ROC_and_AUC_Implementation

November 1, 2022

0.0.1 Select the Right Threshold values using ROC Curve

Import required libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
↪roc_auc_score
%matplotlib inline
```

- make_classification will randomly generate classification problems dataset

Binary Classfication Dataset

Create Dataset

[2]:
```
X, y = make_classification(n_samples=2000, n_classes=2, weights=[1,1],␣
↪random_state=1)
```

Shape of the dataset

[3]:
```
X.shape
```

[3]:
```
(2000, 20)
```

[4]:
```
y
```

[4]:
```
array([0, 0, 0, …, 1, 1, 0])
```

Train Test Split

[5]:
```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1)
```

1 Model Creation

1.1 Random Forests

[6]:
```
## Apply RandomForestClassifier
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
ytrain_pred = rf_model.predict_proba(X_train)
ytest_pred = rf_model.predict_proba(X_test)
```

ROC-AUC

[7]:
```
print('RF train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:,1])))
print('RF test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:,1])))
```

```
RF train roc-auc: 1.0
RF test roc-auc: 0.9838222222222223
```

1.2 Logistic Regression

[8]:
```
log_classifier=LogisticRegression()
log_classifier.fit(X_train, y_train)
ytrain_pred = log_classifier.predict_proba(X_train)
ytest_pred = log_classifier.predict_proba(X_test)
```

ROC-AUC

[9]:
```
print('Logistic train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:
↪,1])))
print('Logistic test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:
↪,1])))
```

```
Logistic train roc-auc: 0.9863568922694498
Logistic test roc-auc: 0.9885777777777777
```

Higher the area, better the model

1.3 Adaboost Classifier

[10]:
```
ada_classifier=AdaBoostClassifier()
ada_classifier.fit(X_train, y_train)
ytrain_pred = ada_classifier.predict_proba(X_train)
ytest_pred = ada_classifier.predict_proba(X_test)
```

ROC-AUC

[11]:
```
print('Adaboost train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:
 ↪,1])))
print('Adaboost test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:
 ↪,1])))
```

```
Adaboost train roc-auc: 0.9975081174960356
Adaboost test roc-auc: 0.9826111111111111
```

1.4 KNNClassifier

[12]:
```
knn_classifier=KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
ytrain_pred = knn_classifier.predict_proba(X_train)
ytest_pred = knn_classifier.predict_proba(X_test)
```

ROC-AUC

[13]:
```
print('Adaboost train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:
 ↪,1])))
print('Adaboost test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:
 ↪,1])))
```

```
Adaboost train roc-auc: 0.981670071491109
Adaboost test roc-auc: 0.9426111111111111
```

2 Select the best threshold for maximum accuracy

[14]:
```
pred=[]
for model in [rf_model,log_classifier,ada_classifier,knn_classifier]:
    pred.append(pd.Series(model.predict_proba(X_test)[:,1]))
final_prediction=pd.concat(pred,axis=1).mean(axis=1)
print('test roc-auc: {}'.format(roc_auc_score(y_test,final_prediction)))
```

```
test roc-auc: 0.9851333333333333
```

2.1 Calculate FPR, TPR and Thresholds

[15]:
```
fpr, tpr, thresholds = roc_curve(y_test, final_prediction)
```

2.2 Calculate Acuuracy, FPR and TPR for each Thresholds

[16]:
```
accuracy_ls = []
TPR_ls = []
FPR_ls = []
for thres in thresholds:
    y_pred = np.where(final_prediction>thres,1,0)
    accuracy_ls.append(accuracy_score(y_test, y_pred, normalize=True))

    conf_mat = confusion_matrix(y_test,y_pred)
    true_positive = conf_mat[0][0]
    false_positive = conf_mat[0][1]
    false_negative = conf_mat[1][0]
    true_negative = conf_mat[1][1]
    TPR = true_positive/(true_positive + false_negative)
    FPR = false_positive/(false_positive + true_negative)
    TPR_ls.append(TPR)
    FPR_ls.append(FPR)

accuracy_ls = pd.concat([pd.Series(thresholds), pd.Series(accuracy_ls), pd.
 ↪Series(TPR_ls), pd.Series(FPR_ls) ], axis=1)
accuracy_ls.columns = ['thresholds', 'accuracy','TPR', 'FPR']
accuracy_ls.sort_values(by='accuracy', ascending=False, inplace=True)
accuracy_ls
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: RuntimeWarning:
invalid value encountered in long_scalars

