[15]

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import r2_score, recall_score, f1_score,
roc_curve, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import pandas as pd
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn import metrics
```

[2]

```
def plot_roc_curve(y_train, preds_train, y_test, preds_test):
    plt.plot(metrics.roc_curve(y_train, preds_train)[0],
metrics.roc_curve(y_train, preds_train)[1],
             color = 'red', label='Train ROC Curve (area =
%0.5f)' % roc_auc_score(y_train, preds_train))
    plt.plot(metrics.roc_curve(y_test, preds_test)
[0],metrics.roc_curve(y_test, preds_test)[1],
             color = 'blue', label='Test ROC Curve (area =
%0.5f)' % roc_auc_score(y_test, preds_test))
    plt.plot([0, 2], [0, 2], color='black', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('AUC')
    plt.legend()
    plt.show()
    sns.set(style='white', rc={'figure.figsize':(10,10)})
```

[3]

```
def important_stats(y_true, y_pred_proba, summary):
    print("--------------------------------------------")
    y_pred_label = pd.Series(y_pred_proba)
    y_pred_label = y_pred_label.map(lambda x: 1 if x > 0.5 else
0)
    print(summary)
    reacll = recall_score(y_true, y_pred_label)
    print('recall:', reacll)
    f1_stat = f1_score(y_true, y_pred_label)
    print('f1_score:', f1_stat)
    accuracyScore= accuracy_score(y_true, y_pred_label)
    print('accuracy_score:', accuracyScore)
    fpr, tpr, thresholds = metrics.roc_curve(y_true,
y_pred_proba)
    auc = metrics.auc(fpr, tpr)
    print('AUC:', auc)
    matrix = pd.crosstab(y_true, y_pred_label, rownames=['True'],
colnames=['Predicted'], margins=True)
    print(matrix)
    print("------------------------------------------")
```

[4]
```
df =
pd.read_csv(f'{os.getcwd()}/data_clean/cleaned_match_data.csv')
```

[5]
```
df.head()
```

|   | Unnamed: 0 | match_id | duration | result | top_towers | mid_towe |
|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 2375 | 1 | 1 | 3 |
| **1** | 1 | 0 | 2375 | 0 | -1 | -3 |
| **2** | 2 | 0 | 2375 | 1 | 1 | 3 |
| **3** | 3 | 0 | 2375 | 0 | -1 | -3 |
| **4** | 4 | 1 | 2582 | 0 | -2 | -2 |

5 rows × 54 columns

[6]
```
df.shape
```

```
(97342, 54)
```

[83]
```
df.dtypes
```

```
Unnamed: 0               int64
match_id                 int64
duration                 int64
result                   int64
top_towers               int64
mid_towers               int64
bottom_towers            int64
ancient_status           int64
top_barracks             int64
mid_barracks             int64
bottom_barracks          int64
gold_total               int64
gold_max                 int64
gold_min                 int64
gold_std               float64
gold_spent_avg         float64
```

```
gold_spent_max           int64
gold_spent_min           int64
gold_spent_std         float64
kills_total              int64
deaths_total             int64
deaths_max               int64
deaths_min               int64
deaths_std             float64
assists_avg            float64
assists_max              int64
assists_min              int64
assists_std            float64
denies_avg             float64
denies_max               int64
denies_min               int64
denies_std             float64
last_hits_avg          float64
last_hits_max            int64
last_hits_min            int64
last_hits_std          float64
hero_damage_total        int64
hero_damage_max          int64
hero_damage_min          int64
hero_damage_std        float64
tower_damage_total       int64
tower_damage_max         int64
tower_damage_min         int64
tower_damage_std       float64
level_total              int64
level_max                int64
level_min                int64
level_std              float64
gold_buyback_avg       float64
gold_buyback_max       float64
gold_buyback_min       float64
gold_buyback_std       float64
teamfight_loss           int64
has_negative_chat         bool
dtype: object
```

[7]

```
df.head()
df = df.drop(columns = ['match_id','Unnamed: 0'])
```

[8]

```
df.corr()
```

| | duration | result | top_towers | mid_towers |
|---|---|---|---|---|
| **duration** | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| **result** | 0.000000 | 1.000000 | 0.780384 | 0.869120 |
| **top_towers** | 0.000000 | 0.780384 | 1.000000 | 0.807989 |

|  | duration | result | top_towers | mid_towers |
|---|---|---|---|---|
| **mid_towers** | 0.000000 | 0.869120 | 0.807989 | 1.000000 |
| **bottom_towers** | 0.000000 | 0.825153 | 0.792817 | 0.856402 |
| **ancient_status** | 0.000000 | 0.986854 | 0.790515 | 0.880311 |
| **top_barracks** | 0.000000 | 0.809052 | 0.880293 | 0.738965 |
| **mid_barracks** | 0.000000 | 0.865915 | 0.752831 | 0.801983 |
| **bottom_barracks** | 0.000000 | 0.942938 | 0.787292 | 0.914214 |
| **gold_total** | 0.230393 | 0.739365 | 0.548157 | 0.613873 |
| **gold_max** | 0.221347 | 0.657264 | 0.493350 | 0.550434 |
| **gold_min** | 0.167829 | 0.546885 | 0.399394 | 0.443538 |
| **gold_std** | 0.188805 | 0.543037 | 0.411468 | 0.459860 |
| **gold_spent_avg** | 0.785802 | 0.484111 | 0.449560 | 0.489240 |
| **gold_spent_max** | 0.688970 | 0.423778 | 0.394271 | 0.422551 |
| **gold_spent_min** | 0.656757 | 0.435685 | 0.407193 | 0.449461 |
| **gold_spent_std** | 0.484055 | 0.281615 | 0.260556 | 0.271514 |
| **kills_total** | 0.573466 | 0.533060 | 0.476218 | 0.559674 |
| **deaths_total** | 0.580572 | -0.524347 | -0.466332 | -0.548470 |
| **deaths_max** | 0.556368 | -0.439920 | -0.396471 | -0.470060 |
| **deaths_min** | 0.474530 | -0.509340 | -0.445814 | -0.519711 |
| **deaths_std** | 0.315636 | -0.114377 | -0.115589 | -0.146337 |
| **assists_avg** | 0.588018 | 0.447070 | 0.355174 | 0.441683 |
| **assists_max** | 0.556651 | 0.450628 | 0.367368 | 0.450488 |
| **assists_min** | 0.526481 | 0.365420 | 0.278232 | 0.355823 |
| **assists_std** | 0.350211 | 0.343699 | 0.299277 | 0.352546 |
| **denies_avg** | 0.089869 | 0.116800 | 0.133224 | 0.145273 |
| **denies_max** | 0.059878 | 0.091193 | 0.105878 | 0.114093 |
| **denies_min** | 0.081750 | 0.059385 | 0.061330 | 0.069323 |
| **denies_std** | 0.046099 | 0.083094 | 0.098017 | 0.105110 |

|  | duration | result | top_towers | mid_towers |
|---|---|---|---|---|
| **last_hits_avg** | 0.848630 | 0.165491 | 0.177155 | 0.177493 |
| **last_hits_max** | 0.736927 | 0.175365 | 0.184410 | 0.178215 |
| **last_hits_min** | 0.602318 | 0.068833 | 0.072592 | 0.087823 |
| **last_hits_std** | 0.633463 | 0.174293 | 0.181553 | 0.171999 |
| **hero_damage_total** | 0.716266 | 0.366720 | 0.337051 | 0.393060 |
| **hero_damage_max** | 0.584585 | 0.367620 | 0.333298 | 0.389706 |
| **hero_damage_min** | 0.493316 | 0.197360 | 0.189298 | 0.219091 |
| **hero_damage_std** | 0.428317 | 0.320831 | 0.285614 | 0.335256 |
| **tower_damage_total** | 0.179902 | 0.883605 | 0.815497 | 0.858416 |
| **tower_damage_max** | 0.192582 | 0.785854 | 0.731209 | 0.757441 |
| **tower_damage_min** | 0.006648 | 0.613200 | 0.560517 | 0.619049 |
| **tower_damage_std** | 0.199974 | 0.747728 | 0.696752 | 0.716737 |
| **level_total** | 0.867513 | 0.320456 | 0.278445 | 0.324326 |
| **level_max** | 0.776655 | 0.318169 | 0.296044 | 0.338097 |
| **level_min** | 0.780949 | 0.293050 | 0.246091 | 0.292922 |
| **level_std** | 0.063222 | 0.059137 | 0.088417 | 0.083004 |
| **gold_buyback_avg** | -0.585443 | 0.189751 | 0.141134 | 0.142127 |
| **gold_buyback_max** | -0.213388 | 0.061799 | 0.036464 | 0.024962 |
| **gold_buyback_min** | -0.602811 | 0.161717 | 0.121168 | 0.131745 |
| **gold_buyback_std** | 0.582065 | -0.154209 | -0.118323 | -0.133527 |
| **teamfight_loss** | 0.242851 | -0.565136 | -0.517809 | -0.609866 |
| **has_negative_chat** | 0.047286 | 0.000000 | 0.000000 | 0.000000 |

52 rows × 52 columns

[9]

