Class Weight+Model

1. Import necessary Libraries

We have not used Feature Engineering and Standardization/Normalization

In [3]:

```
#For computational and random seed purpose
import numpy as np
np.random.seed(42)
#to read csv file
import pandas as pd
#To split into train and cv data
from sklearn.model_selection import train_test_split
#To compute AUROC
from sklearn.metrics import auc,roc_auc_score
#for AUROC graph
import matplotlib.pyplot as plt
#for oversampling technique
from imblearn.over_sampling import SMOTE # (https://imbalanced-learn.org/stable/references/generated/imblearn.ove
r_sampling.SMOTE.html)
#Data is imbalanced, we need calibrated model
from sklearn.calibration import CalibratedClassifierCV
#for hyperparameter tuning and Cross-validation fold
from sklearn.model_selection import GridSearchCV,StratifiedKFold,RepeatedStratifiedKFold
#to ignore the error message
import warnings
warnings.filterwarnings("ignore")
#for heatmap and other plotting technique
import seaborn as sns
#to strandize the real value data
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.preprocessing import LabelEncoder
#To create Knn model on datasets
from sklearn.neighbors import KNeighborsClassifier
#for roc_curve
from sklearn.metrics import roc_curve,roc_auc_score,accuracy_score

import eli5
from eli5.sklearn import PermutationImportance
import joblib
import sys
sys.modules['sklearn.externals.joblib'] = joblib
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from scipy.stats import kurtosis
from scipy.stats import skew
import warnings
warnings.filterwarnings('ignore')
#from catboost import CatBoostClassifier
from sklearn.preprocessing import RobustScaler
```

In [11]:

```
#locate parent directory
data_dir = "./"

#Read the training data
df_train = pd.read_csv('train.csv')
print(df_train)
```

```
      id  target      0      1      2  ...    295    296    297    298    299
0      0     1.0 -0.098  2.165  0.681  ... -2.097  1.051 -0.414  1.038 -1.065
1      1     0.0  1.081 -0.973 -0.383  ... -1.624 -0.458 -1.099 -0.936  0.973
2      2     1.0 -0.523 -0.089 -0.348  ... -1.165 -1.544  0.004  0.800 -1.211
3      3     1.0  0.067 -0.021  0.392  ...  0.467 -0.562 -0.254 -0.533  0.238
4      4     1.0  2.347 -0.831  0.511  ...  1.378  1.246  1.478  0.428  0.253
..   ...     ...    ...    ...    ...  ...    ...    ...    ...    ...    ...
245  245     0.0 -1.199  0.466 -0.908  ... -0.243  0.525  0.281 -0.255 -1.136
246  246     0.0  0.237  0.233 -0.380  ...  1.004 -0.979  0.007  0.112 -0.558
247  247     0.0  1.411 -1.465  0.119  ... -0.727  0.461  0.760  0.168 -0.719
248  248     1.0  0.620  1.040  0.184  ...  0.478 -0.910 -0.805  2.029 -0.423
249  249     0.0  0.489  0.403  0.139  ...  0.812  0.269 -1.454 -0.625  1.474

[250 rows x 302 columns]
```

In [12]:

```
#Read test data
df_test = pd.read_csv('test.csv')
df_test
```

Out[12]:

	id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	250	0.500	-1.033	-1.595	0.309	-0.714	0.502	0.535	-0.129	-0.687	1.291	0.507	-0.317	1.848	-0.232	-0.340	-0.0
1	251	0.776	0.914	-0.494	1.347	-0.867	0.480	0.578	-0.313	0.203	1.356	-1.086	0.322	0.876	-0.563	-1.394	0.3
2	252	1.750	0.509	-0.057	0.835	-0.476	1.428	-0.701	-2.009	-1.378	0.167	-0.132	0.459	-0.341	0.014	0.184	-0.4
3	253	-0.556	-1.855	-0.682	0.578	1.592	0.512	-1.419	0.722	0.511	0.567	0.356	-0.060	0.767	-0.196	0.359	0.0
4	254	0.754	-0.245	1.173	-1.623	0.009	0.370	0.781	-1.763	-1.432	-0.930	-0.098	0.896	0.293	-0.259	0.030	-0.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19745	19995	1.069	0.517	-0.690	0.241	0.913	-0.859	0.093	-0.359	-0.047	0.713	2.191	0.774	-0.110	-0.721	0.375	0.5
19746	19996	-0.529	0.438	0.672	1.436	-0.720	0.698	-0.350	2.150	-1.241	-0.167	-0.188	0.541	-0.392	1.727	-0.965	0.5
19747	19997	-0.554	-0.936	-1.427	0.027	-0.539	0.994	-1.832	-1.156	0.474	1.483	1.524	0.143	-0.607	-1.142	2.786	-0.3
19748	19998	-0.746	1.205	0.750	-0.236	1.139	-1.727	-0.677	-1.254	-0.099	-0.724	0.014	-0.575	-0.142	1.171	-0.198	0.3
19749	19999	0.736	-0.216	-0.110	-1.404	-0.265	-1.770	0.715	0.469	1.077	0.333	-0.994	-0.331	1.009	0.607	-1.729	1.4

19750 rows × 301 columns

In [13]:

```
df_train.dropna(inplace=True)
df_test.dropna(inplace=True)
```

In [14]:

```
X_train = (df_train.drop(['id','target'],axis = 1))
X_test = (df_test.drop(['id'],axis = 1))

y_train = df_train['target']

#n_fold = 20
#folds = StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=42)
#repeated_folds = RepeatedStratifiedKFold(n_splits=20, n_repeats=20, random_state=42)
```

In [15]:

```
X_train.shape
```

Out[15]:

```
(250, 300)
```

In [16]:

```
X_test.shape
```

Out[16]:

```
(19750, 300)
```

I will do hyper tuning

In [104]:

```
def hyperparameter_model(models, params):
  '''
  Hyperparameter tuning with StratifiedKFold follow by GridSearchCV follow by␣
  ,→CalibratedClassifier
  Parameters:
  models: Instance of the model
  params: list of parameters with value fr tuning (dict)
  Return:
  grid_clf: return gridsearch model
  '''
  # Perform KCrossValidation with stratified target
  str_cv = StratifiedKFold(n_splits=11, random_state=42,shuffle=True)
  # Perform Hyperparamter using GridSearchCV
  grid_clf = GridSearchCV(models, params, cv=str_cv, return_train_score=True,scoring='roc_auc')
  # Fit the train model to evaluate score
  grid_clf.fit(X_train, y_train)
  return grid_clf
```

In [105]:

```
#kNN (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.→neighbors.KNeighborsClassifier.html)
# List of params
# List of params
params = {'n_neighbors':np.arange(3,51,2).tolist(), 'algorithm': ['kd_tree','brute']}

# Instance of knn model
knn_model = KNeighborsClassifier()
# Call hyperparameter for find the best params as possible
knn_clf = hyperparameter_model(knn_model, params)
```

In [108]:

```
print(knn_clf.best_params_)
```

```
{'algorithm': 'kd_tree', 'n_neighbors': 45}
```

In [109]:

```
from sklearn.utils import class_weight
knn_model = KNeighborsClassifier(**knn_clf.best_params_,weights='uniform')
knn_model.fit(X_train,y_train)
```

Out[109]:

```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=45)
```

In [110]:

```
y_pred = knn_model.predict(X_train)
print(y_pred)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [111]:

```
train_auc = roc_auc_score(y_train,y_pred)
print(train_auc)
```

```
0.5555555555555556
```

In [112]:

```
y_predict = knn_model.predict_proba(X_test)[:,1]
```

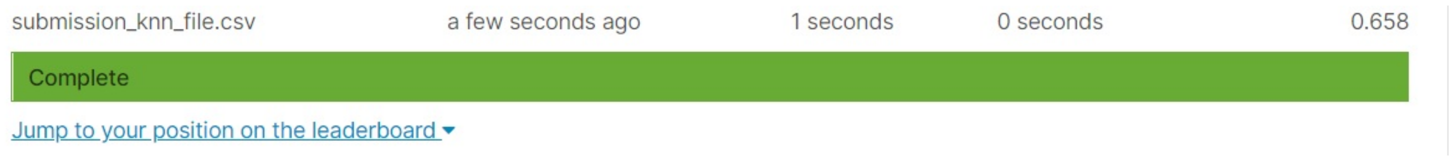
In [113]:

```
y_pred_lr_test = pd.DataFrame({"ID": df_test['id'],"Target": y_predict})


y_pred_lr_test.to_csv('submission_knn_file.csv', index=False)
y_pred_lr_test.head(20)
```

Out[113]:

	ID	Target
0	250	0.666667
1	251	0.622222
2	252	0.600000
3	253	0.644444
4	254	0.644444
5	255	0.600000
6	256	0.622222
7	257	0.644444
8	258	0.644444
9	259	0.555556
10	260	0.555556
11	261	0.600000
12	262	0.533333
13	263	0.644444
14	264	0.644444
15	265	0.644444
16	266	0.666667
17	267	0.600000
18	268	0.555556
19	269	0.555556



submission_knn_file.csv a few seconds ago 1 seconds 0 seconds 0.658

Complete

test_auc = 0.658

Logistic Regresstion Applied

In [25]:

```
#ref= https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

params = {'penalty':['l1','l2','elasticnet'],'C':[10**i for i in range(-4,5)], 'solver':['liblinear','sag']}

#the instance  of Logistic Regression

log_model = LogisticRegression(random_state=42)

#Call Hyper-parameter function to get best hyperparameter tuning

log_clf = hyperparameter_model(log_model,params)
```

In [26]:

```
print(log_clf.best_params_)
```

```
{'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [27]:

```
from sklearn import linear_model

model = LogisticRegression(class_weight = 'balanced',penalty='l1', C=0.1, solver='liblinear')

model.fit(X_train,y_train)
```

Out[27]:

```
LogisticRegression(C=0.1, class_weight='balanced', penalty='l1',
                   solver='liblinear')
```

In [28]:

```
y_pred = model.predict(X_train)
print(y_pred)
```

```
[1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1.
 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1.
 1. 1. 0. 0. 1. 0. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1.
 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 1. 0. 0. 1.
 0. 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0.
 1. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 1. 1. 0. 1. 0. 1.
 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1.
 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0.
 0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
```

In [29]:

```
train_auc_lr = roc_auc_score(y_train,y_pred)
print(train_auc_lr)
```

```
0.8951388888888888
```

In [30]:

```
y_pred_lr_test = model.predict_proba(X_test)[:,1]
print(y_pred_lr_test)
```

```
[0.66542599 0.49363699 0.54808538 ... 0.34306623 0.79406723 0.22330863]
```

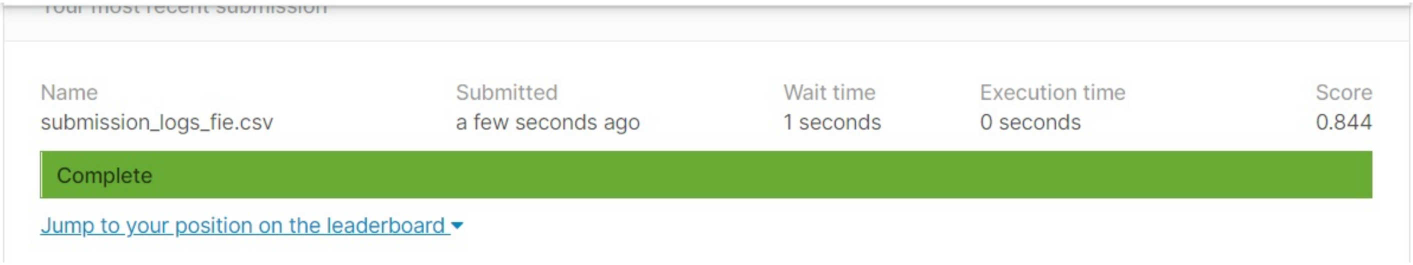
```
In [31]:
```

```
y_pred_lr_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_lr_test})


y_pred_lr_test.to_csv('submission_logs_fie.csv', index=False)
y_pred_lr_test.head(20)
```

```
Out[31]:
```

