[237]
```python
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import r2_score, recall_score, f1_score,
roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear_model import LogisticRegressionCV

import xgboost as xgb
from xgboost import XGBClassifier

import lightgbm as gbm
from lightgbm import LGBMClassifier

from matplotlib.pyplot import figure
import matplotlib.pyplot as plt
import seaborn as sns
```

Data aggregation

[238]
```python
match_file = pd.read_csv('dota-2-matches/match.csv')
match_file.head()
```

	match_id	start_time	duration	tower_status_radiant	tower_sta
0	0	1446750112	2375	1982	4
1	1	1446753078	2582	0	1846
2	2	1446764586	2716	256	1972
3	3	1446765723	3085	4	1924
4	4	1446796385	1887	2047	0

[239]
```python
# Util function for data aggregation
```

```
# details of bit string is in :
#
https://wiki.teamfortress.com/wiki/WebAPI/GetMatchDetails#Player_
Slot
def tower_status(ts_radiant, ts_dire):
    tsr = {}
    tsd = {}
    bit_tsr = '{0:016b}'.format(ts_radiant)
    bit_tsd = '{0:016b}'.format(ts_dire)
    tsr['top'] = bit_tsr.count('1', -3)
    tsd['top'] = bit_tsd.count('1', -3)
    tsr['mid'] = bit_tsr.count('1', 10, 13)
    tsd['mid'] = bit_tsd.count('1', 10, 13)
    tsd['bottom'] = bit_tsd.count('1', 7, 10)
    tsr['bottom'] = bit_tsr.count('1', 7, 10)
    tsd['ancient'] = bit_tsd.count('1', 5, 7)
    tsr['ancient'] = bit_tsr.count('1', 5, 7)
    return (tsr, tsd)

def barracks_status(bs_radiant, bs_dire):
    bsr = {}
    bsd = {}
    bit_bsr = '{0:08b}'.format(bs_radiant)
    bit_bsd = '{0:08b}'.format(bs_dire)
    bsr['top'] = bit_bsr.count('1', -2)
    bsd['top'] = bit_bsd.count('1', -2)
    bsr['mid'] = bit_bsr.count('1', 2, 4)
    bsd['mid'] = bit_bsd.count('1', 2, 4)
    bsd['bottom'] = bit_bsd.count('1', 4, 6)
    bsr['bottom'] = bit_bsr.count('1', 4, 6)
    return (bsr, bsd)
```

[240]

```
df_players = pd.read_csv(
    'dota-2-matches/players.csv',
    usecols=[
        'match_id',
        'player_slot',
        'gold',
        'gold_spent',
        'kills',
        'deaths',
        'assists',
        'denies',
        'last_hits',
        'hero_damage',
        'tower_damage',
        'level',
        'gold_buyback'
        ])
df_team_fights = pd.read_csv('dota-2-matches/teamfights.csv')
```

```
df_team_fights_players = pd.read_csv('dota-2-
matches/teamfights_players.csv')
```

Novel Features - negative chat

We tried a custom known list of reliably negative words in chat as a novel feature. We count the number of occurrences of each word in the dictionary in chat per team per match.

```
[241] df_chat = pd.read_csv('dota-2-matches/chat.csv')
df_chat['key'].fillna('', inplace=True)

naughty_words = [
    'stfu',
    'ez',
    'fuck',
    'wtf',
    'blame',
    'report',
    'reported',
    'shit',
    'ass',
    'asshole',
    'idiot',
    'stupid',
    'support',
    'blyat',
    'noob',
    'gg'
]

def get_naughty_count(phrase):
    naughty_count = 0
    tokens = phrase.split()
    for token in tokens:
        naughty_count = naughty_count + (1 if token in
naughty_words else 0)
    return naughty_count

df_chat['is_radiant'] = df_chat['slot'] < 5
df_chat['naughty_count'] =
df_chat['key'].apply(get_naughty_count)
df_chat.head()
```

	match_id	key	slot	time	unit	is_radiant	naughty_

	match_id	key	slot	time	unit	is_radiant	naughty_
0	0	force it	6	-8	6k Slayer	False	0
1	0	space created	1	5	Monkey	True	0
2	0	hah	1	6	Monkey	True	0
3	0	ez 500	6	9	6k Slayer	False	1
4	0	mvp ulti	4	934	Kira	True	0

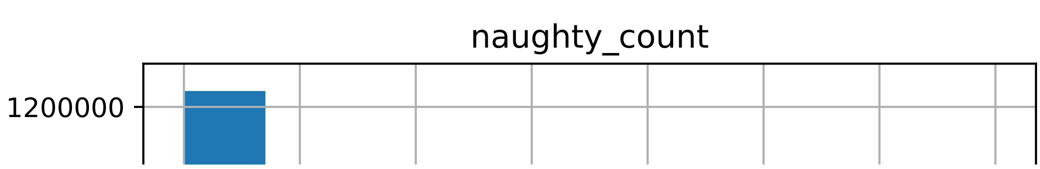
[242]
```
df_chat['naughty_count'].describe()
```

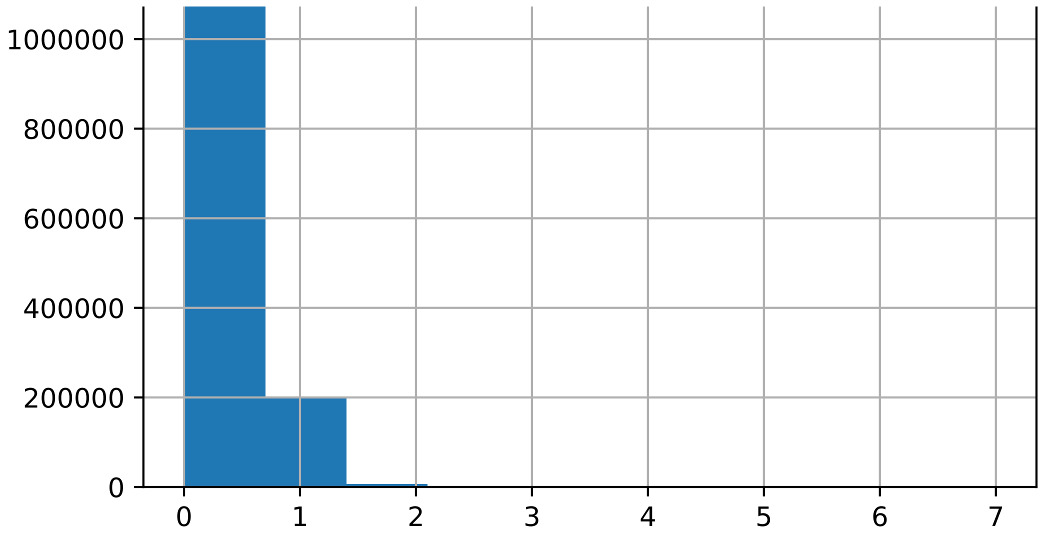
```
count    1.439488e+06
mean     1.467015e-01
std      3.670513e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      7.000000e+00
Name: naughty_count, dtype: float64
```

We can see that the median negative word count is 0, and the majority of games have 0 instances of negative words. Thus, later we convert it to a binary feature (present or not).

[243]
```
df_naughty_count_only = pd.DataFrame(df_chat['naughty_count'],
columns=['naughty_count'])
df_naughty_count_only.hist()
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x13d325a50>]],
      dtype=object)
```






```
[244]  df_match_grouped = df_chat.groupby(['match_id', 'is_radiant'],
       as_index=False).naughty_count.agg('sum')
       df_match_grouped['radiant_naughty_count'] =
       np.where(df_match_grouped['is_radiant'] == True,
       df_match_grouped['naughty_count'], 0)
       df_match_grouped['dire_naughty_count'] =
       np.where(df_match_grouped['is_radiant'] == False,
       df_match_grouped['naughty_count'], 0)
       df_match_grouped.head()
```

	match_id	is_radiant	naughty_count	radiant_naughty_count	di
0	0	False	2	0	2
1	0	True	3	3	0
2	1	False	1	0	1
3	1	True	0	0	0
4	2	False	2	0	2

```
[245]  df_match_naughty_counts =
       df_match_grouped.groupby(['match_id']).agg({
           'radiant_naughty_count': 'sum',
           'dire_naughty_count': 'sum'
       })

       df_match_naughty_counts.head()
```

	radiant_naughty_count	dire_naughty_count
match_id		
0	3	2

	radiant_naughty_count	dire_naughty_count
match_id		
1	0	1
2	1	2
3	1	2
4	1	3

```
[246]
match_data = []
match_file = match_file[match_file['game_mode'] == 22]
df_players.fillna(0, inplace=True)

radiant_pl = [0,1,2,3,4]
dire_pl = [128,129,130,131,132]
player_features = {
    'gold': 'full_total',
    'gold_spent': 'full_avg',
    'kills': 'only_total',
    'deaths': 'full_total',
    'assists': 'full_avg',
    'denies': 'full_avg',
    'last_hits': 'full_avg',
    'hero_damage': 'full_total',
    'tower_damage': 'full_total',
    'level': 'full_total',
    'gold_buyback': 'full_avg'
    }
```