```
#Drop highly correlated features
corr_matrix = df.corr().abs()
upper =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(
np.bool))
```

```
to_drop = [column for column in upper.columns if
any(upper[column]>0.6)]
df.drop(to_drop,axis=1,inplace=True)
df.head()
```

|   | duration | result | deaths_total | denies_avg | denies_min | level_ |
|---|---|---|---|---|---|---|
| **0** | 2375 | 1 | 17 | 6.0 | 1 | 3.0332 |
| **1** | 2375 | 0 | 52 | 7.6 | 0 | 2.6382 |
| **2** | 2375 | 1 | 17 | 6.0 | 1 | 3.0332 |
| **3** | 2375 | 0 | 52 | 7.6 | 0 | 2.6382 |
| **4** | 2582 | 0 | 53 | 5.4 | 0 | 4.4091 |

[10]

```
x_train, x_test, y_train, y_test = train_test_split(
    df.drop(columns = ['result','duration']),
    df['result'],
    test_size=0.2,
    random_state=1
)
```

[11]

```
x_train, x_val, y_train, y_val = train_test_split(x_train,
y_train, test_size = 0.2, random_state = 1)
```

## XGBoost

[12]

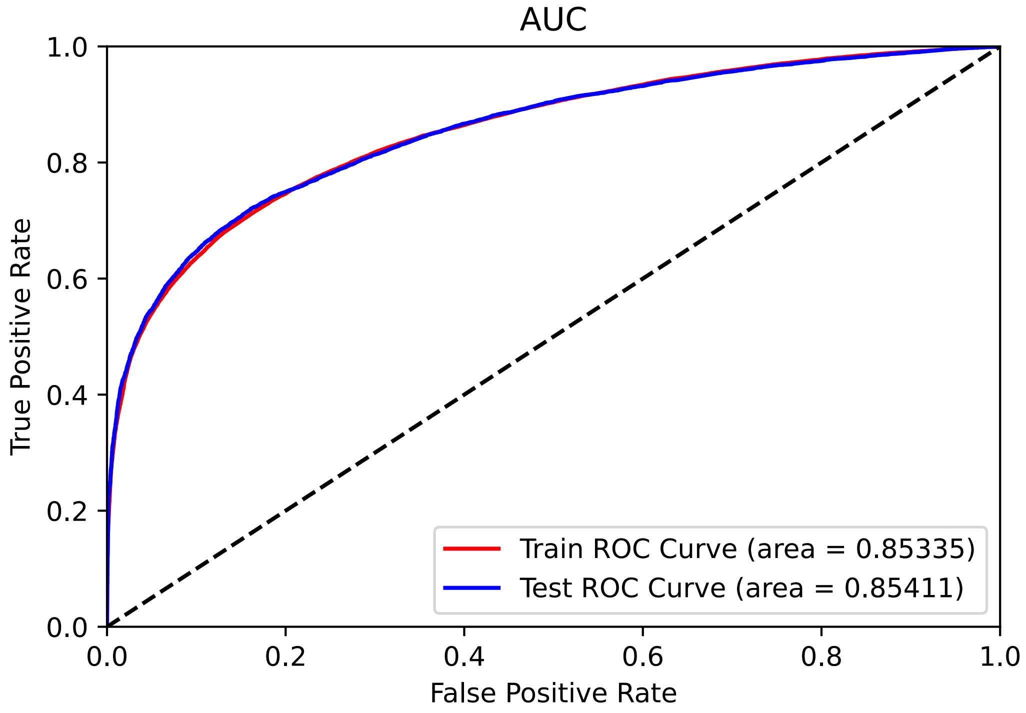
```
xgb = XGBClassifier(random_state=0, n_jobs=-1, learning_rate=0.1,
                      n_estimators=100, max_depth=3)
model = xgb.fit(x_train, y_train)
y_pred_test = xgb.predict_proba(x_test)[:, 1]
y_pred_train = xgb.predict_proba(x_train)[:, 1]
important_stats(y_train, y_pred_train, "train result summary: ")
important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.694505071254333
f1_score: 0.7549368501849137
accuracy_score: 0.7745031943240553
AUC: 0.853348040516045
Predicted      0      1     All
```

```
True
0           11525   8389  19914
1           11625   8272  19897
All         23150  16661  39811
------------------------------------------
------------------------------------------
test result summary:
recall: 0.6991903248949473
f1_score: 0.7596881959910914
accuracy_score: 0.7783142431557861
AUC: 0.8541094150037003
Predicted     0     1   All
True
0           1146   849  1995
1           1151   864  2015
All         2297  1713  4010
------------------------------------------
```

[13]

```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```



## GBDT

[64]

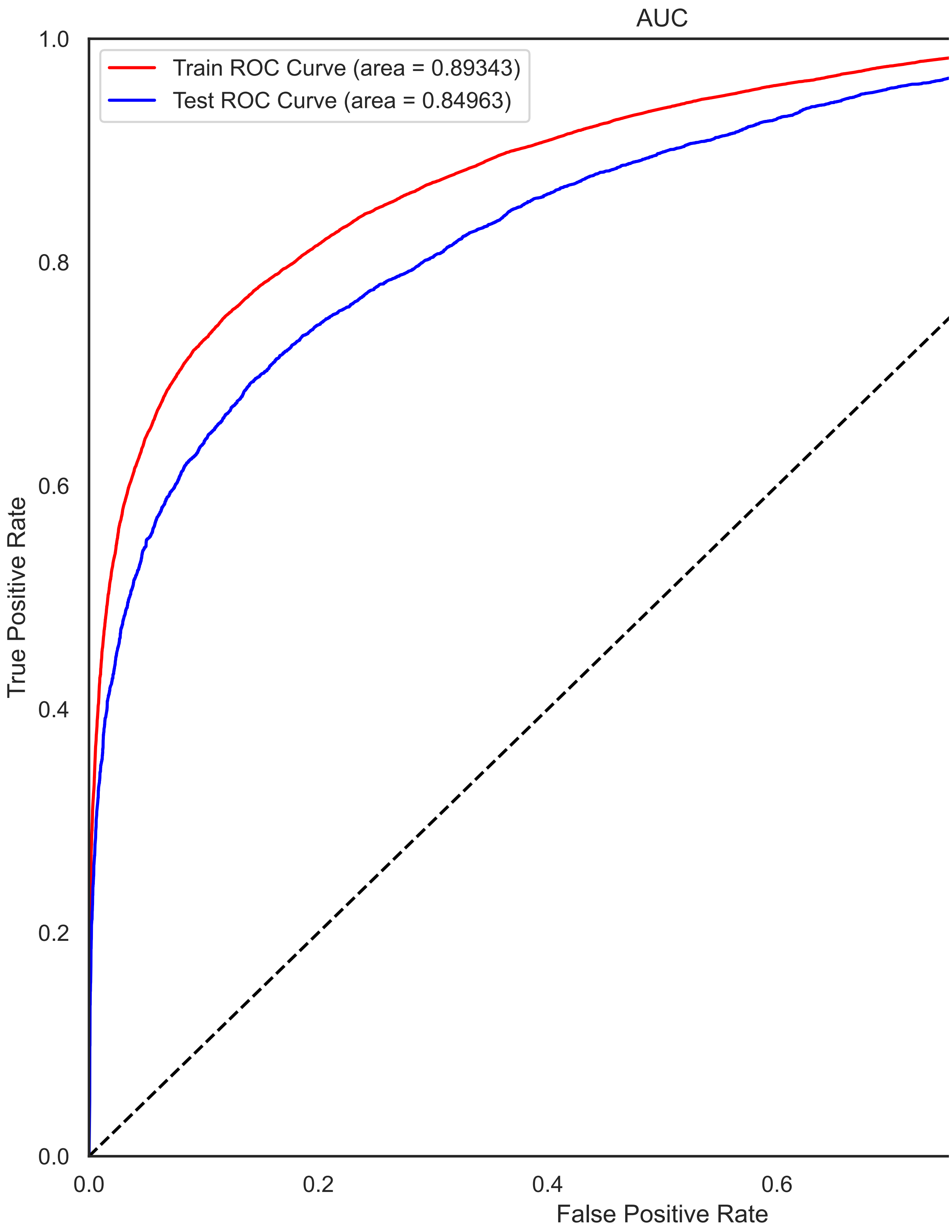
```python
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbdt = GradientBoostingClassifier(random_state=0,
n_estimators=10, max_depth=10)
gbdt = gbdt.fit(x_train, y_train)
y_pred_test = gbdt.predict_proba(x_test)[:, 1]
y_pred_train = gbdt.predict_proba(x_train)[:, 1]
important_stats(y_train, y_pred_train, "train result summary: ")
important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.7515727307741688
f1_score: 0.8036655054656531
accuracy_score: 0.816350444637067
AUC: 0.8934285595601786
Predicted      0      1    All
True
0          11213   8701  19914
1          11324   8573  19897
All        22537  17274  39811
------------------------------------------
------------------------------------------
test result summary:
recall: 0.7122066208875679
f1_score: 0.7600765654908396
accuracy_score: 0.7746674200010273
AUC: 0.8496263887085502
Predicted     0     1   All
True
0          1109   886  1995
1          1124   891  2015
All        2233  1777  4010
------------------------------------------
```