	ID	Target
0	250	0.665426
1	251	0.493637
2	252	0.548085
3	253	0.758678
4	254	0.401662
5	255	0.313830
6	256	0.339976
7	257	0.211075
8	258	0.673940
9	259	0.214554
10	260	0.511906
11	261	0.364041
12	262	0.317890
13	263	0.666236
14	264	0.286479
15	265	0.683664
16	266	0.307416
17	267	0.723669
18	268	0.436554
19	269	0.304132



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_logs_fie.csv	a few seconds ago	1 seconds	0 seconds	0.844

Complete

Jump to your position on the leaderboard ▾

logistic_test_auc = 0.844

Support Vector Machine

```
In [32]:
```

```
from sklearn.svm import SVC
```

```
In [33]:
```

```
#ref = https://scikit-learn.org/stable/modules/svm.html

params = {'C':[10**i for i in range(-4,5)],'kernel':['linear','poly','sigmoid','rdf']}

#The instance of SVC

svc_model = SVC(class_weight='balanced',random_state=42)
#call the hyper-parameter function to get best parameters

svc_clf = hyperparameter_model(svc_model,params)
```

In [34]:

```
print(svc_clf.best_params_)
```

```
{'C': 0.0001, 'kernel': 'sigmoid'}
```

In [35]:

```
svc_clf = SVC(C = 0.0001, kernel = 'sigmoid',probability=True)
svc_clf.fit(X_train,y_train)
```

Out[35]:

```
SVC(C=0.0001, kernel='sigmoid', probability=True)
```

In [36]:

```
y_pred = svc_clf.predict(X_train)
train_svm_auc = roc_auc_score(y_train,y_pred)
print(train_svm_auc)
```

```
0.5
```

In [37]:

```
y_pred_svc_test = svc_clf.predict_proba(X_test)[:,1]
```

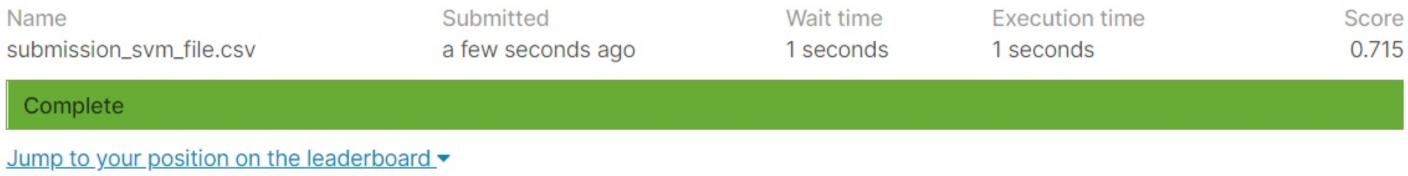
In [38]:

```
y_pred_svc_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_svc_test})


y_pred_svc_test.to_csv('submission_svm_file.csv', index=False)
y_pred_svc_test.head(10)
```

Out[38]:

	ID	Target
0	250	0.588435
1	251	0.809681
2	252	0.429262
3	253	0.868921
4	254	0.468427
5	255	0.514445
6	256	0.267929
7	257	0.790950
8	258	0.414834
9	259	0.634586



Name	Submitted	Wait time	Execution time	Score
submission_svm_file.csv	a few seconds ago	1 seconds	1 seconds	0.715

Complete

Jump to your position on the leaderboard ▾

Test_SVM_auc = 0.72

Ensemble Model : Random Forest

In [39]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [41]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
params = {'n_estimators': [10,20,30,40,50,60,100,200,300,400,500],'max_depth':[2,3,5,7,9]}

#The instance of model

rdf_model = RandomForestClassifier(random_state=42)

# Call the hyperparameter function  to get best parameter
rdf_clf = hyperparameter_model(rdf_model,params)
```

In [42]:

```
print(rdf_clf.best_params_)
```

```
{'max_depth': 2, 'n_estimators': 400}
```

In [43]:

```
rdf_clf = RandomForestClassifier(**rdf_clf.best_params_,bootstrap=True)
rdf_clf.fit(X_train,y_train)
```

Out[43]:

```
RandomForestClassifier(max_depth=2, n_estimators=400)
```

In [44]:

```
y_pred = rdf_clf.predict(X_train)
train_rdf_auc = roc_auc_score(y_train,y_pred)
print(train_rdf_auc)
```

```
0.5166666666666666
```

In [45]:

```
y_pred_rdf_test = rdf_clf.predict_proba(X_test)[:,1]
```

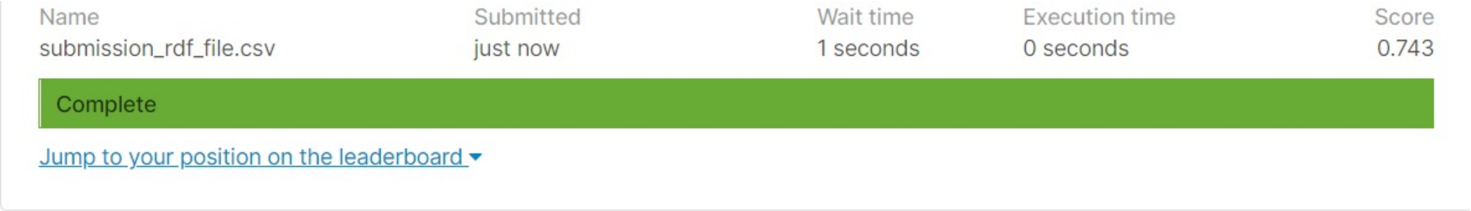
In [46]:

```
y_pred_rdf_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_rdf_test})


y_pred_rdf_test.to_csv('submission_rdf_file.csv', index=False)
y_pred_rdf_test.head(20)
```

Out[46]:

	ID	Target
0	250	0.647066
1	251	0.641644
2	252	0.610497
3	253	0.686939
4	254	0.646141
5	255	0.605542
6	256	0.585545
7	257	0.616321
8	258	0.665471
9	259	0.598260
10	260	0.653798
11	261	0.619836
12	262	0.646692
13	263	0.651312
14	264	0.613722
15	265	0.676058
16	266	0.644975
17	267	0.629752
18	268	0.597876
19	269	0.624994



Name	Submitted	Wait time	Execution time	Score
submission_rdf_file.csv	just now	1 seconds	0 seconds	0.743

Complete

Jump to your position on the leaderboard ▾

Test_rdf_auc : 0.74

Decision Tree Classifier

In [79]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [98]:

```
#ref =https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

params =  {'max_depth':[3,5,7,9]}

#The instance of Decision Tree Classifier

tree_model = DecisionTreeClassifier(random_state=42)

#Call Hyperparameter function to get best parameter

tree_clf = hyperparameter_model(tree_model,params)
```

In [99]:

```
print(tree_clf.best_params_)
```

```
{'max_depth': 3}
```

In [100]:

```
tree_clf = DecisionTreeClassifier(**tree_clf.best_params_,class_weight='balanced')
tree_clf.fit(X_train,y_train)
```

Out[100]:

```
DecisionTreeClassifier(class_weight='balanced', max_depth=3)
```

In [101]:

```
y_pred = tree_clf.predict(X_train)
train_tree_auc = roc_auc_score(y_train,y_pred)
print(train_tree_auc)
```

```
0.7937500000000001
```

In [102]:

```
y_pred_tree_test = tree_clf.predict_proba(X_test)[:,1]
```

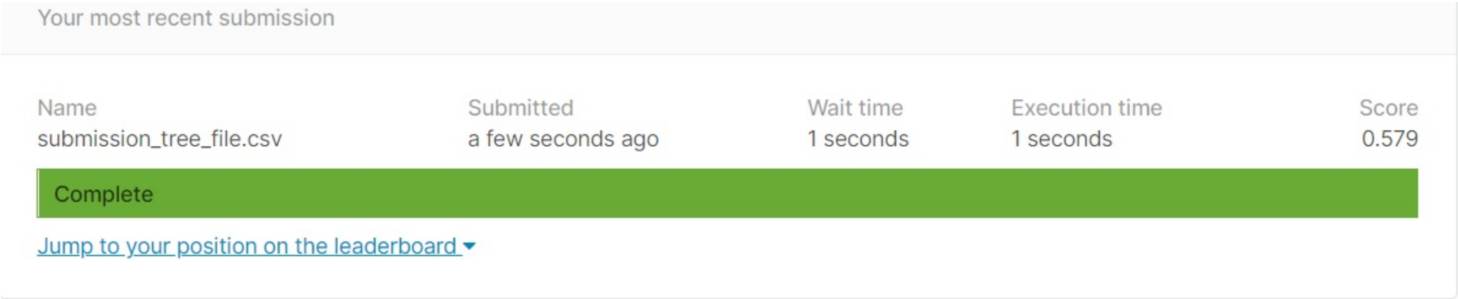
In [103]:

```
y_pred_tree_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_tree_test})


y_pred_tree_test.to_csv('submission_tree_file.csv', index=False)
y_pred_tree_test.head(20)
```

Out[103]:

	ID	Target
0	250	0.387560
1	251	0.830769
2	252	0.195143
3	253	0.830769
4	254	0.830769
5	255	0.195143
6	256	0.830769
7	257	0.830769
8	258	0.387560
9	259	0.195143
10	260	1.000000
11	261	0.195143
12	262	0.830769
13	263	0.195143
14	264	0.195143
15	265	0.830769
16	266	0.830769
17	267	0.195143
18	268	1.000000
19	269	0.616438



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_tree_file.csv	a few seconds ago	1 seconds	1 seconds	0.579

Complete

Jump to your position on the leaderboard ▾

test_auc = 0.614

XGBoost Classifier

In [54]:

```
from xgboost import XGBClassifier
```

In [71]:

```
from scipy.sparse.construct import random
#list of hyper-parameter

params = {'max_depth':[2,3,5,7,9],'n_estimators':[10,20,30,40,50,100,200,400,500]}

# The instance of XGBClassifier

xg_model = XGBClassifier(random_state=42)
# call hyparameter function to get best parameter

xg_clf = hyperparameter_model(xg_model,params)
```

In [72]:

```
print(xg_clf.best_params_)
```

```
{'max_depth': 2, 'n_estimators': 500}
```

In [73]:

```
xg_clf = XGBClassifier(**xg_clf.best_params_,scale_pos_weight=0.5)
xg_clf.fit(X_train,y_train)
```

Out[73]:

```
XGBClassifier(max_depth=2, n_estimators=500, scale_pos_weight=0.5)
```

In [74]:

```
y_pred = xg_clf.predict(X_train)
```

In [75]:

```
train_xgboost_auc = roc_auc_score(y_train,y_pred)
print(train_xgboost_auc)
```

```
1.0
```

In [76]:

```
y_pred_xg_test = xg_clf.predict_proba(X_test)[:,1]
print(y_pred_xg_test)
```

```
[0.6922573  0.62574023 0.61335033 ... 0.14424945 0.96691495 0.2348253 ]
```

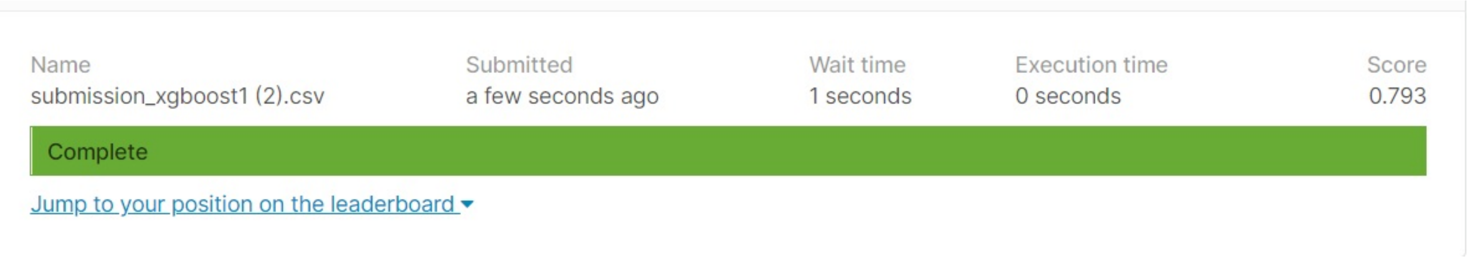
In [77]:

```
y_pred_xg_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_xg_test})


y_pred_xg_test.to_csv('submission_xgboost1.csv', index=False)
y_pred_xg_test.head(10)
```

Out[77]:

	ID	Target
0	250	0.692257
1	251	0.625740
2	252	0.613350
3	253	0.994627
4	254	0.923137
5	255	0.471000
6	256	0.337258
7	257	0.132679
8	258	0.989115
9	259	0.356931



Name	Submitted	Wait time	Execution time	Score
submission_xgboost1 (2).csv	a few seconds ago	1 seconds	0 seconds	0.793

Complete

xgboost_test_auc = 0.793

Stacking Classifier

In [78]:

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
```

In [114]:

```
from mlxtend.classifier import StackingClassifier
```

In [115]:

```
#classifier 1
knn_model = KNeighborsClassifier(algorithm ='kd_tree',n_neighbors = 45,weights='uniform')
knn_model.fit(X_train,y_train)

#Classifier 2
model = LogisticRegression(C= 0.1, penalty = 'l1', solver = 'liblinear',class_weight='balanced')
model.fit(X_train,y_train)

#Classifier 3
svc_clf = SVC(C=0.0001, kernel='sigmoid', probability=True)
svc_clf.fit(X_train,y_train)

#classifier 3

rdf_clf = RandomForestClassifier(max_depth=2, n_estimators=400,class_weight='balanced')
rdf_clf.fit(X_train,y_train)
#classifier 4

tree_clf = DecisionTreeClassifier(max_depth = 3,class_weight='balanced')
tree_clf.fit(X_train,y_train)

#classifier 5
xg_clf = XGBClassifier(max_depth=2, n_estimators=500, scale_pos_weight=0.5)
xg_clf.fit(X_train,y_train)


#Stacking Classifer

sclf = StackingClassifier(classifiers=[knn_model,model,svc_clf,rdf_clf,tree_clf,xg_clf],meta_classifier=model,use
_probas=True)

#fit the model
sclf.fit(X_train,y_train)

#predict in probabilities

y_pred = sclf.predict(X_train)
```

In [116]:

```
train_auc = roc_auc_score(y_train,y_pred)
print(train_auc)
```

```
1.0
```

In [117]:

```
y_pred_stack_test = sclf.predict_proba(X_test)[:,1]
```

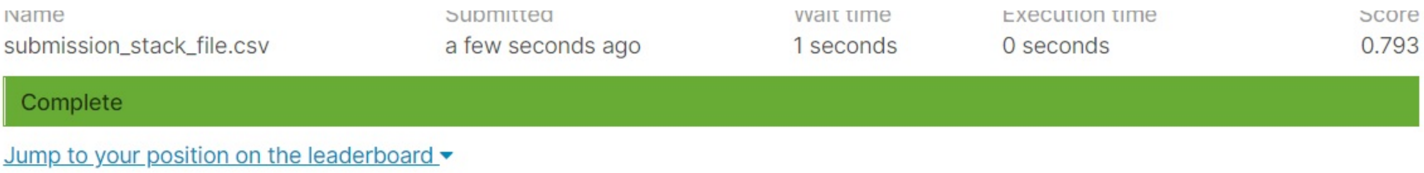
In [118]:

```
y_pred_stack_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_stack_test})


y_pred_stack_test.to_csv('submission_stack_file.csv', index=False)
y_pred_stack_test.head(20)
```

Out[118]:

	ID	Target
0	250	0.723233
1	251	0.652485
2	252	0.638395
3	253	0.921526
4	254	0.891676
5	255	0.465294
6	256	0.309231
7	257	0.139382
8	258	0.919522
9	259	0.330495
10	260	0.871123
11	261	0.453674
12	262	0.121870
13	263	0.905671
14	264	0.375028
15	265	0.914669
16	266	0.340220
17	267	0.904790
18	268	0.626804
19	269	0.832697