Novel Feature - Teamfight result

```
[247]
def teamfight_result(teamfights):
    loss_d = 0
    loss_r = 0
    for i in list(range(0,int(len(tf)/10))):
        tf_df = teamfights[i*10:(i+1)*10]
        rd = sum(tf_df[tf_df.player_slot.isin(radiant_pl)]
['deaths'])
        dd = sum(tf_df[tf_df.player_slot.isin(dire_pl)]
['deaths'])
        if dd < rd:
            loss_r += 1
        elif rd < dd:
            loss_d += 1
    return (loss_r, loss_d)
```

```
[248] def stat_agg(types: str, feature_name: str, data_list: str,
team_data: dict):
          if types == "only_total":
              team_data[f'{feature_name}_total'] = sum(data_list)
          elif types == "full_total":
              team_data[f'{feature_name}_total'] = sum(data_list)
              team_data[f'{feature_name}_max'] = max(data_list)
              team_data[f'{feature_name}_min'] = min(data_list)
              team_data[f'{feature_name}_std'] =
round(np.std(data_list), 4)
          elif types == "full_avg":
              team_data[f'{feature_name}_avg'] = np.average(data_list)
              team_data[f'{feature_name}_max'] = max(data_list)
              team_data[f'{feature_name}_min'] = min(data_list)
              team_data[f'{feature_name}_std'] =
round(np.std(data_list), 4)

          return team_data
```

```
[249] df_players.dtypes
```

```
match_id            int64
player_slot         int64
gold                int64
gold_spent          int64
kills               int64
deaths              int64
assists             int64
denies              int64
last_hits           int64
hero_damage         int64
tower_damage        int64
level               int64
gold_buyback      float64
dtype: object
```

```
[250] def aggregation_data(match_id, team, team_data: dict):
          # getting the player list
          player_ids = radiant_pl if team == 'radiant' else dire_pl

          filter_players = (df_players.player_slot.isin(player_ids)) &
(df_players.match_id == match_id)
          df_team_players = df_players[filter_players]

          for feature in player_features:
              team_data = stat_agg(player_features[feature], feature,
df_team_players[feature], team_data)
```

```
        return team_data
```

```
[251]  tf = df_team_fights_players[df_team_fights_players.match_id == 0]
```

Final Data Aggregation

```
[252]  for idx, row in match_file.iterrows():
           match_id = row['match_id']
           duration = row['duration']

           # Tower, barracks, ancient status
           tower_radiant, tower_dire =
       tower_status(row['tower_status_radiant'],
       row['tower_status_dire'])
           barracks_radiant, barracks_dire =
       barracks_status(row['barracks_status_radiant'],
       row['barracks_status_dire'])

           # teamfights result
           loss_radiant, loss_dire =
       teamfight_result(df_team_fights_players[df_team_fights_players.ma
       tch_id == match_id])

           # naughty word count
           naughty_counts = None
           try:
               naughty_counts = df_match_naughty_counts.loc[match_id]
           except:
               pass

           radiant_naughty_count = 0
           dire_naughty_count = 0

           radiant_naughty_count =
       naughty_counts['radiant_naughty_count'] if naughty_counts is not
       None else 0
           dire_naughty_count = naughty_counts['radiant_naughty_count']
       if naughty_counts is not None else 0

           #-- radiant --#
           # init
           team_radiant = {'match_id': match_id, 'duration': duration}
           # result
           team_radiant['result'] = 1 if row['radiant_win'] else 0
           # tower, barrack, ancient comparison data
```

```
    team_radiant['top_towers'] = tower_radiant['top'] -
tower_dire['top']
    team_radiant['mid_towers'] = tower_radiant['mid'] -
tower_dire['mid']
    team_radiant['bottom_towers'] = tower_radiant['bottom'] -
tower_dire['bottom']
    team_radiant['ancient_status'] = tower_radiant['ancient'] -
tower_dire['ancient']
    team_radiant['top_barracks'] = barracks_radiant['top'] -
barracks_dire['top']
    team_radiant['mid_barracks'] = barracks_radiant['mid'] -
barracks_dire['mid']
    team_radiant['bottom_barracks'] = barracks_radiant['bottom']
- barracks_dire['bottom']
    # aggregating data from players, abilities
    team_radiant = aggregation_data(match_id, 'radiant',
team_radiant)
    # teamfight
    team_radiant['teamfight_loss'] = loss_radiant
    # naughty count
    team_radiant['has_negative_chat'] = True if
radiant_naughty_count > 0 else False

    #-- dire --#
    # init
    team_dire = {'match_id': match_id, 'duration': duration}
    # result
    team_dire['result'] = 0 if row['radiant_win'] else 1
    # tower, barrack, ancient comparison data
    team_dire['top_towers'] = - tower_radiant['top'] +
tower_dire['top']
    team_dire['mid_towers'] = - tower_radiant['mid'] +
tower_dire['mid']
    team_dire['bottom_towers'] = - tower_radiant['bottom'] +
tower_dire['bottom']
    team_dire['ancient_status'] = - tower_radiant['ancient'] +
tower_dire['ancient']
    team_dire['top_barracks'] = - barracks_radiant['top'] +
barracks_dire['top']
    team_dire['mid_barracks'] = - barracks_radiant['mid'] +
barracks_dire['mid']
    team_dire['bottom_barracks'] = - barracks_radiant['bottom'] +
barracks_dire['bottom']
    # aggregating data from players, abilities
    team_dire = aggregation_data(match_id, 'dire', team_dire)
    # teamfight
    team_dire['teamfight_loss'] = loss_dire
    # naughty word count
    team_dire['has_negative_chat'] = True if dire_naughty_count >
0 else False

    match_data.append(team_radiant)
```

```
        match_data.append(team_dire)
```

```
[253]  # Permenet storage for the cleaned and aggregated data
       df_match_data = pd.DataFrame(match_data)
       df_match_data.to_csv('data_clean/cleaned_match_data.csv')
```

Model Building

Util function for model training

```
[254]  # util function for plot building

       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)})
```

```
[255]  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("-----------------------------------------")
```

Loading the data set and initialization

[256]
```
df =
pd.read_csv(f'{os.getcwd()}/data_clean/cleaned_match_data.csv')
```

[257]
```
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	1	2582	0	-2	-2
3	3	1	2582	1	2	2
4	4	2	2716	0	-1	-2

5 rows × 54 columns

[258]
```
df.shape
```

```
(97340, 54)
```

[259]
```
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
```

[260]

