state=2023)
RF.fit(xTrain, yTrain)
RFPre = RF.predict(xTest)
RFScore = RF.predict_proba(xTest)[:,-1]
RFAcc = ACC(yTest, RFPre)
print('  :', RFAcc)
print('   : 0 neutral or dissatisfied, 1 satisfied\n')
print(classification_report(yTest, RFPre))
```

```
 : 0.9034749034749034
  : 0 neutral or dissatisfied, 1 satisfied

              precision    recall  f1-score   support

           0       0.93      0.96      0.94      2176
```

```
           1       0.74      0.60      0.67       414

    accuracy                           0.90      2590
   macro avg       0.84      0.78      0.81      2590
weighted avg       0.90      0.90      0.90      2590
```

[20]:
```
# ## 3.2 Hyperparameter Optimization
RFParam = {'max_depth':[9,10],
           'n_estimators':[100,200,300]}
RF = RandomForestClassifier(random_state=2023)
RFGrid = GridSearchCV(RF, param_grid = RFParam, scoring='accuracy', verbose=2)
RFGrid.fit(xTrain, yTrain)
RFPre = RFGrid.predict(xTest)
RFScore = RFGrid.predict_proba(xTest)[:,-1]
RFACCValue = ACC(yTest, RFPre)
print('    :{}'.format(RFACCValue))
print(classification_report(yTest, RFPre))
RFFPR, RFTPR, _ = roc_curve(yTest,RFScore)
RFAUC = roc_auc_score(yTest,RFScore)
print(RFGrid.best_params_)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] END …max_depth=9, n_estimators=100; total time=   0.7s
[CV] END …max_depth=9, n_estimators=100; total time=   0.6s
[CV] END …max_depth=9, n_estimators=100; total time=   0.6s
[CV] END …max_depth=9, n_estimators=100; total time=   0.7s
[CV] END …max_depth=9, n_estimators=100; total time=   0.7s
[CV] END …max_depth=9, n_estimators=200; total time=   1.3s
[CV] END …max_depth=9, n_estimators=200; total time=   1.4s
[CV] END …max_depth=9, n_estimators=200; total time=   1.4s
[CV] END …max_depth=9, n_estimators=200; total time=   1.4s
[CV] END …max_depth=9, n_estimators=200; total time=   1.4s
[CV] END …max_depth=9, n_estimators=300; total time=   2.2s
[CV] END …max_depth=9, n_estimators=300; total time=   2.1s
[CV] END …max_depth=9, n_estimators=300; total time=   2.1s
[CV] END …max_depth=9, n_estimators=300; total time=   2.2s
[CV] END …max_depth=9, n_estimators=300; total time=   2.2s
[CV] END …max_depth=10, n_estimators=100; total time=   0.7s
[CV] END …max_depth=10, n_estimators=100; total time=   0.7s
[CV] END …max_depth=10, n_estimators=100; total time=   0.7s
[CV] END …max_depth=10, n_estimators=100; total time=   0.7s
[CV] END …max_depth=10, n_estimators=100; total time=   0.7s
[CV] END …max_depth=10, n_estimators=200; total time=   1.5s
[CV] END …max_depth=10, n_estimators=200; total time=   1.5s
[CV] END …max_depth=10, n_estimators=200; total time=   1.4s
[CV] END …max_depth=10, n_estimators=200; total time=   1.4s
[CV] END …max_depth=10, n_estimators=200; total time=   1.5s
```

```
[CV] END …max_depth=10, n_estimators=300; total time=   2.2s
[CV] END …max_depth=10, n_estimators=300; total time=   2.2s
[CV] END …max_depth=10, n_estimators=300; total time=   2.2s
[CV] END …max_depth=10, n_estimators=300; total time=   2.2s
[CV] END …max_depth=10, n_estimators=300; total time=   2.2s
    :0.9054054054054054
                  precision    recall  f1-score   support

               0       0.93      0.96      0.94      2176
               1       0.75      0.61      0.67       414

        accuracy                           0.91      2590
       macro avg       0.84      0.79      0.81      2590
    weighted avg       0.90      0.91      0.90      2590

    {'max_depth': 9, 'n_estimators': 100}
```

[24]:
```
# ## 3.3 Bringing in the optimized parameters
RF_ = RandomForestClassifier(max_depth=RFGrid.best_params_['max_depth'],
                             n_estimators=RFGrid.best_params_['n_estimators'],
                            random_state=2023)
RF_.fit(xTrain, yTrain)


RF_Pre = RF_.predict(xTest)
RF_Score = RF_.predict_proba(xTest)[:,-1]

RF_Acc = ACC(yTest, RF_Pre)
print('  :', RF_Acc)
print('   : 0 neutral or dissatisfied, 1 satisfied\n')
print(classification_report(yTest, RF_Pre))
```

```
  : 0.9054054054054054
   : 0 neutral or dissatisfied, 1 satisfied

                  precision    recall  f1-score   support

               0       0.93      0.96      0.94      2176
               1       0.75      0.61      0.67       414

        accuracy                           0.91      2590
       macro avg       0.84      0.79      0.81      2590
    weighted avg       0.90      0.91      0.90      2590
```

[26]:
```
# # 4 Plotting the ROC curve
plt.figure(figsize=(10,8), dpi=100)
```

```
RFFPR, RFTPR, _ = roc_curve(yTest,RF_Score)# Calculate the positive-positive␣
 ↪rate False-positive rate
RFAUC = roc_auc_score(yTest,RF_Score)# Calculate AUC values

plt.plot(RFFPR, RFTPR, color='black', linewidth=2, label='Random Forest AUC={}'.
 ↪format(round(RFAUC, 3)), linestyle='-')#
plt.plot([0,1], [0, 1], linestyle='dashed', linewidth=3, color='royalblue')#
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.show()
```