Name	Submitted	Wait time	Execution time	Score
submission_stack_file.csv	a few seconds ago	1 seconds	0 seconds	0.793

Complete

Jump to your position on the leaderboard

Test_auc = 0.79

Summary of All Models

In [2]:

```
from texttable import Texttable
t = Texttable()
t.add_rows([['Model','Hyper-parameter','Train AUC','Test AUC'],['Knn_Model',r"{'algorithm': 'kd_tree', 'n_neighbo
rs': 45}",0.55,0.66],
           ['logistic Regresstion',r"{'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}",0.90,0.844],['Support Ve
ctor Machine',
           r"{'C': 0.001, 'kernel': 'sigmoid'}",0.52,0.72],['XGBoost Classifier',r"{'max_depth': 2, 'n_estimators
': 500}",1.00,0.79],
            ['Random forest',r"{'max_depth': 2, 'n_estimators': 400}",0.52,0.74], ['DecisionTree',r"{'max_depth':
3}",0.79,0.58],
            ['Calibrated Model',"--",1.00,0.79]])

print(t.draw())
```

```
+------------------------+------------------------------+-----------+----------+
|         Model          |       Hyper-parameter        | Train AUC | Test AUC |
+========================+==============================+===========+==========+
| Knn_Model              | {'algorithm': 'kd_tree',     | 0.550     | 0.660    |
|                        | 'n_neighbors': 45}           |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | {'C': 0.1, 'penalty': 'l1',  | 0.900     | 0.844    |
|                        | 'solver': 'liblinear'}       |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.001, 'kernel':       | 0.520     | 0.720    |
|                        | 'sigmoid'}                   |           |          |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 2,             | 1         | 0.790    |
|                        | 'n_estimators': 500}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 2,             | 0.520     | 0.740    |
|                        | 'n_estimators': 400}         |           |          |
+------------------------+------------------------------+-----------+----------+
| DecisionTree           | {'max_depth': 3}             | 0.790     | 0.580    |
+------------------------+------------------------------+-----------+----------+
| Calibrated Model       | --                           | 1         | 0.790    |
+------------------------+------------------------------+-----------+----------+
```

Observation

1.We have read the training and test dataset. After reading both of dataset, we got it know that test dataset is having more features compare to training dataset.

1. Dropped the labled data from both train and test datasets.
2. Used GridSerach Validation for hyper-parameter tuning.
3. We have applied following machine learning algorithm: 1.KNN : The KNN algorithm trained the model with parameter(algorithm = 'kd_tree', and n_neighbors=45) and gave train_AUC=0.55 and Test Auc = 0.66 . Model is less accurate but it is not overfitted.

   2.Logistic Regression : The Logistic regression algorithm trained the model with parameter(C=0.1,penalty=l1,solver=liblinear) and gave train_AUC = 0.90 and Test_auc=0.844 which is working as expected.

   3.Support Vector Machine: The SVM algorithm trained the model with parameter(C=0.001,kernel=sigmoid) and got the train_AUC=0.52 and Test_AUC = 0.72.Model is not overfitted

   4.XGBoost Classifier: The XGBoost classifier trained the model with parameter(max_depth=2,n_estimators=500) and got the train_AUC= 1.0 and Test_AUC=0.79.Model is accurate and not overfitted

   5.Random Forest : The Random Forest classifier trained the model with parameter(max_depth=2,n_estimators=400) and got the train_AUC = 0.52 and test_AUC = 0.74. Model is not overfitted

   6.DecisionTree : The Decision Tree classifier trained the model with parameter(max_depth=3) and got the train_AUC = 0.790 and test_AUC=0.580 .Model is overfitting. Decision Tree is poorly working.

   7.Calibrated model gave train_AUC = 1.0 and Test_AUC = 0.79. Model is accurate and not overfitted.

Logistic Regression is working well with Test AUC = 0.844 from above applied algorithm