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: RuntimeWarning:
invalid value encountered in long_scalars
```

[16]:

	thresholds	accuracy	TPR	FPR
31	0.442228	0.961667	0.975945	0.051780
32	0.434603	0.960000	0.975862	0.054839
30	0.449506	0.960000	0.972603	0.051948
29	0.471169	0.960000	0.969388	0.049020
26	0.541059	0.960000	0.960000	0.040000
28	0.472402	0.958333	0.966102	0.049180
27	0.526279	0.958333	0.962963	0.046205
25	0.573004	0.956667	0.950658	0.037162
24	0.573090	0.955000	0.947541	0.037288

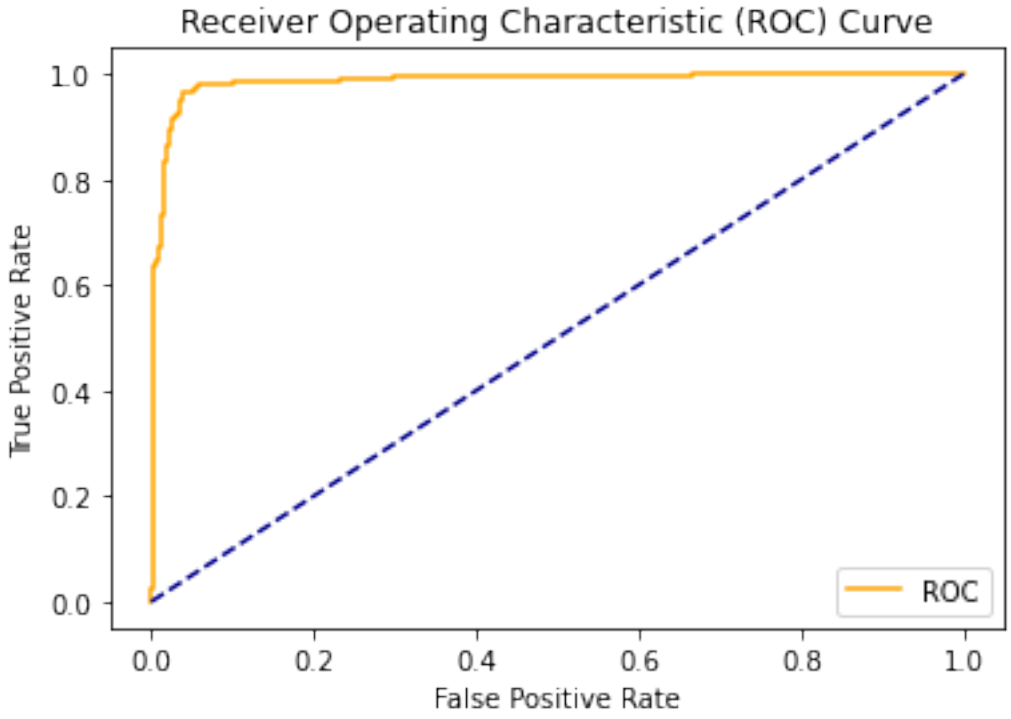
33	0.389348	0.948333	0.978648	0.078370
23	0.577362	0.946667	0.929487	0.034722
34	0.388797	0.946667	0.978571	0.081250
19	0.596143	0.945000	0.921136	0.028269
22	0.583277	0.945000	0.926518	0.034843
21	0.586754	0.945000	0.923810	0.031579
35	0.353073	0.943333	0.981884	0.089506
18	0.615300	0.943333	0.918239	0.028369
20	0.591564	0.943333	0.920886	0.031690
36	0.350866	0.941667	0.981818	0.092308
17	0.664935	0.936667	0.904321	0.025362
16	0.666477	0.935000	0.901538	0.025455
15	0.701374	0.923333	0.880240	0.022556
14	0.702881	0.921667	0.877612	0.022642
13	0.723937	0.910000	0.857558	0.019531
12	0.724075	0.908333	0.855072	0.019608
37	0.236984	0.878333	0.982979	0.189041
38	0.236404	0.876667	0.982906	0.191257
11	0.762871	0.860000	0.787234	0.017857
10	0.764745	0.858333	0.785146	0.017937
39	0.220051	0.856667	0.986364	0.218421
40	0.218969	0.855000	0.986301	0.220472
41	0.212131	0.850000	0.990654	0.227979
42	0.205984	0.848333	0.990610	0.229974
9	0.783212	0.831667	0.751899	0.014634
8	0.784852	0.830000	0.750000	0.014706
7	0.798056	0.820000	0.737624	0.010204
6	0.799524	0.818333	0.735802	0.010256
5	0.801628	0.816667	0.732843	0.005208
4	0.801999	0.815000	0.731051	0.005236
43	0.122835	0.666667	0.990196	0.399598
44	0.122835	0.665000	0.990099	0.400802
3	0.905775	0.511667	0.505902	0.000000
2	0.906233	0.510000	0.505051	0.000000
45	0.105484	0.501667	1.000000	0.499165
1	0.911881	0.500000	0.500000	NaN
0	1.911881	0.500000	0.500000	NaN

Plot ROC Curve

[17]:
```
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
```

```
    plt.show()
```

[18]:
```
plot_roc_curve(fpr,tpr)
```



Observation * Higher the AUC, better the model * Based on the accuracy score, our best results are:

<table>
<tr>
<th>thresholds</th>
<th>accuracy</th>
<th>TPR</th>
<th>FPR</th>
</tr>
<tr>
<td>0.442949</td>
<td>0.961667</td>
<td>0.975945</td>
<td>0.051780</td>
</tr>
</table>