```
df.head()
```

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

```
[261] df.corr()
```

	duration	result	top_towers	mid_towers
duration	1.000000	0.000000	0.000000	0.000000
result	0.000000	1.000000	0.780383	0.869118
top_towers	0.000000	0.780383	1.000000	0.807993
mid_towers	0.000000	0.869118	0.807993	1.000000
bottom_towers	0.000000	0.825150	0.792817	0.856399
ancient_status	0.000000	0.986854	0.790515	0.880309
top_barracks	0.000000	0.809060	0.880296	0.738982
mid_barracks	0.000000	0.865912	0.752830	0.801978
bottom_barracks	0.000000	0.942937	0.787292	0.914213
gold_total	0.230393	0.739364	0.548155	0.613872
gold_max	0.221346	0.657267	0.493350	0.550441
gold_min	0.167828	0.546893	0.399398	0.443550
gold_std	0.188805	0.543037	0.411466	0.459861
gold_spent_avg	0.785806	0.484104	0.449556	0.489234
gold_spent_max	0.688972	0.423775	0.394268	0.422548
gold_spent_min	0.656762	0.435675	0.407188	0.449448
gold_spent_std	0.484056	0.281613	0.260555	0.271512
kills_total	0.573474	0.533049	0.476214	0.559658
deaths_total	0.580580	-0.524336	-0.466327	-0.548455
deaths_max	0.556379	-0.439907	-0.396466	-0.470040
deaths_min	0.474532	-0.509335	-0.445810	-0.519707
deaths_std	0.315641	-0.114353	-0.115577	-0.146303
assists_avg	0.588025	0.447058	0.355167	0.441666
assists_max	0.556654	0.450620	0.367363	0.450477

	duration	result	top_towers	mid_towers
assists_min	0.526493	0.365407	0.278225	0.355802
assists_std	0.350213	0.343708	0.299283	0.352559
denies_avg	0.089871	0.116808	0.133229	0.145287
denies_max	0.059884	0.091221	0.105895	0.114136
denies_min	0.081750	0.059375	0.061325	0.069310
denies_std	0.046105	0.083125	0.098035	0.105156
last_hits_avg	0.848631	0.165490	0.177155	0.177494
last_hits_max	0.736928	0.175361	0.184407	0.178210
last_hits_min	0.602318	0.068833	0.072591	0.087824
last_hits_std	0.633465	0.174286	0.181549	0.171989
hero_damage_total	0.716275	0.366707	0.337045	0.393043
hero_damage_max	0.584600	0.367605	0.333292	0.389684
hero_damage_min	0.493317	0.197365	0.189301	0.219100
hero_damage_std	0.428333	0.320814	0.285607	0.335229
tower_damage_total	0.179902	0.883605	0.815496	0.858420
tower_damage_max	0.192584	0.785853	0.731208	0.757442
tower_damage_min	0.006646	0.613203	0.560518	0.619057
tower_damage_std	0.199976	0.747727	0.696751	0.716737
level_total	0.867515	0.320451	0.278441	0.324320
level_max	0.776659	0.318160	0.296039	0.338086
level_min	0.780951	0.293043	0.246086	0.292913
level_std	0.063224	0.059134	0.088415	0.083000
gold_buyback_avg	-0.585446	0.189747	0.141131	0.142121
gold_buyback_max	-0.213387	0.061800	0.036464	0.024963
gold_buyback_min	-0.602820	0.161707	0.121163	0.131729
gold_buyback_std	0.582075	-0.154202	-0.118319	-0.133516
teamfight_loss	0.242865	-0.565127	-0.517812	-0.609846

	duration	result	top_towers	mid_towers
has_negative_chat	0.047288	0.000000	0.000000	0.000000

52 rows × 52 columns

[262]
```
#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	2582	0	53	5.4	0	4.4091
3	2582	1	37	3.2	0	2.3152
4	2716	0	49	2.0	0	2.1909

[263]
```
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
)
```

[264]
```
x_train, x_val, y_train, y_val = train_test_split(x_train,
y_train, test_size = 0.2, random_state = 1)
```

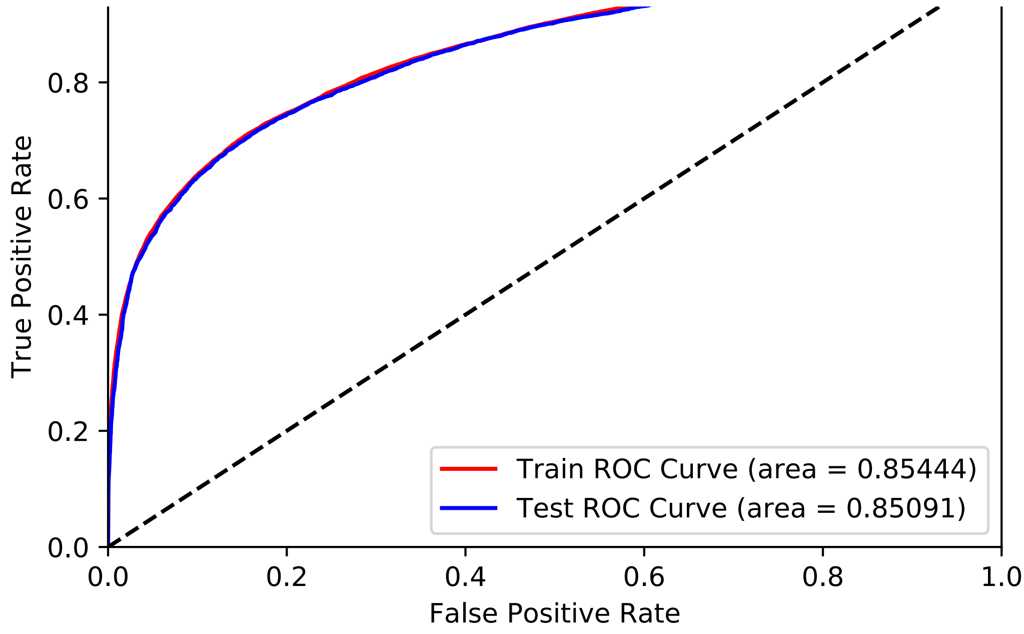
XGBoost

[265]
```
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: 0.6941138835572617
f1_score: 0.7568981756026092
accuracy_score: 0.7762653097259901
AUC: 0.8544377038191739
Predicted      0      1    All
True
0          11490   8265  19755
1          11586   8454  20040
All        23076  16719  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.6896090534979424
f1_score: 0.7531037582158305
accuracy_score: 0.7742449147318676
AUC: 0.8509072664056022
Predicted     0     1   All
True
0          1140   833  1973
1          1186   851  2037
All        2326  1684  4010
------------------------------------------
```

[266]
```
%matplotlib inline
plot_roc_curve(y_train, y_pred_train, y_test, y_pred_test)
```



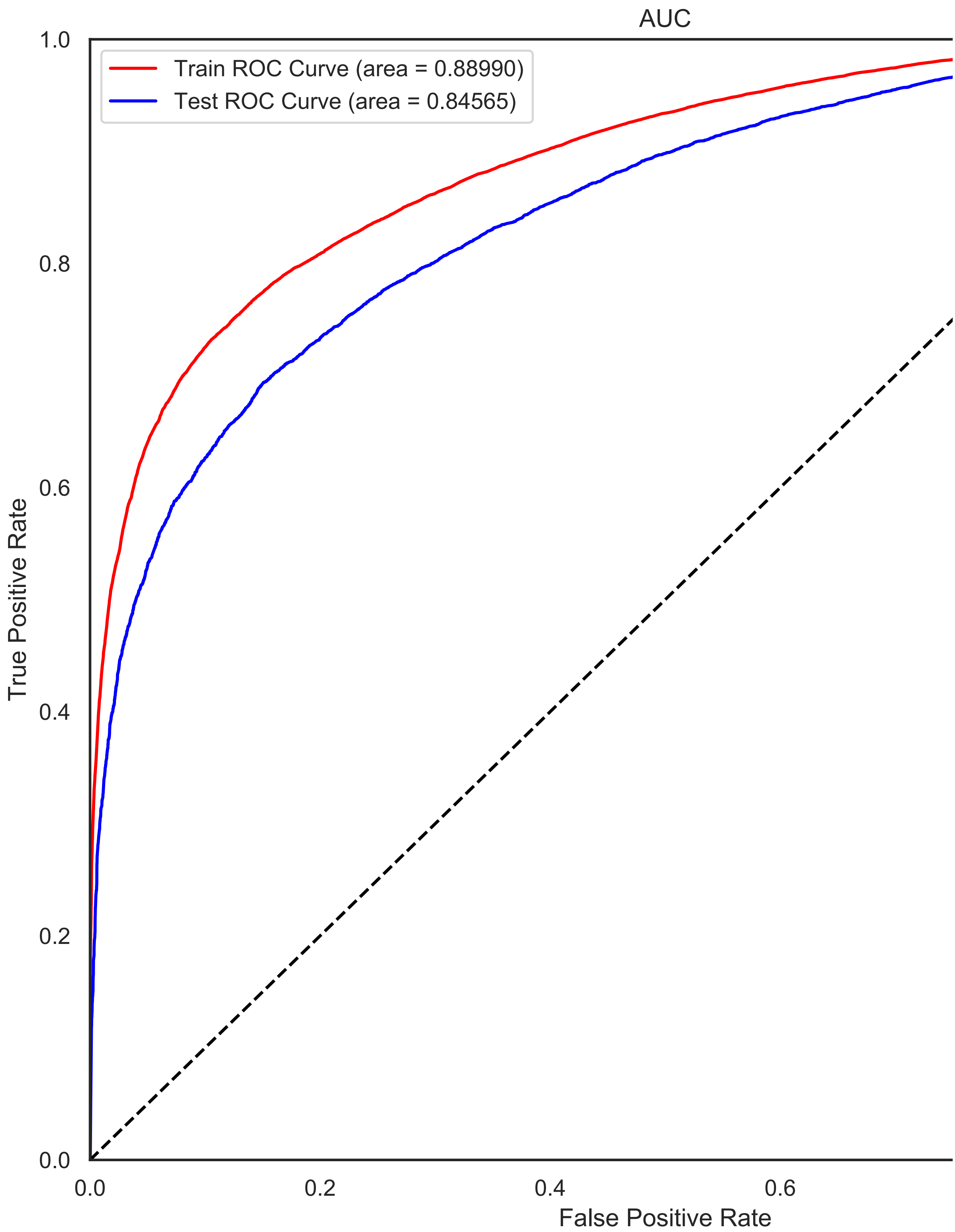
GBDT

```
[267] 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: 0.7428982725527831
f1_score: 0.798892290756476
accuracy_score: 0.8123184101963177
AUC: 0.8898988331080125
Predicted      0      1    All
True
0          11270   8485  19755
1          11322   8718  20040
All        22592  17203  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.7044238683127572
f1_score: 0.7538258284707696
accuracy_score: 0.7702897061845079
AUC: 0.8456451233639146
Predicted     0     1   All
True
```

```
0           1117   856  1973
1           1140   897  2037
All         2257  1753  4010
---------------------------------------------
```