[66]

```
plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```



## LightGBM

```
[107]   import lightgbm as gbm
        from lightgbm import LGBMClassifier
```

```
[108]  gbm_clf = gbm.LGBMClassifier(
           boosting_type = 'gbdt',
           #num_leaves = ,
           #max_depth = ,
           learning_rate = 0.1
           #n_estimators =
           #,subsample_for_bin =
           ,objective = 'binary'
           ,metric = 'binary_logloss'
           #,class_weight =
           #,min_split_gain =
           #,min_split_weight =
           #,min_child_weight =
           #,min_child_samples =
           #,subsample =
           #,subsample_freq =
           #,colsample_bytree =
           ,reg_alpha = 5
           ,reg_lambda = 120
           ,importance_type = 'split' #will rank features by # of times
       it is used in model.'gain' for gain
           ,num_iterations = 1000
       )
```

```
[109]  gbm_clf.fit(
           x_train,
           y_train,
           eval_metric = 'result',
           verbose = True,
           eval_set = [(x_val, y_val)],
           early_stopping_rounds = 20
       )
```

```
[1]	valid_0's binary_logloss: 0.662348
Training until validation scores don't improve for 20 rounds
[2]	valid_0's binary_logloss: 0.637353
[3]	valid_0's binary_logloss: 0.615892
[4]	valid_0's binary_logloss: 0.597688
[5]	valid_0's binary_logloss: 0.582505
[6]	valid_0's binary_logloss: 0.569258
[7]	valid_0's binary_logloss: 0.55816
[8]	valid_0's binary_logloss: 0.548532
[9]	valid_0's binary_logloss: 0.539825
[10]	valid_0's binary_logloss: 0.532286
[11]	valid_0's binary_logloss: 0.52577
[12]	valid_0's binary_logloss: 0.519981
[13]	valid_0's binary_logloss: 0.515134
[14]	valid_0's binary_logloss: 0.510994
[15]	valid_0's binary_logloss: 0.507042
```

```
[16]    valid_0's binary_logloss: 0.50359
[17]    valid_0's binary_logloss: 0.500632
[18]    valid_0's binary_logloss: 0.497876
[19]    valid_0's binary_logloss: 0.49563
[20]    valid_0's binary_logloss: 0.493652
[21]    valid_0's binary_logloss: 0.491747
[22]    valid_0's binary_logloss: 0.4902
[23]    valid_0's binary_logloss: 0.488692
[24]    valid_0's binary_logloss: 0.487449
[25]    valid_0's binary_logloss: 0.486344
[26]    valid_0's binary_logloss: 0.485169
[27]    valid_0's binary_logloss: 0.484033
[28]    valid_0's binary_logloss: 0.483117
[29]    valid_0's binary_logloss: 0.482207
[30]    valid_0's binary_logloss: 0.481406
[31]    valid_0's binary_logloss: 0.480741
[32]    valid_0's binary_logloss: 0.480101
[33]    valid_0's binary_logloss: 0.479612
[34]    valid_0's binary_logloss: 0.479081
[35]    valid_0's binary_logloss: 0.478507
[36]    valid_0's binary_logloss: 0.47803
[37]    valid_0's binary_logloss: 0.477629
[38]    valid_0's binary_logloss: 0.477236
[39]    valid_0's binary_logloss: 0.476853
[40]    valid_0's binary_logloss: 0.476455
[41]    valid_0's binary_logloss: 0.476112
[42]    valid_0's binary_logloss: 0.47588
[43]    valid_0's binary_logloss: 0.475598
[44]    valid_0's binary_logloss: 0.475332
[45]    valid_0's binary_logloss: 0.475145
[46]    valid_0's binary_logloss: 0.474895
[47]    valid_0's binary_logloss: 0.474689
[48]    valid_0's binary_logloss: 0.474463
[49]    valid_0's binary_logloss: 0.474345
[50]    valid_0's binary_logloss: 0.474202
[51]    valid_0's binary_logloss: 0.474086
[52]    valid_0's binary_logloss: 0.4739
[53]    valid_0's binary_logloss: 0.473702
[54]    valid_0's binary_logloss: 0.473563
[55]    valid_0's binary_logloss: 0.473427
[56]    valid_0's binary_logloss: 0.473301
[57]    valid_0's binary_logloss: 0.473238
[58]    valid_0's binary_logloss: 0.473103
[59]    valid_0's binary_logloss: 0.47303
[60]    valid_0's binary_logloss: 0.472866
[61]    valid_0's binary_logloss: 0.472765
[62]    valid_0's binary_logloss: 0.472737
[63]    valid_0's binary_logloss: 0.472584
[64]    valid_0's binary_logloss: 0.472511
[65]    valid_0's binary_logloss: 0.472458
[66]    valid_0's binary_logloss: 0.472435
[67]    valid_0's binary_logloss: 0.472354
```

```
[68]    valid_0's binary_logloss: 0.472355
[69]    valid_0's binary_logloss: 0.472297
[70]    valid_0's binary_logloss: 0.472185
[71]    valid_0's binary_logloss: 0.472137
[72]    valid_0's binary_logloss: 0.472095
[73]    valid_0's binary_logloss: 0.472048
[74]    valid_0's binary_logloss: 0.472025
[75]    valid_0's binary_logloss: 0.471999
[76]    valid_0's binary_logloss: 0.471925
[77]    valid_0's binary_logloss: 0.471845
[78]    valid_0's binary_logloss: 0.471815
[79]    valid_0's binary_logloss: 0.471743
[80]    valid_0's binary_logloss: 0.471733
[81]    valid_0's binary_logloss: 0.471701
[82]    valid_0's binary_logloss: 0.471631
[83]    valid_0's binary_logloss: 0.471544
[84]    valid_0's binary_logloss: 0.471519
[85]    valid_0's binary_logloss: 0.471492
[86]    valid_0's binary_logloss: 0.471465
[87]    valid_0's binary_logloss: 0.471421
[88]    valid_0's binary_logloss: 0.47136
[89]    valid_0's binary_logloss: 0.471333
[90]    valid_0's binary_logloss: 0.471315
[91]    valid_0's binary_logloss: 0.471279
[92]    valid_0's binary_logloss: 0.471268
[93]    valid_0's binary_logloss: 0.471213
[94]    valid_0's binary_logloss: 0.471181
[95]    valid_0's binary_logloss: 0.471163
[96]    valid_0's binary_logloss: 0.471141
[97]    valid_0's binary_logloss: 0.47112
[98]    valid_0's binary_logloss: 0.471043
[99]    valid_0's binary_logloss: 0.471022
[100]   valid_0's binary_logloss: 0.470953
[101]   valid_0's binary_logloss: 0.470941
[102]   valid_0's binary_logloss: 0.470935
[103]   valid_0's binary_logloss: 0.470899
[104]   valid_0's binary_logloss: 0.470889
[105]   valid_0's binary_logloss: 0.470889
[106]   valid_0's binary_logloss: 0.470875
[107]   valid_0's binary_logloss: 0.470829
[108]   valid_0's binary_logloss: 0.470829
[109]   valid_0's binary_logloss: 0.470807
[110]   valid_0's binary_logloss: 0.470772
[111]   valid_0's binary_logloss: 0.470746
[112]   valid_0's binary_logloss: 0.470724
[113]   valid_0's binary_logloss: 0.470666
[114]   valid_0's binary_logloss: 0.470613
[115]   valid_0's binary_logloss: 0.470623
[116]   valid_0's binary_logloss: 0.470581
[117]   valid_0's binary_logloss: 0.470596
[118]   valid_0's binary_logloss: 0.470589
[119]   valid_0's binary_logloss: 0.470559
```

