Class Weight+FE+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 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 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)
```

Observation

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

In [ ]:

```
def feature_engg(df,test=False):
  '''
  perform feature Engieering in basic statistics, trignometory, hyperbolic and exponential function

  parameters:

  '''
  if test:
    data = df.drop(['id'],axis=1)
  else:
    data = df.drop(['id','target'],axis=1)

  #mean and std
  df['mean'] = np.mean(data,axis=1) # taking mean value along with column
  df['std'] = np.std(data,axis=1) # taking std along with column
  df['median'] = np.median(data,axis=1)
  df['min'] = np.min(data,axis=1)
  df['max'] = np.max(data,axis=1)


  # applying trignometric function
  df['sin_mean'] = np.sin(df['mean'])
  df['cos_mean'] = np.cos(df['mean'])
  df['tan_mean'] = np.tan(df['mean'])
  df['sin_std'] = np.sin(df['std'])
  df['cos_std'] = np.cos(df['std'])
  df['tan_std'] = np.tan(df['std'])

  df['sin_median'] = np.sin(df['median'])
  df['cos_median'] = np.cos(df['median'])
  df['tan_median'] = np.tan(df['median'])
  sin_data = np.sin(data) #calculated the sin_data
  cos_data = np.cos(data) #calculated the cos_data
  tan_data = np.tan(data) #calculated the tan_data

  df['mean_sin'] = np.mean(sin_data,axis=1) #calculating the mean of sin_data
```

```
  df['mean_cos'] = np.mean(cos_data,axis=1) #calculating the mean of cos_data

  df['mean_tan'] = np.mean(tan_data,axis=1) #calculating the mean of tan_data

  #hyperbolic function

  sinh_data = np.sinh(data)
  cosh_data = np.cosh(data)
  tanh_data = np.tanh(data)
  arcsinh_data = np.arcsinh(data)
  arccosh_data = np.arccosh(data)

  df['mean_sinh'] = np.mean(sinh_data,axis=1)
  df['mean_cosh'] = np.mean(cosh_data,axis=1)
  df['mean_tanh'] = np.mean(tanh_data,axis=1)
  df['mean_arsinh'] = np.mean(arcsinh_data,axis=1)
  df['mean_arcosh'] = np.mean(arccosh_data,axis=1)
  df['sinh_mean'] = np.sinh(df['mean'])

  df['tanh_mean'] = np.tanh(df['mean'])
  df['arsinh_mean'] = np.arcsinh(df['mean'])
  df['sinh_std'] = np.sinh(df['std'])
  df['cosh_std'] = np.cosh(df['std'])
  df['tanh_std'] = np.tanh(df['std'])
  df['sinh_median'] = np.sinh(df['median'])
  df['cosh_median'] = np.cosh(df['median'])
  df['tanh_median'] = np.tanh(df['median'])

#exponential function


  exp_data = np.exp(data)
  expm1_data = np.expm1(data)
  exp2_data = np.exp2(data)

  df['mean_exp'] = np.mean(exp_data,axis=1)
  df['mean_expm1'] = np.mean(expm1_data,axis=1)
  df['mean_exp2'] = np.mean(exp2_data,axis=1)
  df['exp1_mean'] = np.exp(df['mean'])
  df['expm1_mean'] = np.expm1(df['mean'])
  df['exp2_mean'] = np.exp2(df['mean'])
  df['exp1_median'] = np.exp(df['median'])
  df['expm1_median'] = np.expm1(df['median'])
  df['exp2_median'] = np.exp2(df['median'])

  df['exp1_std'] = np.exp(df['std'])
  df['expm1_std'] = np.expm1(df['std'])
  df['exp2_std'] = np.exp2(df['std'])
  # Polynomial FE
  # X**2
  df['mean_x2'] = np.mean(np.power(data,2), axis=1)
  # X**3
  df['mean_x3'] = np.mean(np.power(data,3), axis=1)
  # X**4
  df['mean_x4'] = np.mean(np.power(data,4), axis=1)
  # X**5
  df['mean_x5'] = np.mean(np.power(data,5), axis=1)
  # X**6
  df['mean_x6'] = np.mean(np.power(data,6), axis=1)
  # X**7
  df['mean_x7'] = np.mean(np.power(data,7), axis=1)
  #logithm FE
  df['x2_mean'] = np.power(df['mean'],2)
  # X**3
  df['x3_mean'] = np.power(df['mean'],3)
  # X**4
  df['x4_mean'] = np.power(df['mean'],4)
  # X**5
  df['x5_mean'] = np.power(df['mean'],5)
  # X**6
  df['x6_mean'] = np.power(df['mean'],6)
  # X**7
  df['x7_mean'] = np.power(df['mean'],7)
  #skewness and kurtosis
  skew_data = skew(data)
  kurtosis_data = kurtosis(data)
  df['skewness'] = np.mean(skew_data)

