Pseudo Technique+Nearest Neighbors+Standardization+Model

1. Import necessary Libraries

In [ ]:

```
#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.over_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 [ ]:

```
#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 [ ]:

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

Out[ ]:

	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 [ ]:

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

In [ ]:

```
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 [ ]:

```
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 [ ]:

```
#ref=https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html
from sklearn.neighbors import NearestNeighbors
from sklearn import linear_model
neigh = NearestNeighbors(n_neighbors=5, n_jobs=-1)
neigh.fit(X_train)

dists, _ = neigh.kneighbors(X_train, n_neighbors=5)
mean_dist = dists.mean(axis=1)
max_dist = dists.max(axis=1)
min_dist = dists.min(axis=1)

X_train['308'] = X_train.std(1)
print(X_train['308'])
X_train = np.hstack((X_train, mean_dist.reshape(-1, 1), max_dist.reshape(-1, 1), min_dist.reshape(-1, 1)))

test_dists, _ = neigh.kneighbors(X_test, n_neighbors=5)

test_mean_dist = test_dists.mean(axis=1)
test_max_dist = test_dists.max(axis=1)
test_min_dist = test_dists.min(axis=1)

X_test['308'] = X_test.std(1)
X_test = np.hstack((X_test, test_mean_dist.reshape(-1, 1), test_max_dist.reshape(-1, 1), test_min_dist.reshape(-1
, 1)))
```

```
0      1.089171
1      0.985838
2      1.012757
3      0.939743
4      0.941277
         ...
245    1.081211
246    0.979557
247    1.042626
248    1.017994
249    0.947306
Name: 308, Length: 250, dtype: float64
```

In [ ]:

```
stand = StandardScaler()
X_train = stand.fit_transform(X_train)
X_test = stand.transform(X_test)
```

In [ ]:

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

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	
0	-0.121736	2.176002	0.503692	-0.609972	1.265232	-0.469388	-0.266814	0.210682	-2.296917	1.758518	-0.837523	-0.2
1	1.061577	-0.939278	-0.539790	0.320974	-0.415729	0.340017	1.134681	0.291718	0.042547	-0.320787	2.689063	0.94
2	-0.548290	-0.061678	-0.505465	0.144689	-0.022827	0.431232	-0.054798	-0.267006	0.180835	0.144993	0.428502	0.35
3	0.043868	0.005829	0.220265	-1.623118	-0.433148	-0.752470	-1.062122	0.805660	0.561388	0.234415	0.313996	-1.2
4	2.332208	-0.798306	0.336970	-0.022683	1.183942	1.678890	0.550393	1.525391	0.025911	2.125050	0.180099	0.33

5 rows x 304 columns

In [ ]:

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

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	
0	0.478452	-0.998843	-1.728417	0.304138	-0.692502	0.533981	0.500624	-0.221157	-0.675928	1.233779	0.472827	-0.413
1	0.755461	0.934060	-0.648649	1.332140	-0.840565	0.510915	0.543426	-0.417350	0.249460	1.297652	-0.998200	0.2047
2	1.733024	0.531992	-0.220077	0.825072	-0.462180	1.504847	-0.729665	-2.225742	-1.394404	0.129271	-0.117246	0.3372
3	-0.581411	-1.814892	-0.833024	0.570547	1.539102	0.544465	-1.444348	0.686238	0.569706	0.522334	0.333388	-0.164
4	0.733381	-0.216549	0.986204	-1.609253	0.007173	0.395585	0.745488	-1.963439	-1.450551	-0.948706	-0.085850	0.7597

5 rows × 304 columns

In [ ]:

```
X_test.shape
```

Out[ ]:

```
(19750, 304)
```

Used GridSearch for hyper-parameter tuning

In [ ]:

```
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 [ ]:

```
#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 [ ]:

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

```
{'algorithm': 'kd_tree', 'n_neighbors': 47}
```

In [ ]:

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

Out[ ]:

```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=47)
```

In [ ]:

```
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 0 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 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 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 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 0 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 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]
```

In [ ]:

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

```
0.5413194444444445
```

In [ ]:

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

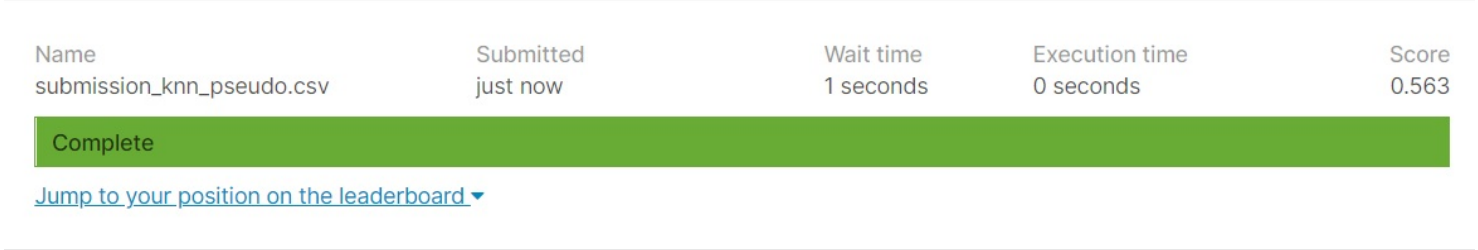
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.617021
1	251	0.595745
2	252	0.595745
3	253	0.638298
4	254	0.595745
5	255	0.617021
6	256	0.574468
7	257	0.638298
8	258	0.595745
9	259	0.595745



test_auc = 0.564

Logistic Regresstion Applied

In [ ]:

```
#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','saga'
]}

#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 [ ]:

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

```
{'C': 1, 'penalty': 'l1', 'solver': 'saga'}
```

In [ ]:

```
from sklearn import linear_model

model = LogisticRegression(penalty='l1', C=1, solver='saga')

model.fit(X_train,y_train)
```

Out[ ]:

```
LogisticRegression(C=1, penalty='l1', solver='saga')
```

In [ ]:

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

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

In [ ]:

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

```
1.0
```

In [ ]:

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

```
[0.63294452 0.34819197 0.84229144 ... 0.52536113 0.9974678  0.32538815]
```

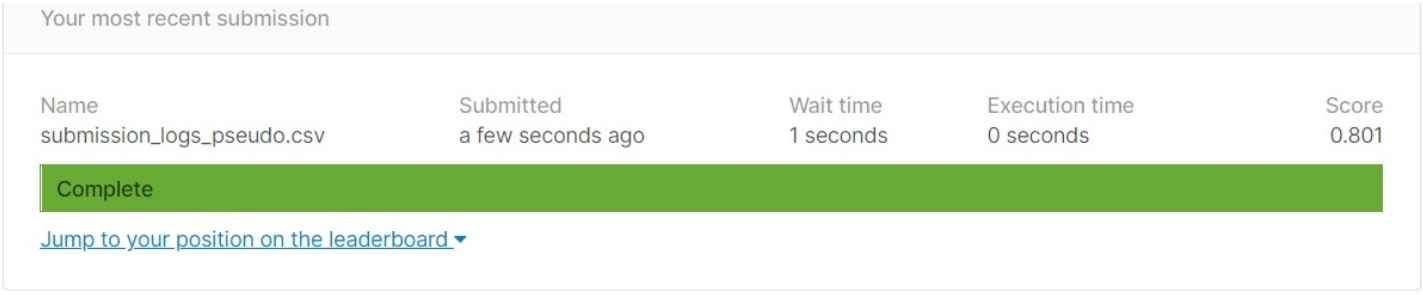
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.632945
1	251	0.348192
2	252	0.842291
3	253	0.999404
4	254	0.887499
5	255	0.269127
6	256	0.141706
7	257	0.388645
8	258	0.971322
9	259	0.352787
10	260	0.447912
11	261	0.167381
12	262	0.053797
13	263	0.914092
14	264	0.568148
15	265	0.965213
16	266	0.935332
17	267	0.833466
18	268	0.323775
19	269	0.937280



logistic_test_auc = 0.80

Support Vector Machine

In [79]:

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

In [80]:

```
#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 [81]:

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

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

In [83]:

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

Out[83]:

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

In [84]:

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

```
0.5
```

In [85]:

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

In [86]:

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


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

Out[86]:

	ID	Target
0	250	0.622945
1	251	0.622945
2	252	0.622945
3	253	0.622945
4	254	0.622945
5	255	0.622945
6	256	0.622945
7	257	0.622945
8	258	0.622945
9	259	0.622945
10	260	0.622945
11	261	0.622945
12	262	0.622945
13	263	0.622945
14	264	0.622945
15	265	0.622945
16	266	0.622945
17	267	0.622945
18	268	0.622945
19	269	0.622945

Name	Submitted	Wait time	Execution time	Score
submission_svm_psuedo.csv	just now	1 seconds	0 seconds	0.500

Complete

Jump to your position on the leaderboard

Test_SVM_auc = 0.50

Ensemble Model : Random Forest

In [93]:

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

In [94]:

```
#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 [95]:

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

```
{'max_depth': 5, 'n_estimators': 300}
```

In [96]:

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

Out[96]:

```
RandomForestClassifier(max_depth=5, n_estimators=300)
```

In [97]:

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