```
[120]	valid_0's binary_logloss: 0.470511
[121]	valid_0's binary_logloss: 0.470501
[122]	valid_0's binary_logloss: 0.470474
[123]	valid_0's binary_logloss: 0.470504
[124]	valid_0's binary_logloss: 0.470503
[125]	valid_0's binary_logloss: 0.470496
[126]	valid_0's binary_logloss: 0.470461
[127]	valid_0's binary_logloss: 0.470453
[128]	valid_0's binary_logloss: 0.470476
[129]	valid_0's binary_logloss: 0.470482
[130]	valid_0's binary_logloss: 0.470472
[131]	valid_0's binary_logloss: 0.470463
[132]	valid_0's binary_logloss: 0.470432
[133]	valid_0's binary_logloss: 0.470437
[134]	valid_0's binary_logloss: 0.470412
[135]	valid_0's binary_logloss: 0.470408
[136]	valid_0's binary_logloss: 0.470363
[137]	valid_0's binary_logloss: 0.470332
[138]	valid_0's binary_logloss: 0.470325
[139]	valid_0's binary_logloss: 0.470301
[140]	valid_0's binary_logloss: 0.47029
[141]	valid_0's binary_logloss: 0.470277
[142]	valid_0's binary_logloss: 0.470267
[143]	valid_0's binary_logloss: 0.470269
[144]	valid_0's binary_logloss: 0.470284
[145]	valid_0's binary_logloss: 0.47028
[146]	valid_0's binary_logloss: 0.470289
[147]	valid_0's binary_logloss: 0.470284
[148]	valid_0's binary_logloss: 0.47028
[149]	valid_0's binary_logloss: 0.470262
[150]	valid_0's binary_logloss: 0.470252
[151]	valid_0's binary_logloss: 0.470263
[152]	valid_0's binary_logloss: 0.470245
[153]	valid_0's binary_logloss: 0.470234
[154]	valid_0's binary_logloss: 0.470231
[155]	valid_0's binary_logloss: 0.470219
[156]	valid_0's binary_logloss: 0.470187
[157]	valid_0's binary_logloss: 0.470198
[158]	valid_0's binary_logloss: 0.470208
[159]	valid_0's binary_logloss: 0.470194
[160]	valid_0's binary_logloss: 0.470202
[161]	valid_0's binary_logloss: 0.470212
[162]	valid_0's binary_logloss: 0.470182
[163]	valid_0's binary_logloss: 0.470189
[164]	valid_0's binary_logloss: 0.470199
[165]	valid_0's binary_logloss: 0.470197
[166]	valid_0's binary_logloss: 0.470195
[167]	valid_0's binary_logloss: 0.470171
[168]	valid_0's binary_logloss: 0.470141
[169]	valid_0's binary_logloss: 0.470143
[170]	valid_0's binary_logloss: 0.470142
[171]	valid_0's binary_logloss: 0.47016
```

```
[172]	valid_0's binary_logloss: 0.470152
[173]	valid_0's binary_logloss: 0.470118
[174]	valid_0's binary_logloss: 0.470107
[175]	valid_0's binary_logloss: 0.47011
[176]	valid_0's binary_logloss: 0.470103
[177]	valid_0's binary_logloss: 0.470108
[178]	valid_0's binary_logloss: 0.470115
[179]	valid_0's binary_logloss: 0.470092
[180]	valid_0's binary_logloss: 0.470081
[181]	valid_0's binary_logloss: 0.470085
[182]	valid_0's binary_logloss: 0.470089
[183]	valid_0's binary_logloss: 0.470088
[184]	valid_0's binary_logloss: 0.470071
[185]	valid_0's binary_logloss: 0.470057
[186]	valid_0's binary_logloss: 0.470062
[187]	valid_0's binary_logloss: 0.470058
[188]	valid_0's binary_logloss: 0.470057
[189]	valid_0's binary_logloss: 0.470063
[190]	valid_0's binary_logloss: 0.470061
[191]	valid_0's binary_logloss: 0.470069
[192]	valid_0's binary_logloss: 0.470061
[193]	valid_0's binary_logloss: 0.470053
[194]	valid_0's binary_logloss: 0.470045
[195]	valid_0's binary_logloss: 0.47005
[196]	valid_0's binary_logloss: 0.470052
[197]	valid_0's binary_logloss: 0.470063
[198]	valid_0's binary_logloss: 0.470061
[199]	valid_0's binary_logloss: 0.470056
[200]	valid_0's binary_logloss: 0.470027
[201]	valid_0's binary_logloss: 0.47003
[202]	valid_0's binary_logloss: 0.470007
[203]	valid_0's binary_logloss: 0.469976
[204]	valid_0's binary_logloss: 0.469943
[205]	valid_0's binary_logloss: 0.469951
[206]	valid_0's binary_logloss: 0.469941
[207]	valid_0's binary_logloss: 0.46995
[208]	valid_0's binary_logloss: 0.469962
[209]	valid_0's binary_logloss: 0.469961
[210]	valid_0's binary_logloss: 0.469949
[211]	valid_0's binary_logloss: 0.469952
[212]	valid_0's binary_logloss: 0.46996
[213]	valid_0's binary_logloss: 0.469955
[214]	valid_0's binary_logloss: 0.469957
[215]	valid_0's binary_logloss: 0.469945
[216]	valid_0's binary_logloss: 0.469948
[217]	valid_0's binary_logloss: 0.469945
[218]	valid_0's binary_logloss: 0.469943
[219]	valid_0's binary_logloss: 0.469931
[220]	valid_0's binary_logloss: 0.469931
[221]	valid_0's binary_logloss: 0.469918
[222]	valid_0's binary_logloss: 0.469912
[223]	valid_0's binary_logloss: 0.469909
```

```
[224]	valid_0's binary_logloss: 0.469879
[225]	valid_0's binary_logloss: 0.469873
[226]	valid_0's binary_logloss: 0.469876
[227]	valid_0's binary_logloss: 0.469894
[228]	valid_0's binary_logloss: 0.469905
[229]	valid_0's binary_logloss: 0.469903
[230]	valid_0's binary_logloss: 0.469893
[231]	valid_0's binary_logloss: 0.469912
[232]	valid_0's binary_logloss: 0.469902
[233]	valid_0's binary_logloss: 0.469902
[234]	valid_0's binary_logloss: 0.469904
[235]	valid_0's binary_logloss: 0.469896
[236]	valid_0's binary_logloss: 0.469901
[237]	valid_0's binary_logloss: 0.469907
[238]	valid_0's binary_logloss: 0.469931
[239]	valid_0's binary_logloss: 0.46992
[240]	valid_0's binary_logloss: 0.469916
[241]	valid_0's binary_logloss: 0.46992
[242]	valid_0's binary_logloss: 0.46992
[243]	valid_0's binary_logloss: 0.46991
[244]	valid_0's binary_logloss: 0.469895
[245]	valid_0's binary_logloss: 0.469879
Early stopping, best iteration is:
[225]	valid_0's binary_logloss: 0.469873
```

