1. Import necessary Libraries

We have used only Pseudo Technique without Feature Engineering and standardization/Normalization

In [132]:

```
#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
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 [141]:

```
#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 [145]:

```
df_train['target'].value_counts()
```

Out[145]:

```
1.0    160
0.0     90
Name: target, dtype: int64
```

In [146]:

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

Out[146]:

	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	-(
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
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	-(
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
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	-(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
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
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
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	-(
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
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

19750 rows × 301 columns

In [147]:

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

In [148]:

```
col = ['target']
df_test['target'] = 0
combi = df_train.append(df_test)
number = LabelEncoder()
for i in col:
 combi[i] = number.fit_transform(combi[i].astype('str'))
 combi[i] = combi[i].astype('int')
df_train = combi[:df_train.shape[0]]
df_test = combi[df_train.shape[0]:]
```

In [149]:

```
df_train.head()
```

Out[149]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1	-0.098	2.165	0.681	-0.614	1.309	-0.455	-0.236	0.276	-2.246	1.825	-0.912	-0.107	0.305	0.102	0.826	0.4
1	1	0	1.081	-0.973	-0.383	0.326	-0.428	0.317	1.172	0.352	0.004	-0.291	2.907	1.085	2.144	1.540	0.584	1.1
2	2	1	-0.523	-0.089	-0.348	0.148	-0.022	0.404	-0.023	-0.172	0.137	0.183	0.459	0.478	-0.425	0.352	1.095	0.3
3	3	1	0.067	-0.021	0.392	-1.637	-0.446	-0.725	-1.035	0.834	0.503	0.274	0.335	-1.148	0.067	-1.010	1.048	-1.
4	4	1	2.347	-0.831	0.511	-0.021	1.225	1.594	0.585	1.509	-0.012	2.198	0.190	0.453	0.494	1.478	-1.412	0.2

5 rows × 302 columns

In [150]:

```
df_test.head()
```

Out[150]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	250	0	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	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	0.38
2	252	0	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	-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	0.08
4	254	0	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

5 rows × 302 columns

In [151]:

```
X_train = (df_train.drop(['id','target'],axis = 1))
X_test = (df_test.drop(['id','target'],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 [152]:

```
X_train = pd.DataFrame(X_train)
X_train.head()
```

Out[152]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	-0.098	2.165	0.681	-0.614	1.309	-0.455	-0.236	0.276	-2.246	1.825	-0.912	-0.107	0.305	0.102	0.826	0.417	0.177
1	1.081	-0.973	-0.383	0.326	-0.428	0.317	1.172	0.352	0.004	-0.291	2.907	1.085	2.144	1.540	0.584	1.133	1.098
2	-0.523	-0.089	-0.348	0.148	-0.022	0.404	-0.023	-0.172	0.137	0.183	0.459	0.478	-0.425	0.352	1.095	0.300	-1.044
3	0.067	-0.021	0.392	-1.637	-0.446	-0.725	-1.035	0.834	0.503	0.274	0.335	-1.148	0.067	-1.010	1.048	-1.442	0.210
4	2.347	-0.831	0.511	-0.021	1.225	1.594	0.585	1.509	-0.012	2.198	0.190	0.453	0.494	1.478	-1.412	0.270	-1.312

5 rows × 300 columns

In [153]:

```
X_test = pd.DataFrame(X_test)
X_test.head()
```

Out[153]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	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	-0.051	0.804
1	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	0.385	1.891
2	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	-0.460	-0.991
3	-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	0.080	-0.956
4	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	-0.661	0.921

5 rows × 300 columns

In [154]:

```
X_test.shape
```

Out[154]:

```
(19750, 300)
```

Used Grid Search for hyper-parameter tuning

In [155]:

```
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 [156]:

```
#kNN (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.→neighbors.KNeighborsClassifier.html)
# 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 [157]:

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

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

In [158]:

```
knn_model = KNeighborsClassifier(**knn_clf.best_params_)
knn_model.fit(X_train,y_train)
```

Out[158]:

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

In [159]:

```
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 [117]:

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

```
0.5388888888888889
```

In [119]:

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

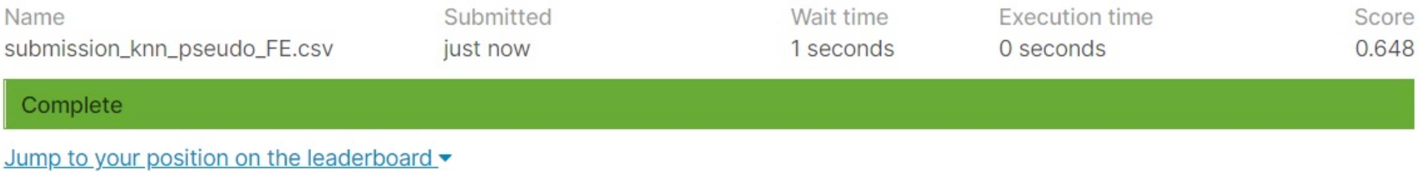
In [120]:

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


y_pred_lr_test.to_csv('submission_knn_pseudo_FE.csv', index=False)
y_pred_lr_test.head(10)
```

Out[120]:

	ID	Target
0	250	0.666667
1	251	0.622222
2	252	0.555556
3	253	0.644444
4	254	0.644444
5	255	0.577778
6	256	0.600000
7	257	0.666667
8	258	0.644444
9	259	0.555556



Name	Submitted	Wait time	Execution time	Score
submission_knn_pseudo_FE.csv	just now	1 seconds	0 seconds	0.648

Complete

Jump to your position on the leaderboard

test_auc = 0.65

Logistic Regresstion Applied

In [160]:

```
#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 [161]:

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

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

In [162]:

```
from sklearn import linear_model

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

model.fit(X_train,y_train)
```

Out[162]:

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

In [163]:

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

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

In [164]:

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

```
0.8545138888888888
```

In [165]:

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

```
[0.7311261  0.6158929  0.64559574 ... 0.45605361 0.84490119 0.31432532]
```

In [166]:

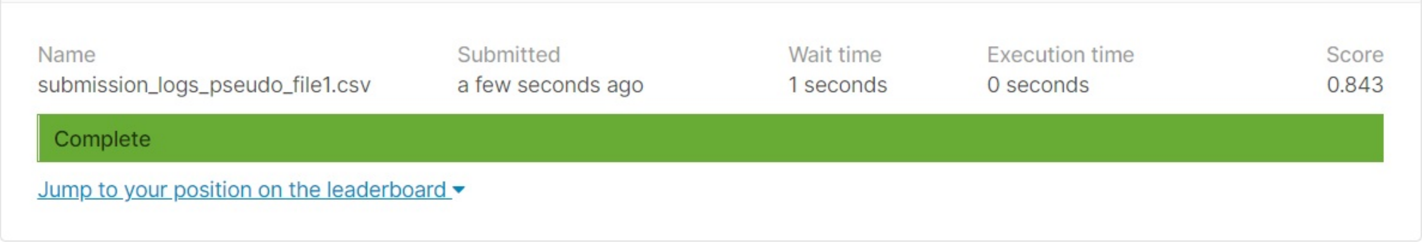
```
y_pred_lr_test = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_lr_test})



y_pred_lr_test.to_csv('submission_logs_pseudo_file1.csv', index=False)
y_pred_lr_test.head(20)
```

Out[166]:

	ID	Target
0	250	0.731126
1	251	0.615893
2	252	0.645596
3	253	0.804024
4	254	0.510489
5	255	0.436458
6	256	0.467490
7	257	0.302518
8	258	0.765161
9	259	0.320234
10	260	0.619605
11	261	0.481176
12	262	0.424977
13	263	0.764846
14	264	0.400445
15	265	0.759117
16	266	0.419489
17	267	0.813255
18	268	0.544783
19	269	0.420502



Name
submission_logs_pseudo_file1.csv

Submitted
a few seconds ago

Wait time
1 seconds

Execution time
0 seconds

Score
0.843

Complete

logistic_test_auc = 0.843

Support Vector Machine

In [167]:

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

In [168]:

```
#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(random_state=42)
#call the hyper-parameter function to get best parameters

svc_clf = hyperparameter_model(svc_model,params)
```

In [169]:

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

```
{'C': 0.001, 'kernel': 'linear'}
```

In [171]:

```
svc_clf = SVC(C = 0.001, kernel = 'linear',probability=True)
svc_clf.fit(X_train,y_train)
```

Out[171]:

```
SVC(C=0.001, kernel='linear', probability=True)
```

In [172]:

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

```
0.5111111111111111
```

In [173]:

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

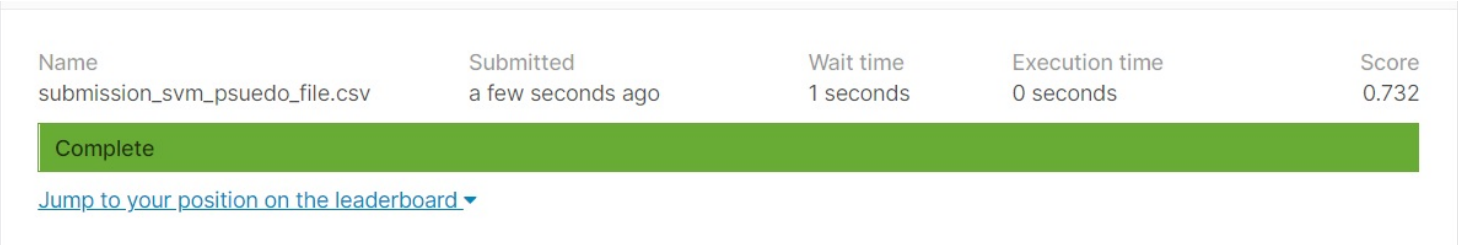
In [174]:

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


y_pred_svc_test.to_csv('submission_svm_psuedo_file.csv', index=False)
y_pred_svc_test.head(20)
```

Out[174]:

	ID	Target
0	250	0.593883
1	251	0.602458
2	252	0.500000
3	253	0.874720
4	254	0.591752
5	255	0.467980
6	256	0.334170
7	257	0.549882
8	258	0.727856
9	259	0.477202
10	260	0.728619
11	261	0.482549
12	262	0.194793
13	263	0.697461
14	264	0.647926
15	265	0.760737
16	266	0.761857
17	267	0.545396
18	268	0.429742
19	269	0.564750



Name	Submitted	Wait time	Execution time	Score
submission_svm_psuedo_file.csv	a few seconds ago	1 seconds	0 seconds	0.732

Test_SVM_auc = 0.73

Ensemble Model : Random Forest

In [175]:

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

In [176]:

```
#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 [177]:

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

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

In [178]:

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

Out[178]:

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

In [179]:

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

```
0.5166666666666666
```

In [180]:

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

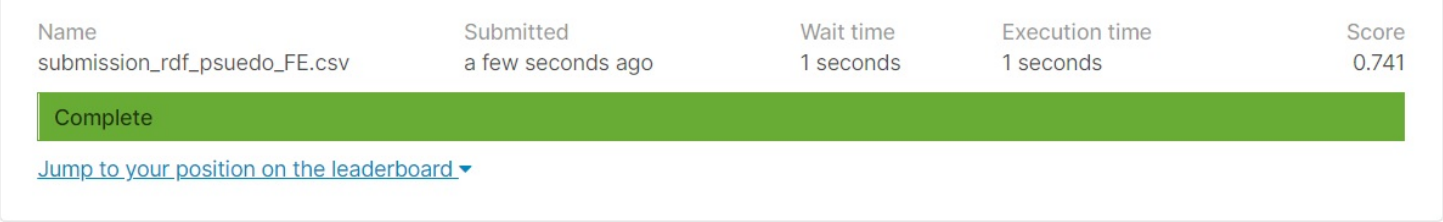
In [181]:

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


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

Out[181]:

	ID	Target
0	250	0.647094
1	251	0.641673
2	252	0.609725
3	253	0.686968
4	254	0.646169
5	255	0.604770
6	256	0.584772
7	257	0.617640
8	258	0.665500
9	259	0.599057
10	260	0.653698
11	261	0.619064
12	262	0.648372
13	263	0.650540
14	264	0.612950
15	265	0.676086
16	266	0.646294
17	267	0.628980
18	268	0.598755
19	269	0.625873



Name	Submitted	Wait time	Execution time	Score
submission_rdf_psuedo_FE.csv	a few seconds ago	1 seconds	1 seconds	0.741

Complete

Jump to your position on the leaderboard

Test_rdf_auc : 0.74

Decision Tree Classifier

In [182]:

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

In [183]:

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

params =  {'max_depth':[2,3,5,7,9],'min_samples_split':[3,5,7,9],'criterion' :['gini', 'entropy']}

#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 [184]:

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

```
{'criterion': 'gini', 'max_depth': 2, 'min_samples_split': 3}
```

In [185]:

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

Out[185]:

```
DecisionTreeClassifier(max_depth=2, min_samples_split=3)
```

In [186]:

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

```
0.6621527777777778
```

In [187]:

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

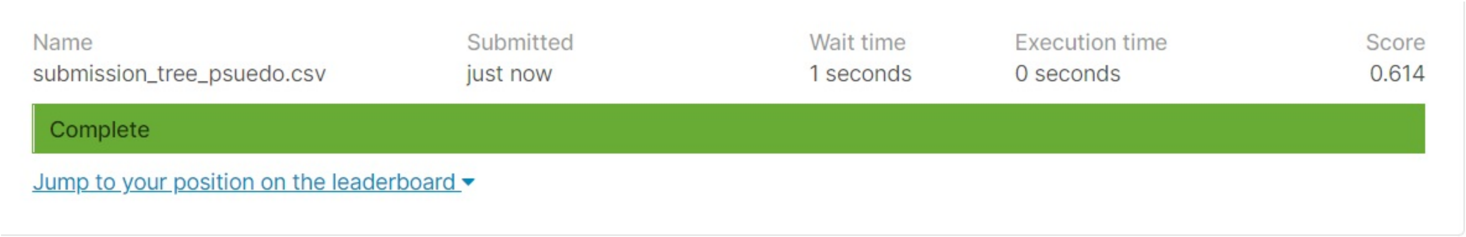
In [188]:

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


y_pred_tree_test.to_csv('submission_tree_psuedo_FE.csv', index=False)
y_pred_tree_test.head(10)
```

Out[188]:

	ID	Target
0	250	0.846774
1	251	0.846774
2	252	0.561798
3	253	0.846774
4	254	0.846774
5	255	0.561798
6	256	0.846774
7	257	0.846774
8	258	0.846774
9	259	0.561798



Name	Submitted	Wait time	Execution time	Score
submission_tree_psuedo.csv	just now	1 seconds	0 seconds	0.614

Complete

Test_auc = 0.614

XGBoost Classifier

In [189]:

```
from xgboost import XGBClassifier
```

In [190]:

```
#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 [191]:

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

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

In [192]:

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

Out[192]:

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

In [193]:

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

In [194]:

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

```
1.0
```

In [195]:

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

```
[0.8545241  0.7210328  0.68291163 ... 0.18938394 0.9828232  0.2910965 ]
```

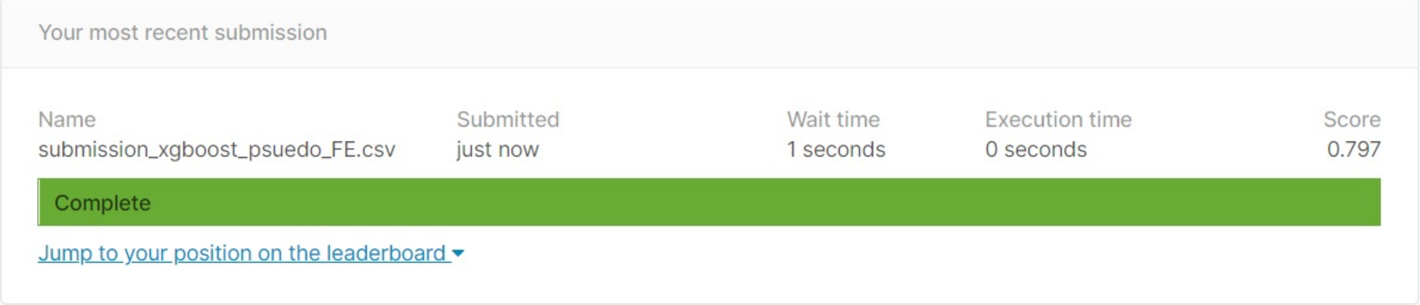
In [196]:

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


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

Out[196]:

	ID	Target
0	250	0.854524
1	251	0.721033
2	252	0.682912
3	253	0.997933
4	254	0.904231
5	255	0.571506
6	256	0.338783
7	257	0.151278
8	258	0.995602
9	259	0.453835



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_xgboost_psuedo_FE.csv	just now	1 seconds	0 seconds	0.797

Complete

Jump to your position on the leaderboard ▾

xgboost_test_auc = 0.797

Stacking Classifier

In [197]:

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

In [199]:

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

In [200]:

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

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

#Classifier 3
svc_clf = SVC(C = 0.001, kernel = 'linear',probability=True)
svc_clf.fit(X_train,y_train)

#classifier 3

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

tree_clf = DecisionTreeClassifier(max_depth = 2,min_samples_split=3,criterion='gini')
tree_clf.fit(X_train,y_train)

#classifier 5
xg_clf = XGBClassifier(max_depth = 2, n_estimators = 500)
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 [201]:

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

```
1.0
```

In [202]:

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

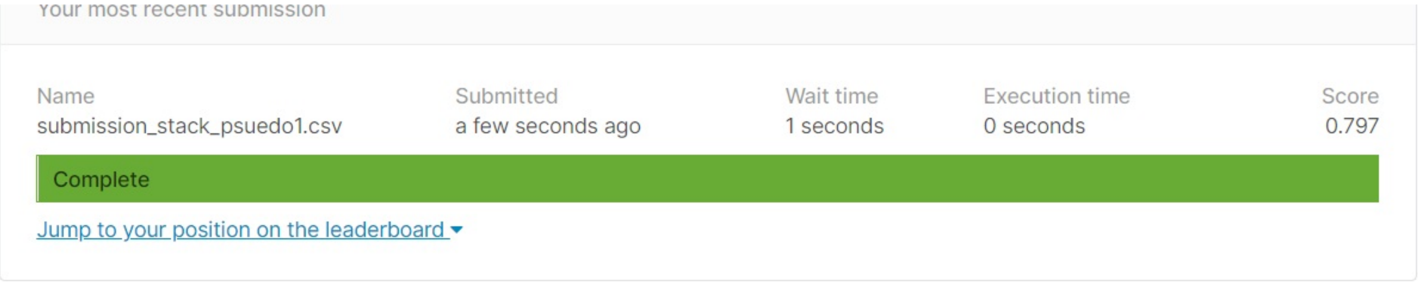
In [203]:

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


y_pred_stack_test.to_csv('submission_stack_psuedo1.csv', index=False)
y_pred_stack_test.head(20)
```

Out[203]:

	ID	Target
0	250	0.882698
1	251	0.797532
2	252	0.766041
3	253	0.937826
4	254	0.905446
5	255	0.656084
6	256	0.381644
7	257	0.199123
8	258	0.937163
9	259	0.518817
10	260	0.868914
11	261	0.788055
12	262	0.166941
13	263	0.933581
14	264	0.676394
15	265	0.935637
16	266	0.320781
17	267	0.919440
18	268	0.701263
19	269	0.900704



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_stack_psuedo1.csv	a few seconds ago	1 seconds	0 seconds	0.797

Test_AUc = 0.797

Summary of All Models

In [1]:

```
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}",0.54,0.65],
           ['logistic Regresstion',r"{'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}",0.85,0.84],['Support Vec
tor Machine',
           r"{'C': 0.001, 'kernel': 'linear'}",0.51,0.73],['XGBoost Classifier',r"{'max_depth': 2, 'n_estimators'
: 500}",1.00,0.80],
            ['Random forest',r"{'max_depth': 2, 'n_estimators': 400}",0.52,0.74], ['DecisionTree',r"{'criterion':
'gini', 'max_depth': 2, 'min_samples_split': 3}",0.66,0.61],
            ['Calibrated Model',"--",1.00,0.80]])

print(t.draw())
```

```
+------------------------+------------------------------+-----------+----------+
|         Model          |       Hyper-parameter        | Train AUC | Test AUC |
+========================+==============================+===========+==========+
| Knn_Model              | {'algorithm': 'kd_tree',     | 0.540     | 0.650    |
|                        | 'n_neighbors': 45}           |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | {'C': 0.1, 'penalty': 'l1',  | 0.850     | 0.840    |
|                        | 'solver': 'liblinear'}       |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.001, 'kernel':       | 0.510     | 0.730    |
|                        | 'linear'}                    |           |          |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 2,             | 1         | 0.800    |
|                        | 'n_estimators': 500}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 2,             | 0.520     | 0.740    |
|                        | 'n_estimators': 400}         |           |          |
+------------------------+------------------------------+-----------+----------+
| DecisionTree           | {'criterion': 'gini',        | 0.660     | 0.610    |
|                        | 'max_depth': 2,              |           |          |
|                        | 'min_samples_split': 3}      |           |          |
+------------------------+------------------------------+-----------+----------+
| Calibrated Model       | --                           | 1         | 0.800    |
+------------------------+------------------------------+-----------+----------+
```

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. 2.To Balance it, We have teken the idea of Pseudo Technique(https://www.analyticsvidhya.com/blog/2017/09/pseudo-labelling-semi-supervised-learning-techn (https://www.analyticsvidhya.com/blog/2017/09/pseudo-labelling-semi-supervised-learning-techn))

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=0.54 and Test Auc = 0.65 . 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 = 0.85 and Test_auc=0.84 which is working as expected.

   3.Support Vector Machine: The SVM algorithm trained the model with parameter(C=0.001,kernel=linear) and got the train_AUC=0.51 and Test_AUC = 0.73.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=0.80.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=2,min_samples_split=3,Criterion:gini) and got the train_AUC = 0.66 and test_AUC=0.61 .Model is comparable less accurate but it is not overfitted.

   7.Calibrated model gave train_AUC = 1.0 and Test_AUC = 0.80. Model is accurate and not overfitted.