```
1.0
```

In [98]:

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

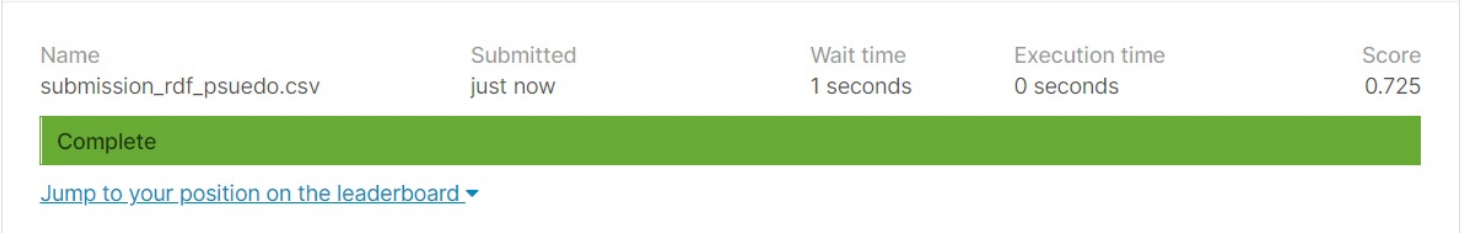
In [99]:

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


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

Out[99]:

	ID	Target
0	250	0.650319
1	251	0.641019
2	252	0.613636
3	253	0.694247
4	254	0.655788
5	255	0.613511
6	256	0.591895
7	257	0.631163
8	258	0.705134
9	259	0.586853
10	260	0.614333
11	261	0.621265
12	262	0.605150
13	263	0.638817
14	264	0.619725
15	265	0.699669
16	266	0.674954
17	267	0.615253
18	268	0.583332
19	269	0.650641



Test_rdf_auc : 0.73

Decision Tree Classifier

In [72]:

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

In [73]:

```
#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]}

#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 [74]:

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

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

In [75]:

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

Out[75]:

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

In [76]:

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

```
0.6621527777777778
```

In [77]:

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

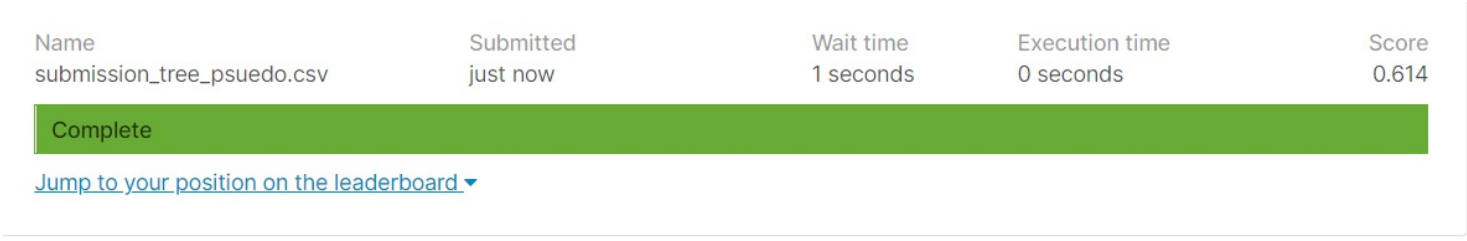
In [78]:

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


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

Out[78]:

	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



Test_auc = 0.614

XGBoost Classifier

In [ ]:

```
from xgboost import XGBClassifier
```

In [ ]:

```
#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 [ ]:

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

```
{'max_depth': 5, 'n_estimators': 500}
```

In [ ]:

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

Out[ ]:

```
XGBClassifier(max_depth=5, n_estimators=500)
```

In [ ]:

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

In [ ]:

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

```
1.0
```

In [ ]:

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

```
[0.83462036 0.894321   0.52436954 ... 0.14851385 0.98524    0.4055095 ]
```

In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.834620
1	251	0.894321
2	252	0.524370
3	253	0.991975
4	254	0.793879
5	255	0.791808
6	256	0.300758
7	257	0.396944
8	258	0.979606
9	259	0.806753

Name	Submitted	Wait time	Execution time	Score
submission_xgboost_psuedo.csv	just now	1 seconds	0 seconds	0.785

Complete

Jump to your position on the leaderboard

xgboost_test_auc = 0.785

Stacking Classifier

In [101]:

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

In [102]:

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

In [103]:

```
#classifier 1
knn_model = KNeighborsClassifier(algorithm ='kd_tree',n_neighbors = 47)
knn_model.fit(X_train,y_train)

#Classifier 2
model = LogisticRegression(penalty='l1', C=1, solver='saga')
model.fit(X_train,y_train)

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

#classifier 3

rdf_clf = RandomForestClassifier(max_depth=5, n_estimators=300)
rdf_clf.fit(X_train,y_train)
#classifier 4

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

#classifier 5
xg_clf = XGBClassifier(max_depth = 5, 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 [104]:

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

```
1.0
```

In [105]:

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

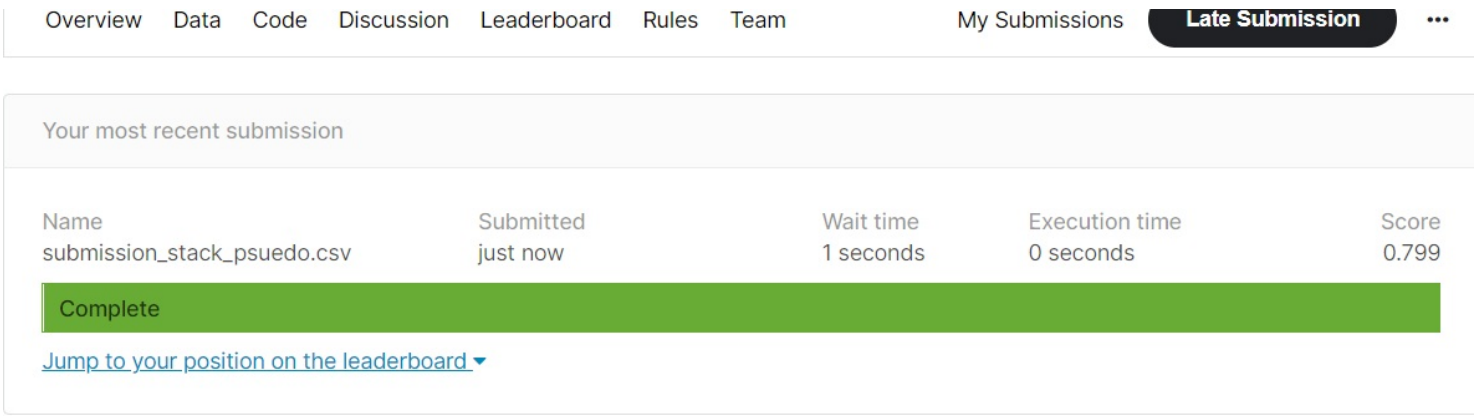
In [106]:

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


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

Out[106]:

	ID	Target
0	250	0.964193
1	251	0.972284
2	252	0.692306
3	253	0.993517
4	254	0.960358
5	255	0.927848
6	256	0.137883
7	257	0.319981
8	258	0.992553
9	259	0.941870
10	260	0.980268
11	261	0.642882
12	262	0.143131
13	263	0.992447
14	264	0.070366
15	265	0.993425
16	266	0.641084
17	267	0.980865
18	268	0.912787
19	269	0.751352



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_neighbors=4
7)",0.54,0.56],
           ['logistic Regresstion',r"(C=1, penalty='l1', solver='saga')",1.0,0.80],['Support Vector Machine',
           r"{'C': 0.001, 'kernel': 'sigmoid'}",0.5,0.5],['XGBoost Classifier',r"{'max_depth': 5, 'n_estimators':
500}",1.00,0.79],
            ['Random forest',r"{'max_depth': 5, 'n_estimators': 300}",1.0,0.73], ['DecisionTree',r"{'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     | 0.560    |
|                        | n_neighbors=47)              |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | (C=1, penalty='l1',          | 1         | 0.800    |
|                        | solver='saga')               |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.001, 'kernel':       | 0.500     | 0.500    |
|                        | 'sigmoid'}                   |           |          |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 5,             | 1         | 0.790    |
|                        | 'n_estimators': 500}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 5,             | 1         | 0.730    |
|                        | 'n_estimators': 300}         |           |          |
+------------------------+------------------------------+-----------+----------+
| DecisionTree           | {'max_depth': 2,             | 0.660     | 0.610    |
|                        | '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. Used the Nearest Neighborhood technique and apply the basic statistic on features.
2. Dropped the labled data from both train and test datasets.
3. Used Standization method to standardize the features.
4. Used GridSerach Validation for hyper-parameter tuning.
5. We have applied following machine learning algorithm: 1.KNN : The KNN algorithm trained the model with parameter(algorithm = 'kd_tree', and n_neighbors=47) and gave train_AUC=0.54 and Test Auc = 0.56 . Model is less accurate but it is not overfitted.

   2.Logistic Regression : The Logistic regression algorithm trained the model with parameter(C=1,penalty=l1,solver=saga) and gave train_AUC = 1.00 and Test_auc=0.80 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=0.50 and Test_AUC = 0.50 which is less accurate but not overfitted model

   4.XGBoost Classifier: The XGBoost classifier trained the model with parameter(max_depth=5,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=5,n_estimators=300) and got the train_AUC = 1.0 and test_AUC = 0.73. Model is not overfitted 6.DecisionTree : The Decision Tree classifier trained the model with parameter(max_depth=2,min_samples_split=3) 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.

XGBoost and Calibrated Model working well from above applied algorithm