```
LGBMClassifier(boosting_type='gbdt', class_weight=None,
colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               metric='binary_logloss', min_child_samples=20,
               min_child_weight=0.001, min_split_gain=0.0,
n_estimators=100,
               n_jobs=-1, num_iterations=1000, num_leaves=31,
               objective='binary', random_state=None, reg_alpha=5,
               reg_lambda=120, silent=True, subsample=1.0,
               subsample_for_bin=200000, subsample_freq=0)
```

[54]

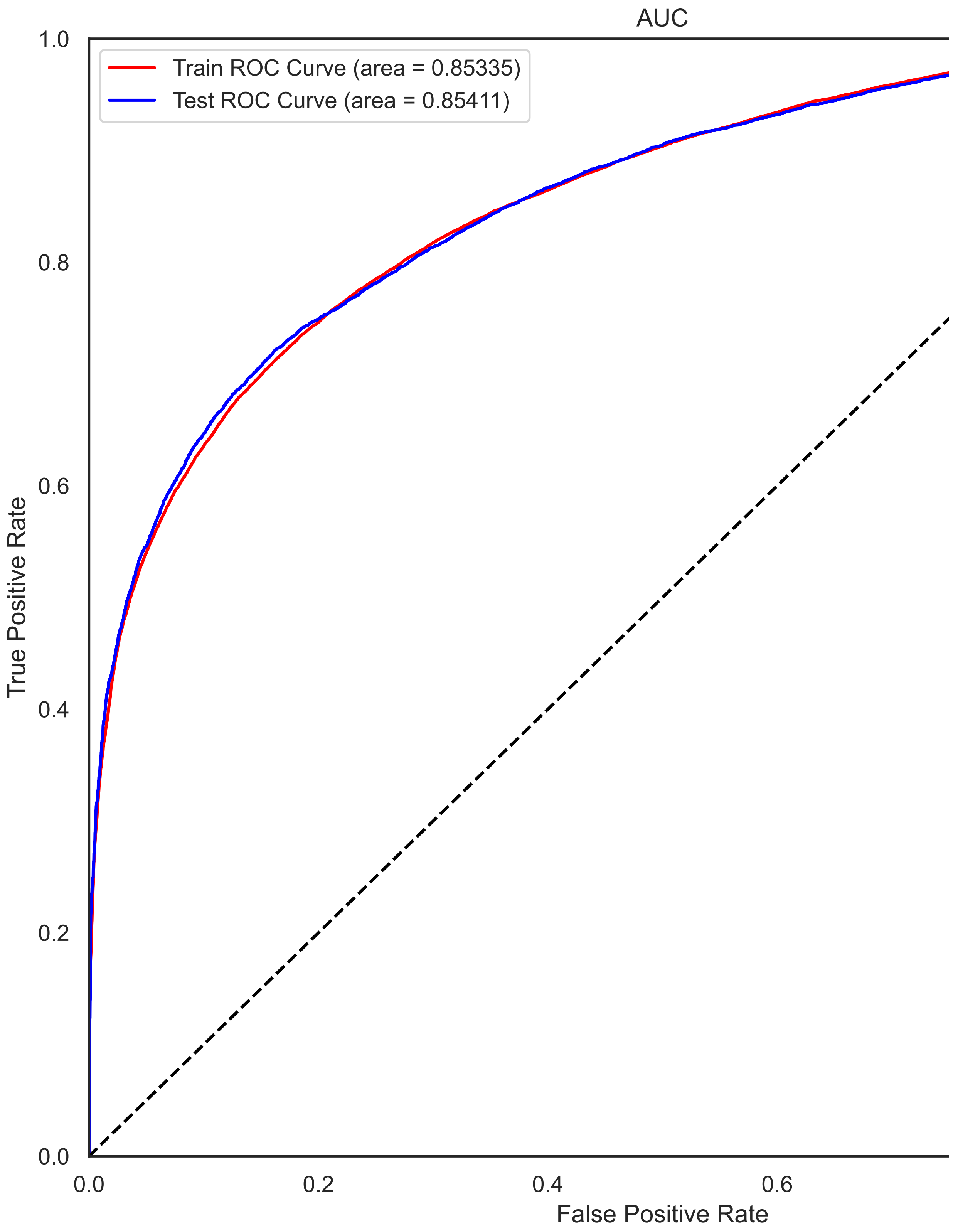
```
y_pred_test = gbm_clf.predict_proba(x_test)[:, 1]
y_pred_train = gbm_clf.predict_proba(x_train)[:, 1]
important_stats(y_train, y_pred_train, "train result summary: ")
important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.7247920665387076
f1_score: 0.770344933104398
accuracy_score: 0.7831516766457454
AUC: 0.8630115360217185
Predicted      0      1    All
True
0          11015   8740  19755
1          11121   8919  20040
```

```
All        22136  17659  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.7157407407407408
f1_score: 0.7611597374179432
accuracy_score: 0.7757345387302239
AUC: 0.853773792999218
Predicted     0     1   All
True
0          1091   882  1973
1          1135   902  2037
All        2226  1784  4010
------------------------------------------
```

[110]

```
plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```



## GridSearch for GBDT

[111]

```
from sklearn.model_selection import GridSearchCV
param_test1 = {'n_estimators':range(20,81,10)}
```

```
gsearch1 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1,
min_samples_split=300,

min_samples_leaf=20,max_depth=8,max_features='sqrt',

subsample=0.8,random_state=10),
                         param_grid = param_test1,
scoring='roc_auc',iid=False,cv=5)
gsearch1.fit(df.drop(columns =
['result','duration']),df['result'])
gsearch1.cv_results_, gsearch1.best_params_, gsearch1.best_score_
```

```
({'mean_fit_time': array([1.32538543, 1.89380035, 2.47538829,
2.98179235, 3.5398037 ,
         4.12919822, 4.63930659]),
  'std_fit_time': array([0.03889608, 0.02579396, 0.02366507, 0.06919284,
0.10178732,
         0.06592678, 0.0537543 ]),
  'mean_score_time': array([0.04561749, 0.04720511, 0.05919404,
0.06441445, 0.07180109,
         0.08361435, 0.08692303]),
  'std_score_time': array([0.01460292, 0.00039702, 0.00487201,
0.00100702, 0.00132823,
         0.00927824, 0.00326713]),
  'param_n_estimators': masked_array(data=[20, 30, 40, 50, 60, 70, 80],
               mask=[False, False, False, False, False, False, False],
         fill_value='?',
              dtype=object),
  'params': [{'n_estimators': 20},
   {'n_estimators': 30},
   {'n_estimators': 40},
   {'n_estimators': 50},
   {'n_estimators': 60},
   {'n_estimators': 70},
   {'n_estimators': 80}],
  'split0_test_score': array([0.84731079, 0.84979652, 0.85103372,
0.85197477, 0.85263569,
         0.85299489, 0.8531087 ]),
  'split1_test_score': array([0.85201503, 0.85391403, 0.85478833,
0.85529403, 0.85543664,
         0.85559767, 0.85558883]),
  'split2_test_score': array([0.84818393, 0.84985972, 0.85031364,
0.85085944, 0.85107263,
         0.85120242, 0.85141507]),
  'split3_test_score': array([0.8520118 , 0.85381196, 0.85483564,
0.85561528, 0.8556993 ,
         0.85586223, 0.85588279]),
  'split4_test_score': array([0.84593804, 0.84839916, 0.84974214,
0.85067136, 0.85102791,
         0.85125823, 0.8514505 ]),
  'mean_test_score': array([0.84909192, 0.85115628, 0.8521427 ,
0.85288298, 0.85317444,
         0.85338309, 0.85348918]),
  'std_test_score': array([0.00249055, 0.0022711 , 0.00221762,
0.00214893, 0.00204   ,
         0.00202344, 0.00193602]),
  'rank_test_score': array([7, 6, 5, 4, 3, 2, 1])},
 {'n_estimators': 80},
 0.8534891785733354)
```