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



LightGBM

```
[269]  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
       )
```

```
[270]  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: 0.662861
Training until validation scores don't improve for 20 rounds
[2]     valid_0's binary_logloss: 0.637722
[3]     valid_0's binary_logloss: 0.616462
[4]     valid_0's binary_logloss: 0.598544
[5]     valid_0's binary_logloss: 0.583187
[6]     valid_0's binary_logloss: 0.570031
[7]     valid_0's binary_logloss: 0.558754
[8]     valid_0's binary_logloss: 0.54899
[9]     valid_0's binary_logloss: 0.540151
[10]    valid_0's binary_logloss: 0.532668
[11]    valid_0's binary_logloss: 0.525932
```

```
[12]	valid_0's binary_logloss: 0.520214
[13]	valid_0's binary_logloss: 0.515268
[14]	valid_0's binary_logloss: 0.510909
[15]	valid_0's binary_logloss: 0.506911
[16]	valid_0's binary_logloss: 0.503441
[17]	valid_0's binary_logloss: 0.500306
[18]	valid_0's binary_logloss: 0.497588
[19]	valid_0's binary_logloss: 0.495152
[20]	valid_0's binary_logloss: 0.492824
[21]	valid_0's binary_logloss: 0.490803
[22]	valid_0's binary_logloss: 0.48906
[23]	valid_0's binary_logloss: 0.487419
[24]	valid_0's binary_logloss: 0.485901
[25]	valid_0's binary_logloss: 0.484638
[26]	valid_0's binary_logloss: 0.483488
[27]	valid_0's binary_logloss: 0.482453
[28]	valid_0's binary_logloss: 0.481493
[29]	valid_0's binary_logloss: 0.480708
[30]	valid_0's binary_logloss: 0.479867
[31]	valid_0's binary_logloss: 0.479074
[32]	valid_0's binary_logloss: 0.478372
[33]	valid_0's binary_logloss: 0.477704
[34]	valid_0's binary_logloss: 0.477049
[35]	valid_0's binary_logloss: 0.476537
[36]	valid_0's binary_logloss: 0.475989
[37]	valid_0's binary_logloss: 0.47551
[38]	valid_0's binary_logloss: 0.475056
[39]	valid_0's binary_logloss: 0.474666
[40]	valid_0's binary_logloss: 0.474254
[41]	valid_0's binary_logloss: 0.473985
[42]	valid_0's binary_logloss: 0.473666
[43]	valid_0's binary_logloss: 0.473386
[44]	valid_0's binary_logloss: 0.473144
[45]	valid_0's binary_logloss: 0.472886
[46]	valid_0's binary_logloss: 0.472671
[47]	valid_0's binary_logloss: 0.472466
[48]	valid_0's binary_logloss: 0.472273
[49]	valid_0's binary_logloss: 0.472119
[50]	valid_0's binary_logloss: 0.471931
[51]	valid_0's binary_logloss: 0.471807
[52]	valid_0's binary_logloss: 0.471683
[53]	valid_0's binary_logloss: 0.471563
[54]	valid_0's binary_logloss: 0.471379
[55]	valid_0's binary_logloss: 0.471284
[56]	valid_0's binary_logloss: 0.471142
[57]	valid_0's binary_logloss: 0.471033
[58]	valid_0's binary_logloss: 0.470944
[59]	valid_0's binary_logloss: 0.470861
[60]	valid_0's binary_logloss: 0.470715
[61]	valid_0's binary_logloss: 0.470559
[62]	valid_0's binary_logloss: 0.470442
[63]	valid_0's binary_logloss: 0.470339
```

```
[64]	valid_0's binary_logloss: 0.470259
[65]	valid_0's binary_logloss: 0.470181
[66]	valid_0's binary_logloss: 0.470053
[67]	valid_0's binary_logloss: 0.469932
[68]	valid_0's binary_logloss: 0.469852
[69]	valid_0's binary_logloss: 0.46986
[70]	valid_0's binary_logloss: 0.46978
[71]	valid_0's binary_logloss: 0.469656
[72]	valid_0's binary_logloss: 0.469606
[73]	valid_0's binary_logloss: 0.469558
[74]	valid_0's binary_logloss: 0.469504
[75]	valid_0's binary_logloss: 0.469407
[76]	valid_0's binary_logloss: 0.469354
[77]	valid_0's binary_logloss: 0.469315
[78]	valid_0's binary_logloss: 0.469221
[79]	valid_0's binary_logloss: 0.469155
[80]	valid_0's binary_logloss: 0.469103
[81]	valid_0's binary_logloss: 0.469056
[82]	valid_0's binary_logloss: 0.469002
[83]	valid_0's binary_logloss: 0.468929
[84]	valid_0's binary_logloss: 0.468907
[85]	valid_0's binary_logloss: 0.468886
[86]	valid_0's binary_logloss: 0.468866
[87]	valid_0's binary_logloss: 0.468819
[88]	valid_0's binary_logloss: 0.468772
[89]	valid_0's binary_logloss: 0.468739
[90]	valid_0's binary_logloss: 0.468716
[91]	valid_0's binary_logloss: 0.468668
[92]	valid_0's binary_logloss: 0.468597
[93]	valid_0's binary_logloss: 0.468596
[94]	valid_0's binary_logloss: 0.468521
[95]	valid_0's binary_logloss: 0.468518
[96]	valid_0's binary_logloss: 0.468501
[97]	valid_0's binary_logloss: 0.468456
[98]	valid_0's binary_logloss: 0.468422
[99]	valid_0's binary_logloss: 0.468357
[100]	valid_0's binary_logloss: 0.468301
[101]	valid_0's binary_logloss: 0.468299
[102]	valid_0's binary_logloss: 0.468276
[103]	valid_0's binary_logloss: 0.468261
[104]	valid_0's binary_logloss: 0.468241
[105]	valid_0's binary_logloss: 0.468222
[106]	valid_0's binary_logloss: 0.468226
[107]	valid_0's binary_logloss: 0.468196
[108]	valid_0's binary_logloss: 0.46818
[109]	valid_0's binary_logloss: 0.468169
[110]	valid_0's binary_logloss: 0.46813
[111]	valid_0's binary_logloss: 0.468123
[112]	valid_0's binary_logloss: 0.468142
[113]	valid_0's binary_logloss: 0.468126
[114]	valid_0's binary_logloss: 0.468093
[115]	valid_0's binary_logloss: 0.468089
```

```
[116]	valid_0's binary_logloss: 0.4681
[117]	valid_0's binary_logloss: 0.468084
[118]	valid_0's binary_logloss: 0.468096
[119]	valid_0's binary_logloss: 0.468061
[120]	valid_0's binary_logloss: 0.468076
[121]	valid_0's binary_logloss: 0.468053
[122]	valid_0's binary_logloss: 0.46802
[123]	valid_0's binary_logloss: 0.467982
[124]	valid_0's binary_logloss: 0.46798
[125]	valid_0's binary_logloss: 0.467995
[126]	valid_0's binary_logloss: 0.467977
[127]	valid_0's binary_logloss: 0.46795
[128]	valid_0's binary_logloss: 0.467951
[129]	valid_0's binary_logloss: 0.46795
[130]	valid_0's binary_logloss: 0.467958
[131]	valid_0's binary_logloss: 0.467963
[132]	valid_0's binary_logloss: 0.46797
[133]	valid_0's binary_logloss: 0.467977
[134]	valid_0's binary_logloss: 0.467975
[135]	valid_0's binary_logloss: 0.467971
[136]	valid_0's binary_logloss: 0.467956
[137]	valid_0's binary_logloss: 0.467941
[138]	valid_0's binary_logloss: 0.467938
[139]	valid_0's binary_logloss: 0.467928
[140]	valid_0's binary_logloss: 0.467927
[141]	valid_0's binary_logloss: 0.467928
[142]	valid_0's binary_logloss: 0.467924
[143]	valid_0's binary_logloss: 0.467943
[144]	valid_0's binary_logloss: 0.467962
[145]	valid_0's binary_logloss: 0.467971
[146]	valid_0's binary_logloss: 0.467966
[147]	valid_0's binary_logloss: 0.467983
[148]	valid_0's binary_logloss: 0.467971
[149]	valid_0's binary_logloss: 0.467966
[150]	valid_0's binary_logloss: 0.467968
[151]	valid_0's binary_logloss: 0.467966
[152]	valid_0's binary_logloss: 0.467945
[153]	valid_0's binary_logloss: 0.467966
[154]	valid_0's binary_logloss: 0.467942
[155]	valid_0's binary_logloss: 0.467931
[156]	valid_0's binary_logloss: 0.467925
[157]	valid_0's binary_logloss: 0.467927
[158]	valid_0's binary_logloss: 0.467946
[159]	valid_0's binary_logloss: 0.467922
[160]	valid_0's binary_logloss: 0.467931
[161]	valid_0's binary_logloss: 0.467938
[162]	valid_0's binary_logloss: 0.467933
[163]	valid_0's binary_logloss: 0.467909
[164]	valid_0's binary_logloss: 0.467897
[165]	valid_0's binary_logloss: 0.467879
[166]	valid_0's binary_logloss: 0.467888
[167]	valid_0's binary_logloss: 0.4679
```

```
[168]	valid_0's binary_logloss: 0.467901
[169]	valid_0's binary_logloss: 0.467894
[170]	valid_0's binary_logloss: 0.467887
[171]	valid_0's binary_logloss: 0.467884
[172]	valid_0's binary_logloss: 0.467877
[173]	valid_0's binary_logloss: 0.467887
[174]	valid_0's binary_logloss: 0.467883
[175]	valid_0's binary_logloss: 0.467894
[176]	valid_0's binary_logloss: 0.467886
[177]	valid_0's binary_logloss: 0.467885
[178]	valid_0's binary_logloss: 0.46787
[179]	valid_0's binary_logloss: 0.467889
[180]	valid_0's binary_logloss: 0.46788
[181]	valid_0's binary_logloss: 0.46789
[182]	valid_0's binary_logloss: 0.467857
[183]	valid_0's binary_logloss: 0.46784
[184]	valid_0's binary_logloss: 0.467829
[185]	valid_0's binary_logloss: 0.467804
[186]	valid_0's binary_logloss: 0.4678
[187]	valid_0's binary_logloss: 0.46779
[188]	valid_0's binary_logloss: 0.467804
[189]	valid_0's binary_logloss: 0.467794
[190]	valid_0's binary_logloss: 0.467786
[191]	valid_0's binary_logloss: 0.467782
[192]	valid_0's binary_logloss: 0.467782
[193]	valid_0's binary_logloss: 0.467759
[194]	valid_0's binary_logloss: 0.467773
[195]	valid_0's binary_logloss: 0.467751
[196]	valid_0's binary_logloss: 0.467735
[197]	valid_0's binary_logloss: 0.467716
[198]	valid_0's binary_logloss: 0.467713
[199]	valid_0's binary_logloss: 0.467703
[200]	valid_0's binary_logloss: 0.467709
[201]	valid_0's binary_logloss: 0.46772
[202]	valid_0's binary_logloss: 0.467724
[203]	valid_0's binary_logloss: 0.467723
[204]	valid_0's binary_logloss: 0.467713
[205]	valid_0's binary_logloss: 0.467703
[206]	valid_0's binary_logloss: 0.4677
[207]	valid_0's binary_logloss: 0.467687
[208]	valid_0's binary_logloss: 0.467676
[209]	valid_0's binary_logloss: 0.467678
[210]	valid_0's binary_logloss: 0.467669
[211]	valid_0's binary_logloss: 0.467668
[212]	valid_0's binary_logloss: 0.467672
[213]	valid_0's binary_logloss: 0.467634
[214]	valid_0's binary_logloss: 0.467629
[215]	valid_0's binary_logloss: 0.467598
[216]	valid_0's binary_logloss: 0.467582
[217]	valid_0's binary_logloss: 0.467569
[218]	valid_0's binary_logloss: 0.467565
[219]	valid_0's binary_logloss: 0.467575
```

```
[220]	valid_0's binary_logloss: 0.467585
[221]	valid_0's binary_logloss: 0.467579
[222]	valid_0's binary_logloss: 0.467567
[223]	valid_0's binary_logloss: 0.467605
[224]	valid_0's binary_logloss: 0.467596
[225]	valid_0's binary_logloss: 0.467585
[226]	valid_0's binary_logloss: 0.467583
[227]	valid_0's binary_logloss: 0.467574
[228]	valid_0's binary_logloss: 0.467571
[229]	valid_0's binary_logloss: 0.467581
[230]	valid_0's binary_logloss: 0.467568
[231]	valid_0's binary_logloss: 0.467576
[232]	valid_0's binary_logloss: 0.467573
[233]	valid_0's binary_logloss: 0.467568
[234]	valid_0's binary_logloss: 0.467579
[235]	valid_0's binary_logloss: 0.467579
[236]	valid_0's binary_logloss: 0.467578
[237]	valid_0's binary_logloss: 0.467582
[238]	valid_0's binary_logloss: 0.467575
Early stopping, best iteration is:
[218]	valid_0's binary_logloss: 0.467565
```

```
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)
```

[271]

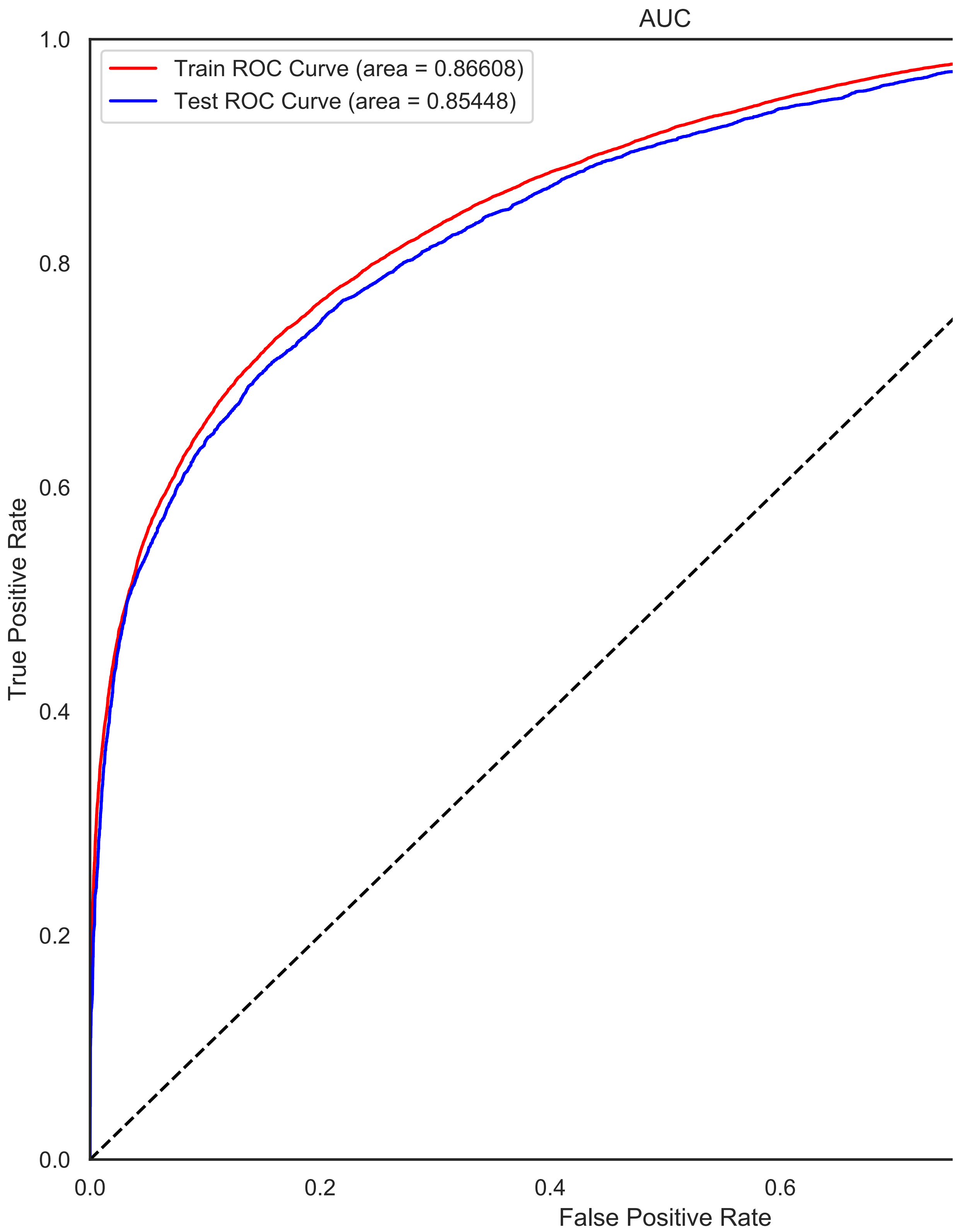
```
y_pred_test = gbm_clf.predict_proba(x_test)[:, 1]
y_pred_train = gbm_clf.predict_proba(x_train)[:, 1]
important_stats(y_train, y_pred_train, "train result summary: ")
important_stats(y_test, y_pred_test, "test result summary: ")
```

```
------------------------------------------
train result summary:
recall: 0.7262316058861165
f1_score: 0.7723476278769115
accuracy_score: 0.7851742459508484
AUC: 0.8660834159901322
Predicted      0      1    All
True
0          11014   8741  19755
1          11150   8890  20040
All        22164  17631  39795
------------------------------------------
------------------------------------------
```

```
test result summary:
recall: 0.7155349794238683
f1_score: 0.7613574165298304
accuracy_score: 0.7760427367988494
AUC: 0.8544830394669962
Predicted     0     1   All
True
0          1096   877  1973
1          1140   897  2037
All        2236  1774  4010
------------------------------------------
```