[112]

```
param_test1 = {'n_estimators':range(80,151,10)}
gsearch1 = GridSearchCV(estimator =
GradientBoostingClassifier(learning_rate=0.1,
min_samples_split=300,

min_samples_leaf=20,max_depth=8,max_features='sqrt',

subsample=0.8,random_state=10),
                         param_grid = param_test1,
scoring='roc_auc',iid=False,cv=5)
gsearch1.fit(df.drop(columns =
['result','duration']),df['result'])
gsearch1.cv_results_, gsearch1.best_params_, gsearch1.best_score_
```

```
({'mean_fit_time': array([4.64361835, 5.13062172, 5.66787639,
6.19273343, 6.72101607,
         7.27307591, 7.79701047, 8.74241862]),
  'std_fit_time': array([0.03414646, 0.11800037, 0.0731966 , 0.12261328,
0.17383439,
         0.16389111, 0.07250512, 0.3989696 ]),
  'mean_score_time': array([0.08697863, 0.09180603, 0.09779897,
0.10341878, 0.11660056,
         0.11560607, 0.12040043, 0.12919893]),
  'std_score_time': array([0.00109128, 0.00271011, 0.00074137,
0.00049959, 0.00847716,
         0.00233217, 0.00100669, 0.00213085]),
  'param_n_estimators': masked_array(data=[80, 90, 100, 110, 120, 130,
140, 150],
               mask=[False, False, False, False, False, False, False,
False],
         fill_value='?',
              dtype=object),
  'params': [{'n_estimators': 80},
   {'n_estimators': 90},
   {'n_estimators': 100},
   {'n_estimators': 110},
   {'n_estimators': 120},
   {'n_estimators': 130},
   {'n_estimators': 140},
   {'n_estimators': 150}],
  'split0_test_score': array([0.8531087 , 0.8530571 , 0.85316144,
0.85306623, 0.85301808,
         0.85309912, 0.85309443, 0.85307371]),
  'split1_test_score': array([0.85558883, 0.85564495, 0.85574738,
0.85565897, 0.85548331,
         0.85535441, 0.85535005, 0.85542428]),
  'split2_test_score': array([0.85141507, 0.85155896, 0.85140605,
0.85140464, 0.85153647,
         0.85153528, 0.85153919, 0.85154953]),
  'split3_test_score': array([0.85588279, 0.85591718, 0.85597176,
0.8558617 , 0.85586745,
         0.85575223, 0.85561992, 0.85554805]),
  'split4_test_score': array([0.8514505 , 0.85148205, 0.85147098,
0.85149088, 0.8516372 ,
         0.85152291, 0.85151866, 0.85149613]),
  'mean_test_score': array([0.85348918, 0.85353205, 0.85355153,
0.85349648, 0.8535085 ,
         0.85345279, 0.85342445, 0.85341834]),
  'std_test_score': array([0.00193602, 0.0019222 , 0.00198813,
0.00194185, 0.00184909,
         0.00181273, 0.00177894, 0.00178132]),
  'rank_test_score': array([5, 2, 1, 4, 3, 6, 7, 8])},
 {'n_estimators': 100},
 0.8535515251856326)
```

```
[114]  param_test2 = {'max_depth':range(3,14,2),
       'min_samples_split':range(100,801,200)}
       gsearch2 = GridSearchCV(
           estimator = GradientBoostingClassifier(
               learning_rate=0.1,
               n_estimators=120,
               min_samples_leaf=20,
               max_features='sqrt',
               subsample=0.8,
               random_state=10
           ),
           param_grid = param_test2,
           scoring='roc_auc',
           iid=False,
           cv=5)
       gsearch2.fit(df.drop(columns =
       ['result','duration']),df['result'])
       gsearch2.cv_results_, gsearch2.best_params_, gsearch2.best_score_
```

```
({'mean_fit_time': array([ 3.45556402,  3.61038938,  3.72701602,
3.67620912,  5.08439488,
          5.28479586,  5.2830235 ,  4.97647386,  6.7170043 ,
6.26970787,
          5.97208714,  5.83681006,  8.37736716,  7.60057206,
7.70422783,
          7.66988192, 10.5684083 ,  9.56854734,  8.91962228,  8.798598
,
         12.91343093, 12.09202356, 10.35147781,  9.29873238]),
  'std_fit_time': array([0.1250714 , 0.05000499, 0.08158398, 0.12801082,
0.08381246,
         0.25093659, 0.13585953, 0.17017597, 0.38197038, 0.16975322,
         0.12140917, 0.13991024, 0.43689841, 0.26756902, 0.451388  ,
         0.34377249, 0.16163023, 0.13505017, 0.10145558, 0.16877364,
         0.27428557, 0.96775998, 1.02669756, 0.15228346]),
  'mean_score_time': array([0.05162759, 0.06540051, 0.05380282,
0.05259786, 0.08100562,
         0.07601733, 0.08438973, 0.07564855, 0.11180573, 0.10120153,
         0.09640412, 0.09418507, 0.13622217, 0.12282095, 0.131393  ,
         0.12900147, 0.16540813, 0.16740503, 0.15238347, 0.15100279,
         0.19861937, 0.19217443, 0.18200316, 0.15680385]),
  'std_score_time': array([0.00301823, 0.02093085, 0.00712394,
0.00632493, 0.00868302,
         0.00352076, 0.01046627, 0.00427177, 0.01122829, 0.00685354,
         0.00233601, 0.00040692, 0.01814766, 0.00160372, 0.01127262,
         0.01121384, 0.01241693, 0.01517542, 0.0059765 , 0.00828008,
         0.01509932, 0.02085057, 0.02308627, 0.0021403 ]),
  'param_max_depth': masked_array(data=[3, 3, 3, 3, 5, 5, 5, 5, 7, 7, 7,
7, 9, 9, 9, 9, 11, 11,
                     11, 11, 13, 13, 13, 13],
               mask=[False, False, False, False, False, False, False,
False,
                     False, False, False, False, False, False, False,
False,
                     False, False, False, False, False, False, False,
False],
         fill_value='?',
              dtype=object),
  'param_min_samples_split': masked_array(data=[100, 300, 500, 700, 100,
300, 500, 700, 100, 300, 500,
                     700, 100, 300, 500, 700, 100, 300, 500, 700, 100,
300,
                     500, 700],
               mask=[False, False, False, False, False, False, False,
False,
                     False, False, False, False, False, False, False,
False,
                     False, False, False, False, False, False, False,
False],
         fill_value='?',
              dtype=object),
  'params': [{'max_depth': 3, 'min_samples_split': 100},
   {'max_depth': 3, 'min_samples_split': 300},
   {'max_depth': 3, 'min_samples_split': 500},
   {'max_depth': 3, 'min_samples_split': 700},
   {'max_depth': 5, 'min_samples_split': 100},
   {'max_depth': 5, 'min_samples_split': 300},
   {'max_depth': 5, 'min_samples_split': 500},
   {'max_depth': 5, 'min_samples_split': 700},
   {'max_depth': 7, 'min_samples_split': 100},
   {'max_depth': 7, 'min_samples_split': 300},
   {'max_depth': 7, 'min_samples_split': 500},
   {'max_depth': 7, 'min_samples_split': 700},
   {'max_depth': 9, 'min_samples_split': 100},
   {'max_depth': 9, 'min_samples_split': 300},
   {'max_depth': 9, 'min_samples_split': 500},
   {'max_depth': 9, 'min_samples_split': 700},
   {'max_depth': 11, 'min_samples_split': 100},
   {'max_depth': 11, 'min_samples_split': 300},
   {'max_depth': 11, 'min_samples_split': 500},
   {'max_depth': 11, 'min_samples_split': 700},
   {'max_depth': 13, 'min_samples_split': 100},
   {'max_depth': 13, 'min_samples_split': 300},
   {'max_depth': 13, 'min_samples_split': 500},
   {'max_depth': 13, 'min_samples_split': 700}],
  'split0_test_score': array([0.84799678, 0.84807903, 0.84810839,
0.84805585, 0.85254784,
         0.85206581, 0.85197321, 0.85224872, 0.85242121, 0.85358496,
         0.85297244, 0.85319115, 0.85197848, 0.85273365, 0.85315618,
         0.85312648, 0.84960534, 0.85186351, 0.85160557, 0.85241442,
         0.84834351, 0.85062906, 0.85112665, 0.85207774]),
  'split1_test_score': array([0.85251421, 0.85295653, 0.85256148,
0.85272752, 0.85486332,
         0.85520937, 0.85494075, 0.85493413, 0.8559036 , 0.85593558,
         0.85578692, 0.85605588, 0.85459506, 0.85525841, 0.85522146,
         0.85559023, 0.85328955, 0.85378909, 0.85512607, 0.85520454,
         0.85138227, 0.85367802, 0.85426616, 0.85504019]),
  'split2_test_score': array([0.84742484, 0.84802475, 0.84804271,
0.84769015, 0.85133416,
         0.85150392, 0.85129539, 0.85110923, 0.85137496, 0.85176178,
         0.8515078 , 0.85209579, 0.85083211, 0.85141843, 0.85187714,
         0.85186189, 0.84955372, 0.85036691, 0.85102396, 0.85224837,
         0.84757987, 0.84947853, 0.85057135, 0.85122504]),
  'split3_test_score': array([0.85118431, 0.85156696, 0.85157943,
0.85167321, 0.85567397,
         0.85527397, 0.855096  , 0.85527526, 0.855685  , 0.85551696,
         0.85575569, 0.85565237, 0.85494414, 0.85536433, 0.85599175,
         0.85594658, 0.8528644 , 0.85497016, 0.85605099, 0.85545646,
         0.85045409, 0.85392792, 0.85460427, 0.85501261]),
  'split4_test_score': array([0.84557792, 0.84519506, 0.84560721,
0.84561366, 0.85006306,
         0.85052858, 0.85039992, 0.85019692, 0.85095416, 0.85171547,
         0.85110735, 0.85148942, 0.84951247, 0.85175528, 0.8513141 ,
         0.85206876, 0.84893426, 0.85021242, 0.85076372, 0.85114496,
         0.8469133 , 0.84990433, 0.85041393, 0.85108742]),
  'mean_test_score': array([0.84893961, 0.84916447, 0.84917984,
0.84915208, 0.85289647,
         0.85291633, 0.85274105, 0.85275285, 0.85326779, 0.85370295,
         0.85342604, 0.85369692, 0.85237245, 0.85330602, 0.85351213,
         0.85371879, 0.85084945, 0.85224042, 0.85291406, 0.85329375,
         0.84893461, 0.85152357, 0.85219647, 0.8528886 ]),
  'std_test_score': array([0.00254169, 0.00277074, 0.00254552,
0.00264572, 0.00210588,
         0.00196144, 0.00192587, 0.00203022, 0.00211862, 0.00178927,
         0.0020131 , 0.00184829, 0.00211002, 0.00169375, 0.00182765,
         0.00173129, 0.00183896, 0.00187695, 0.00222   , 0.00172115,
         0.00170706, 0.00189878, 0.0018463 , 0.0017782 ]),
  'rank_test_score': array([23, 21, 20, 22, 11,  9, 14, 13,  8,  2,  5,
3, 15,  6,  4,  1, 19,
         16, 10,  7, 24, 18, 17, 12])},
 {'max_depth': 9, 'min_samples_split': 700},
 0.8537187869015375)
```