[272]

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



GridSearch for GBDT

[273]

```
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([1.18589277, 1.75672255, 2.3179038 ,
2.85591702, 3.43987155,
         4.39900851, 4.59655123]),
  'std_fit_time': array([0.03619218, 0.01933421, 0.06160265, 0.03179153,
0.12833467,
         0.17601816, 0.15251512]),
  'mean_score_time': array([0.03278232, 0.04221201, 0.04960127,
0.05456357, 0.06191044,
         0.07364426, 0.07276506]),
  'std_score_time': array([0.00061949, 0.00232535, 0.00196299,
0.00048463, 0.00196626,
         0.0015026 , 0.00270393]),
  '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([0.84704343, 0.84934   , 0.85066351,
0.85164848, 0.85233725,
         0.85252898, 0.85273903]),
  'split1_test_score': array([0.85097429, 0.85288895, 0.85389058,
0.85461955, 0.85488261,
         0.8549854 , 0.85511101]),
  'split2_test_score': array([0.84803328, 0.8500056 , 0.85091844,
0.851407  , 0.85166438,
         0.85177852, 0.85186379]),
  'split3_test_score': array([0.85097374, 0.85333805, 0.85438446,
0.85489024, 0.85523669,
         0.8553798 , 0.8555119 ]),
  'split4_test_score': array([0.84530619, 0.8475806 , 0.849208  ,
0.85002687, 0.85031076,
         0.85070306, 0.85077498]),
  'mean_test_score': array([0.84846619, 0.85063064, 0.851813  ,
0.85251843, 0.85288634,
         0.85307515, 0.85320014]),
  'std_test_score': array([0.00222601, 0.00218125, 0.0019918 ,
0.00190998, 0.00189407,
         0.00182026, 0.00183714]),
  'rank_test_score': array([7, 6, 5, 4, 3, 2, 1], dtype=int32)},
```

```
 {'n_estimators': 80},
 0.8532001422764374)
```

[274]
```python
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([4.49697628, 5.01665998, 5.44260459,
5.96744561, 6.48579392,
         6.98027577, 7.64321055, 8.06507726]),
  'std_fit_time': array([0.03594253, 0.07112514, 0.03123464, 0.05553373,
0.05426594,
         0.03553206, 0.11960741, 0.08641089]),
  'mean_score_time': array([0.0742712 , 0.07820964, 0.0815258 ,
0.08630128, 0.09127488,
         0.09622941, 0.10427294, 0.10570512]),
  'std_score_time': array([0.00099604, 0.00167455, 0.00093926,
0.00116354, 0.00110176,
         0.00118578, 0.00201494, 0.00133661]),
  '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([0.85273903, 0.85264277, 0.85271434,
0.85263474, 0.85265814,
         0.85263989, 0.85268617, 0.85268511]),
  'split1_test_score': array([0.85511101, 0.85534126, 0.85545558,
0.85530304, 0.85525374,
         0.85520609, 0.85510016, 0.85510813]),
  'split2_test_score': array([0.85186379, 0.85176068, 0.85162715,
0.85167884, 0.85157668,
         0.85153735, 0.85153533, 0.85145986]),
  'split3_test_score': array([0.8555119 , 0.85563478, 0.85550452,
```

```
0.85548764, 0.85574952,
         0.85584497, 0.85583859, 0.85585265]),
  'split4_test_score': array([0.85077498, 0.85085063, 0.85086168,
0.85093518, 0.85099117,
         0.851004  , 0.85100048, 0.85120463]),
  'mean_test_score': array([0.85320014, 0.85324602, 0.85323265,
0.85320789, 0.85324585,
         0.85324646, 0.85323215, 0.85326208]),
  'std_test_score': array([0.00183714, 0.00191855, 0.00192721,
0.00186647, 0.00192432,
         0.00194474, 0.00192045, 0.00189383]),
  'rank_test_score': array([8, 3, 5, 7, 4, 2, 6, 1], dtype=int32)},
 {'n_estimators': 150},
 0.853262076346916)
```

[275]

```
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([ 3.48853068,  3.39634099,  3.39455457,
3.43347812,  4.64345179,
          4.60082121,  4.56209478,  4.56678076,  6.34000621,
5.86132083,
          5.73890586,  5.69404888,  7.5066227 ,  7.41571236,
7.14768195,
          6.95553279,  9.45066652,  8.55600491,  8.26806479,  8.0076838
,
         11.34809875,  9.93848939,  9.26545234,  8.86286163]),
  'std_fit_time': array([0.11076404, 0.02538327, 0.01496   , 0.03094978,
0.05813355,
         0.04708578, 0.02209854, 0.07126331, 0.49522405, 0.03737676,
         0.03268715, 0.01363775, 0.07619449, 0.32928721, 0.09114357,
         0.20452219, 0.12739237, 0.11304985, 0.09440511, 0.14671984,
         0.28810278, 0.36388441, 0.29953594, 0.1161987 ]),
  'mean_score_time': array([0.05202641, 0.05200176, 0.05119538,
0.05212131, 0.06980042,
         0.06802421, 0.06749463, 0.06770535, 0.08623476, 0.08358855,
         0.08128104, 0.08558702, 0.10602479, 0.1024652 , 0.09826579,
```

```
        0.10148697, 0.12903781, 0.1191082 , 0.11520948, 0.11269598,
        0.14921942, 0.13861098, 0.13138766, 0.1272449 ]),
  'std_score_time': array([0.00101954, 0.00099614, 0.00044065,
0.00094682, 0.00527676,
        0.0023532 , 0.00133068, 0.00125558, 0.00457646, 0.00140984,
        0.00115832, 0.00933179, 0.0062306 , 0.00597242, 0.00122566,
        0.00622691, 0.00609916, 0.0019374 , 0.00169106, 0.00326453,
        0.00156325, 0.00700037, 0.00319262, 0.0032186 ]),
  '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([0.84777253, 0.84776983, 0.8477603 ,
0.8476702 , 0.85220437,
```

```
          0.85172229, 0.85251049, 0.85220056, 0.85300891, 0.85265362,
          0.85291461, 0.85278674, 0.85179373, 0.85233323, 0.85297491,
          0.85312805, 0.84921937, 0.85214609, 0.85226287, 0.85256502,
          0.84781067, 0.85090919, 0.85116725, 0.85182468]),
  'split1_test_score': array([0.85208815, 0.85176144, 0.85185197,
0.85213735, 0.85533105,
          0.85474297, 0.85456499, 0.85484166, 0.85522585, 0.8556709 ,
          0.85513008, 0.85535528, 0.85459337, 0.85540617, 0.85575402,
          0.85525774, 0.85293278, 0.85435374, 0.85542195, 0.85582571,
          0.85135313, 0.85428185, 0.85436975, 0.85457137]),
  'split2_test_score': array([0.84855929, 0.84878037, 0.84839715,
0.84875785, 0.85199249,
          0.85183366, 0.85155448, 0.85163481, 0.85196639, 0.8514462 ,
          0.85203049, 0.85130573, 0.85038003, 0.85081896, 0.85087366,
          0.85186418, 0.84818517, 0.85013629, 0.85134934, 0.85138944,
          0.84725692, 0.849962  , 0.84985691, 0.85093257]),
  'split3_test_score': array([0.85136163, 0.85114683, 0.85093625,
0.85191237, 0.85600038,
          0.85602791, 0.85568532, 0.85512496, 0.8556125 , 0.85539301,
          0.85584883, 0.85542526, 0.85484668, 0.85551519, 0.85576064,
          0.855813  , 0.85370795, 0.85508336, 0.85505763, 0.85493062,
          0.85106586, 0.85387805, 0.85400326, 0.85443065]),
  'split4_test_score': array([0.8457143 , 0.84540357, 0.84511958,
0.84555017, 0.85039735,
          0.850214  , 0.85035926, 0.84983441, 0.85122843, 0.85119234,
          0.85087747, 0.85117596, 0.85058815, 0.85092746, 0.8513934 ,
          0.85121024, 0.84842215, 0.85071472, 0.85075886, 0.85080383,
          0.84721122, 0.84915882, 0.85063965, 0.85023291]),
  'mean_test_score': array([0.84909918, 0.84897241, 0.84881305,
0.84920559, 0.85318513,
          0.85290817, 0.85293491, 0.85272728, 0.85340842, 0.85327121,
          0.8533603 , 0.85320979, 0.85244039, 0.8530002 , 0.85335133,
          0.85345464, 0.85049348, 0.85248684, 0.85297013, 0.85310293,
          0.84893956, 0.85163798, 0.85200737, 0.85239843]),
  'std_test_score': array([0.00234783, 0.00231199, 0.00239441,
0.00252356, 0.0021301 ,
          0.00214097, 0.00194636, 0.00200295, 0.00174083, 0.00191282,
          0.0018685 , 0.00186828, 0.00192456, 0.00207908, 0.00208287,
          0.00181585, 0.00234628, 0.00194981, 0.0019176 , 0.00196292,
          0.00186758, 0.00207337, 0.00183114, 0.00178992]),
  'rank_test_score': array([21, 22, 24, 20,  7, 12, 11, 13,  2,  5,  3,
6, 15,  9,  4,  1, 19,
          14, 10,  8, 23, 18, 17, 16], dtype=int32)},
 {'max_depth': 9, 'min_samples_split': 700},
 0.8534546436758774)
```

[276]

```python
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:  2.5min finished

({'mean_fit_time': array([5.79782844, 5.85214982, 5.76163898,
5.91523104, 5.81000743]),
  'std_fit_time': array([0.15607121, 0.08187269, 0.11168804, 0.26937505,
0.22457809]),
  'mean_score_time': array([0.08212223, 0.08508878, 0.08615308,
0.08794856, 0.08320065]),
  'std_score_time': array([0.00164248, 0.00373827, 0.00396298,
0.00776839, 0.00141486]),
  '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([0.85232657, 0.85251935, 0.85263793,
0.85249281, 0.85259465]),
  'split1_test_score': array([0.85587165, 0.85583704, 0.8558496 ,
0.85601258, 0.85598933]),
  'split2_test_score': array([0.85217107, 0.85179541, 0.85174686,
0.85186248, 0.85188482]),
  'split3_test_score': array([0.85613501, 0.85606541, 0.85576755,
0.8560766 , 0.85599088]),
  'split4_test_score': array([0.85046651, 0.85086711, 0.85119383,
0.85068584, 0.85118444]),
  'mean_test_score': array([0.85339416, 0.85341686, 0.85343915,
0.85342606, 0.85352883]),
  'std_test_score': array([0.00222967, 0.00213579, 0.00198892, 0.0022154
, 0.00205851]),
  'rank_test_score': array([5, 4, 2, 3, 1], dtype=int32)},
 {'min_samples_leaf': 100},
 0.8535288256697393)
```

```
[277] 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'])
```

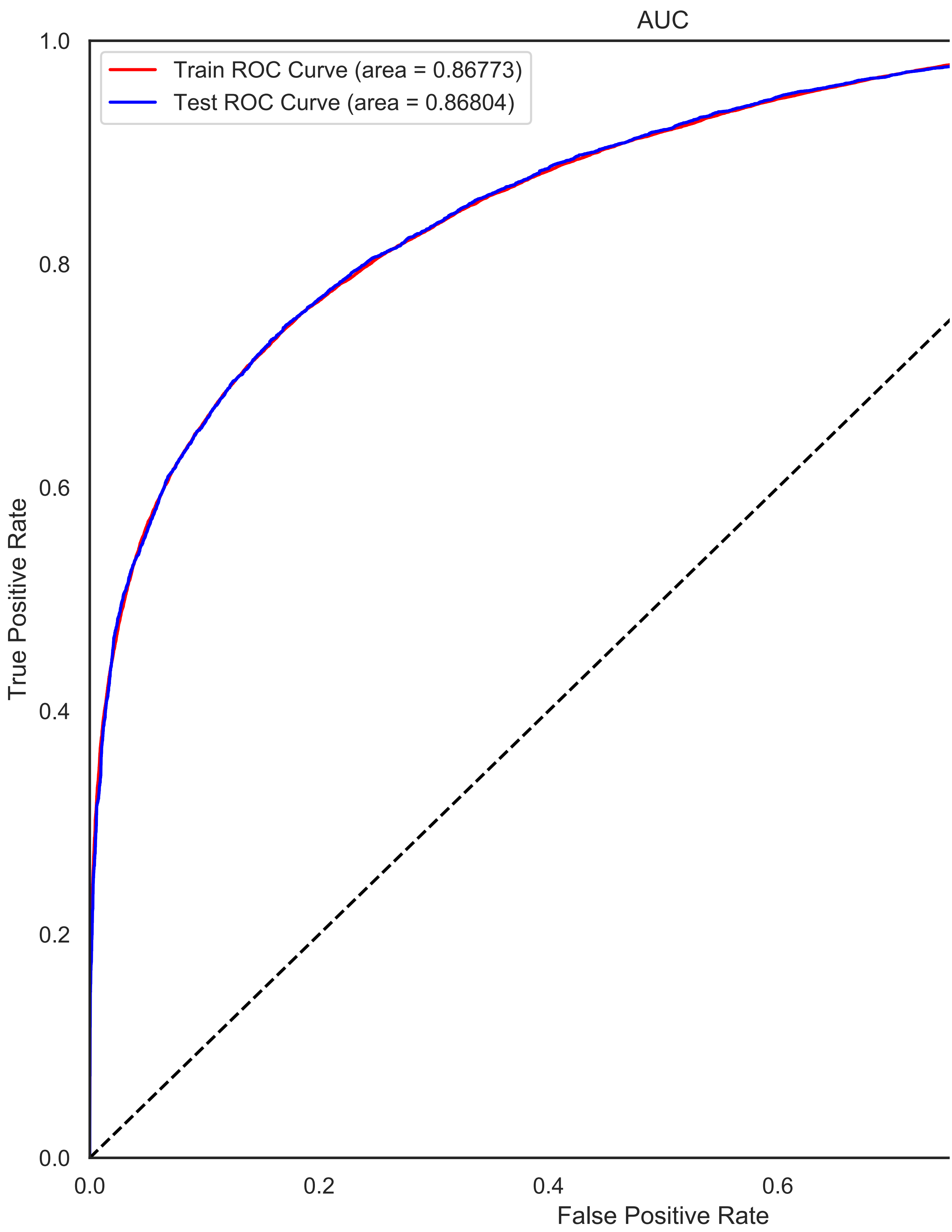
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
init=None,
                           learning_rate=0.1, loss='deviance',
max_depth=9,
                           max_features='sqrt', max_leaf_nodes=None,
                           min_impurity_decrease=0.0,
min_impurity_split=None,
                           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)
```

```
[278] 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: 0.7201855406269994
f1_score: 0.7708611539120014
accuracy_score: 0.7851581938135063
AUC: 0.8677326440769485
Predicted      0      1    All
True
0          11153   8602  19755
1          11243   8797  20040
All        22396  17399  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.7196502057613169
f1_score: 0.7705866152575048
accuracy_score: 0.7860591740291761
AUC: 0.8680424368995814
```