[116]

```
param_test3 = {'min_samples_leaf':range(60,101,10)}
gsearch3 = GridSearchCV(
    estimator = GradientBoostingClassifier(
        learning_rate=0.1,
        n_estimators=120,
        max_depth=7,
        min_samples_split=700,
        max_features='sqrt',
        subsample=0.8,
        random_state=10
    ),
    param_grid = param_test3,
    scoring='roc_auc',
    iid=False,
    verbose=1,
    cv=5
)

gsearch3.fit(df.drop(columns =
['result','duration']),df['result'])
gsearch3.cv_results_, gsearch3.best_params_, gsearch3.best_score_
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done  25 out of  25 | elapsed:  2.7min finished

({'mean_fit_time': array([6.06449895, 6.34776134, 6.46800838,
6.06420708, 6.41620145]),
  'std_fit_time': array([0.1295597 , 0.08398248, 0.31314299, 0.35224719,
0.3227051 ]),
  'mean_score_time': array([0.10140619, 0.10560927, 0.11020746,
0.10400147, 0.10178876]),
  'std_score_time': array([0.01030432, 0.0075005 , 0.01550756,
0.00918721, 0.00205163]),
  'param_min_samples_leaf': masked_array(data=[60, 70, 80, 90, 100],
               mask=[False, False, False, False, False],
         fill_value='?',
              dtype=object),
  'params': [{'min_samples_leaf': 60},
   {'min_samples_leaf': 70},
   {'min_samples_leaf': 80},
   {'min_samples_leaf': 90},
   {'min_samples_leaf': 100}],
  'split0_test_score': array([0.85278824, 0.85324258, 0.85328524,
0.85319134, 0.85283255]),
  'split1_test_score': array([0.85619283, 0.85620537, 0.85633786,
0.85643664, 0.85641502]),
  'split2_test_score': array([0.85210429, 0.85168207, 0.85245235,
0.8522972 , 0.85226799]),
  'split3_test_score': array([0.85606564, 0.85636268, 0.85600638,
0.85561514, 0.85574515]),
  'split4_test_score': array([0.85183398, 0.85146712, 0.8512513 ,
0.85090273, 0.85107601]),
  'mean_test_score': array([0.853797  , 0.85379196, 0.85386663,
0.85368861, 0.85366734]),
  'std_test_score': array([0.00192992, 0.00212563, 0.00199317,
0.00205949, 0.00206093]),
  'rank_test_score': array([2, 3, 1, 4, 5])},
 {'min_samples_leaf': 80},
 0.8538666256737037)
```

```
[16]  gbdt_best = GradientBoostingClassifier(
          learning_rate=0.1,
          n_estimators=100,
          max_depth=9,
          min_samples_leaf =80,
          min_samples_split =700,
          max_features='sqrt',
          subsample=0.8,
          random_state=10
      )
      gbdt_best.fit(df.drop(columns =
      ['result','duration']),df['result'])
```

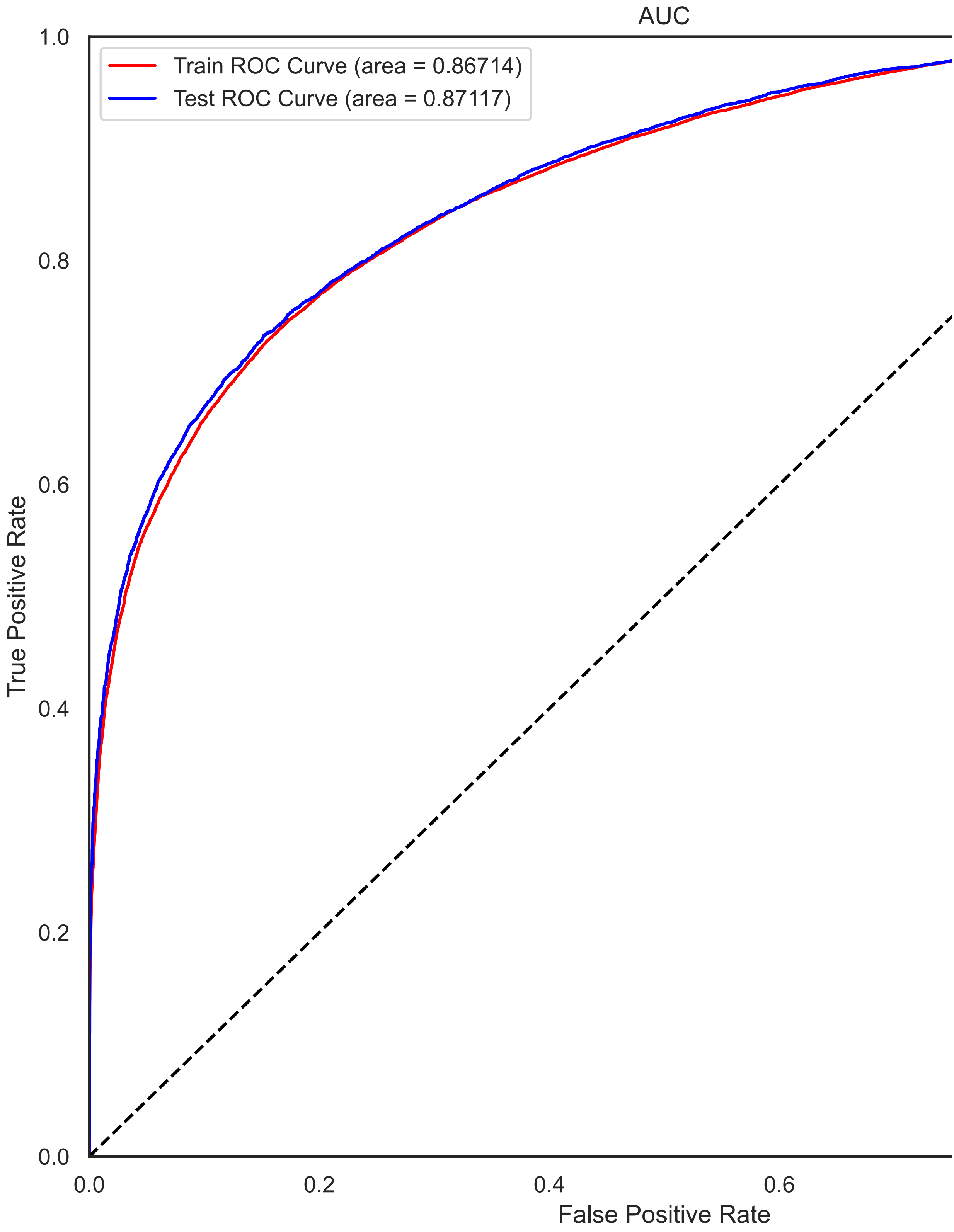
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
init=None,
                           learning_rate=0.1, loss='deviance',
max_depth=9,
                           max_features='sqrt', max_leaf_nodes=None,
                           min_impurity_decrease=0.0,
min_impurity_split=None,
                           min_samples_leaf=80, min_samples_split=700,
                           min_weight_fraction_leaf=0.0,
n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=10, subsample=0.8, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

```
[17]  y_pred_test = gbdt_best.predict_proba(x_test)[:, 1]
      y_pred_train = gbdt_best.predict_proba(x_train)[:, 1]
      important_stats(y_train, y_pred_train, "train result summary: ")
      important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.7176787777635126
f1_score: 0.7702642185400805
accuracy_score: 0.7859000288933834
AUC: 0.8671379869262116
Predicted      0      1    All
True
0          11315   8599  19914
1          11361   8536  19897
All        22676  17135  39811
------------------------------------------
------------------------------------------
test result summary:
recall: 0.7218407297324998
f1_score: 0.7745518530737929
accuracy_score: 0.7894088037392778
```

```
AUC: 0.871171321641422
Predicted     0     1   All
True
0          1128   867  1995
1          1127   888  2015
All        2255  1755  4010
------------------------------------------
```

```
[18]  plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```



## Features importance

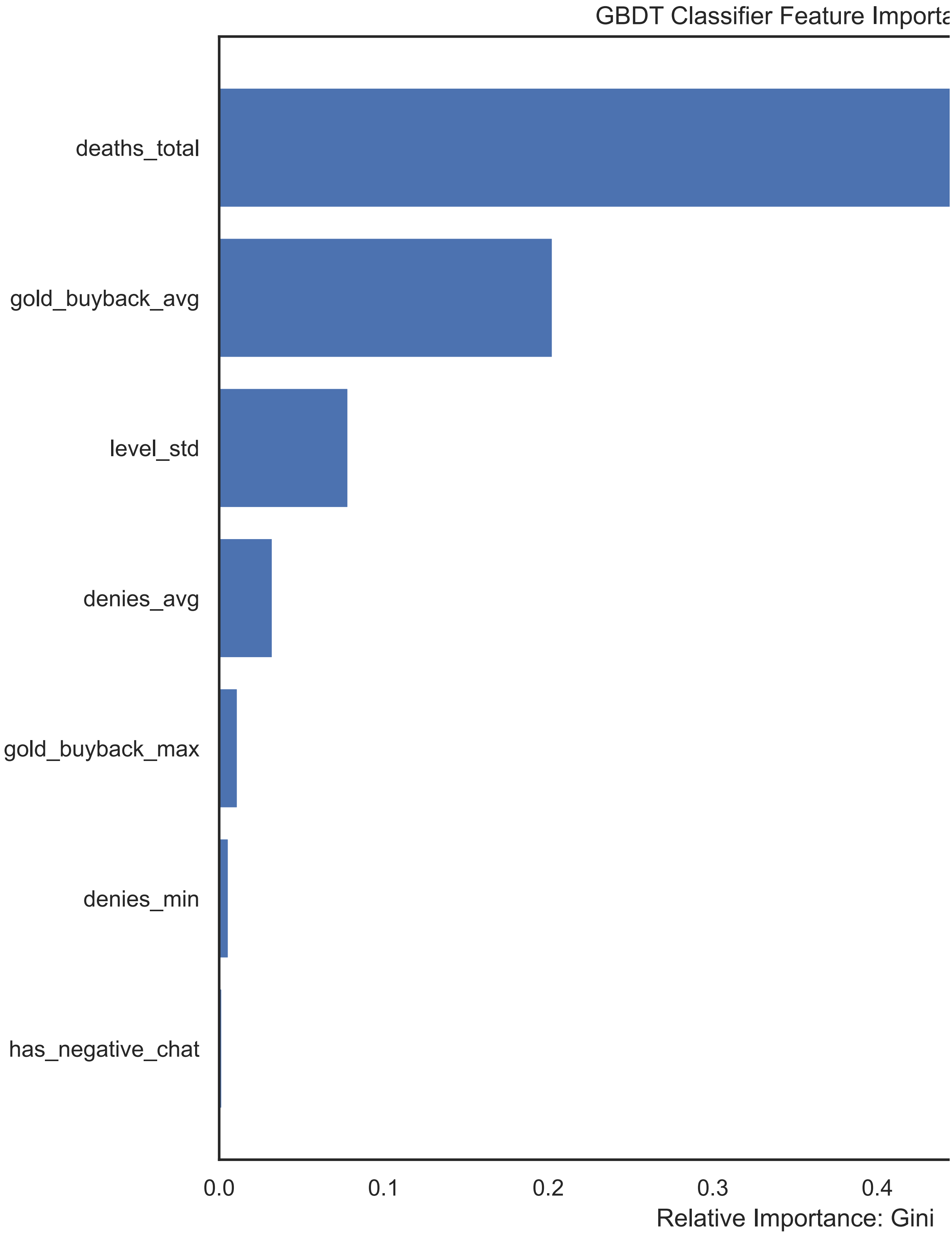
[19]