```
Predicted     0     1   All
True
0          1106   867  1973
1          1151   886  2037
All        2257  1753  4010
------------------------------------------
```

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

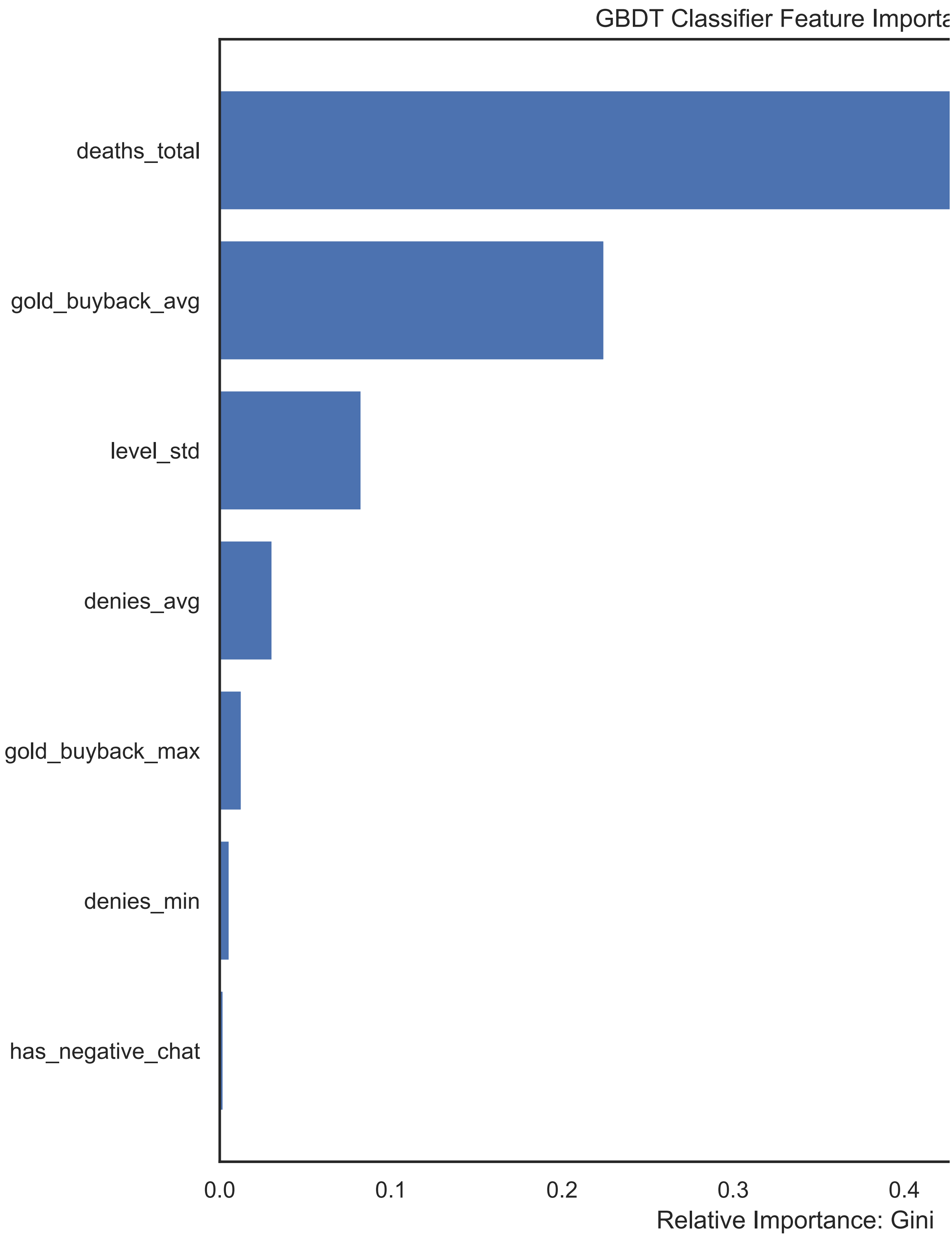


Features importance

[280]
```
gbdt_importance = gbdt_best.feature_importances_
```

[281]
```
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

[282]

```
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)
```

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

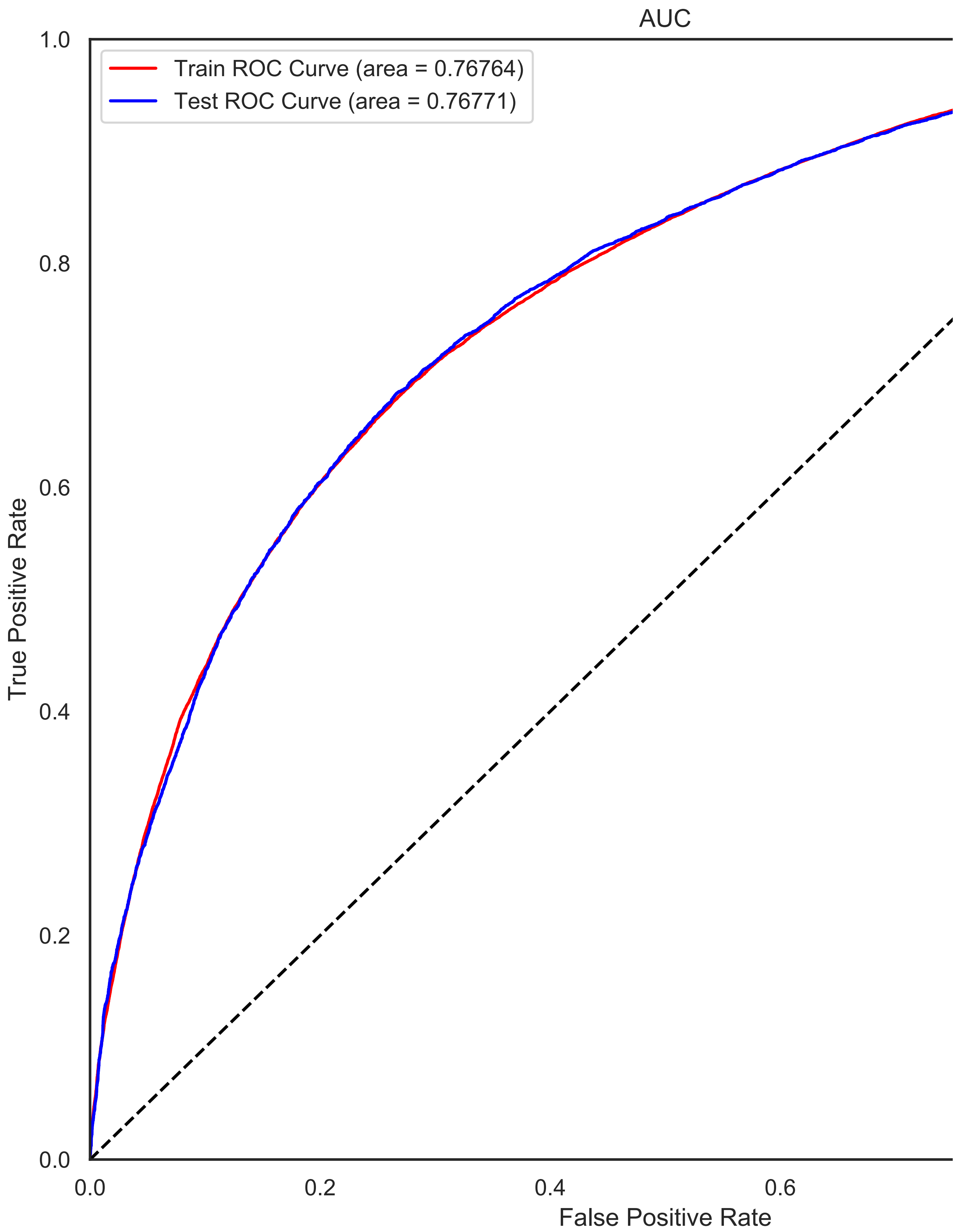
[283]

```
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: 0.5585412667946257
f1_score: 0.6477582592888015
accuracy_score: 0.6951859640111081
AUC: 0.7676367536354921
Predicted      0      1    All
True
0          12585   7170  19755
1          12738   7302  20040
All        25323  14472  39795
------------------------------------------
------------------------------------------
test result summary:
recall: 0.5618312757201646
f1_score: 0.6489987521540198
accuracy_score: 0.6965790014382577
AUC: 0.7677052884964479
Predicted     0     1   All
True
0          1251   722  1973
1          1266   771  2037
All        2517  1493  4010
------------------------------------------
```

[284]

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


AUC
Train ROC Curve (area = 0.76764)
Test ROC Curve (area = 0.76771)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
False Positive Rate