```
gbdt_importance = gbdt_best.feature_importances_
```

[122]

```
from matplotlib.pyplot import figure
figure(num=None, figsize = (10,10))
indices = np.argsort(gbdt_importance)
plt.figure(1)
plt.title('GBDT Classifier Feature Importance')
plt.barh(range(len(indices)), gbdt_importance[indices], color =
'b', align = 'center')
gbdt_feat_names = x_train.columns
plt.yticks(range(len(indices)), gbdt_feat_names[indices])
plt.xlabel('Relative Importance: Gini')
```

```
Text(0.5, 0, 'Relative Importance: Gini')
```



## Logistic Regression for predicting probabilities

[79]

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV(solver = 'saga',
                          penalty = 'elasticnet',
                          l1_ratios = [0.1, 0.2, 0.3],
                          Cs = 20,
                          n_jobs = -1,
                          random_state = 0,
                          class_weight = 0.9
)
lr.fit(x_train,y_train)
```

```
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\model_selection\_split.py:1978: FutureWarning: The
default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
```

```
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
LogisticRegressionCV(Cs=20, class_weight=0.9, cv='warn', dual=False,
                     fit_intercept=True, intercept_scaling=1.0,
                     l1_ratios=[0.1, 0.2, 0.3], max_iter=100,
                     multi_class='warn', n_jobs=-1,
penalty='elasticnet',
                     random_state=0, refit=True, scoring=None,
solver='saga',
                     tol=0.0001, verbose=0)
```

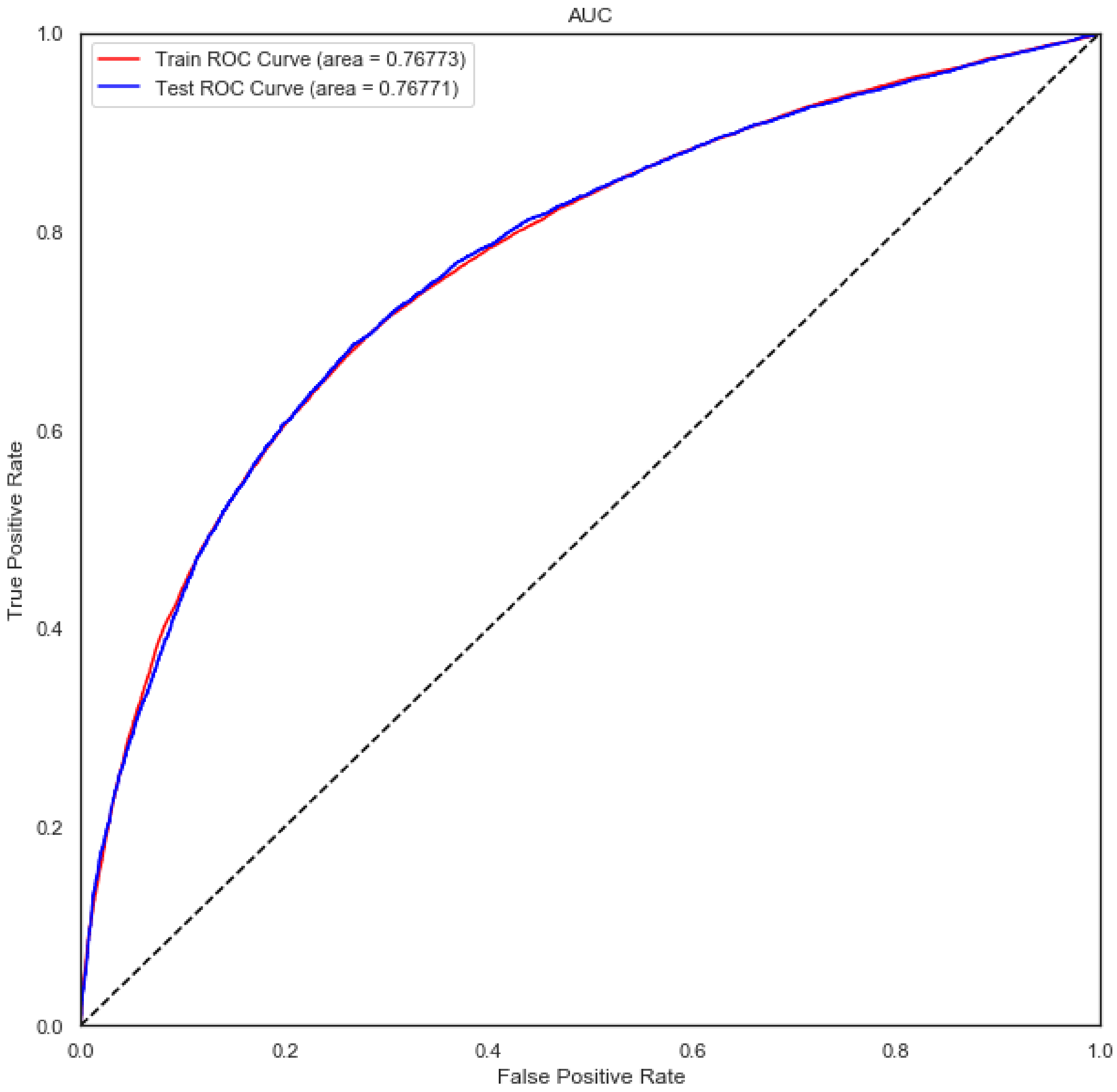
[80]

```
y_pred_test = lr.predict_proba(x_test)[:, 1]
y_pred_train = lr.predict_proba(x_train)[:, 1]
important_stats(y_train, y_pred_train, "train result summary: ")
important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.5398912348048625
f1_score: 0.6377824805381301
accuracy_score: 0.6922805271521904
AUC: 0.7677302786258187
Predicted      0      1    All
True
0          12899   6856  19755
1          13041   6999  20040
All        25940  13855  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.5417695473251029
f1_score: 0.6383030303030303
accuracy_score: 0.6934456544072324
AUC: 0.7677087607714403
Predicted     0     1   All
True
0          1280   693  1973
1          1292   745  2037
```

```
All        2572  1438  4010
------------------------------------------
```

```
[81]  plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```


AUC
Train ROC Curve (area = 0.76773)
Test ROC Curve (area = 0.76771)
True Positive Rate
False Positive Rate