  df['kurtosis'] = np.mean(kurtosis_data)
  data['mean_skewness'] = skew(df['mean'])
  data['mean_kurtosis'] = kurtosis(df['mean'])
  df['x2_median'] = np.power(df['median'],2)
  # X**3
  df['x3_median'] = np.power(df['median'],3)
```

```
    # X**4

    df['x4_median'] = np.power(df['median'],4)
    # X**5
    df['x5_median'] = np.power(df['median'],5)
    # X**6
    df['x6_median'] = np.power(df['median'],6)
    # X**7
    df['x7_median'] = np.power(df['median'],7)


    return df
```

In [ ]:

```
df_train = feature_engg(df_train)
df_train.head(5)
```

Out[ ]:

	id	target	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	0	1.0	-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	0.4
1	1	0.0	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	-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	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	1.0	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 × 365 columns

In [ ]:

```
df_test = feature_engg(df_test,True)
df_test.head(5)
```

Out[ ]:

	id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
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	-0.051	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	0.385	1.
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	-0.460	-0
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	0.080	-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	-0.661	0.

5 rows × 364 columns

Observation

We applied feature engineering on the training dataset and came up with new features

In [ ]:

```
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 [ ]:

```
X_train.shape
```

Out[ ]:

```
(250, 363)
```

Observation

We have dropped the labled data from both training and test dataset

In [ ]:

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

In [ ]:

```
X_train.shape
```

Out[ ]:

```
(250, 363)
```

In [ ]:

```
X_train.shape
```

Out[ ]:

```
(250, 363)
```

Observation

We have used StandardScaler() method to standardize the both training and test dataset.

Used GridSerach 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
# 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': 49}
```

In [ ]:

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

Out[ ]:

```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=49)
```

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. 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. 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. 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. 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)
```

```
0.5302083333333333
```

In [ ]:

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

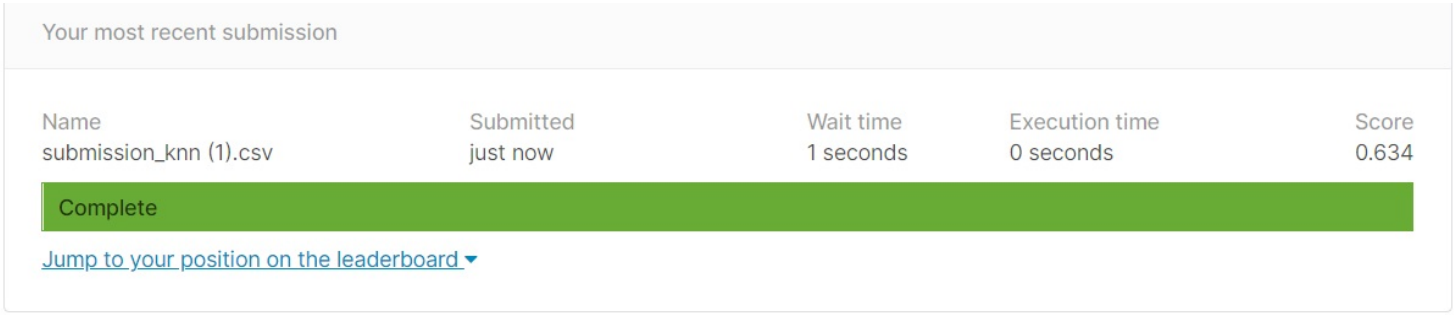
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.551020
1	251	0.653061
2	252	0.591837
3	253	0.632653
4	254	0.571429
5	255	0.571429
6	256	0.612245
7	257	0.755102
8	258	0.693878
9	259	0.632653
10	260	0.673469
11	261	0.591837
12	262	0.387755
13	263	0.571429
14	264	0.653061
15	265	0.571429
16	266	0.673469
17	267	0.673469
18	268	0.551020
19	269	0.551020



test_auc = 0.634

Observation

As the first model, We have used KNN algorithm to trained the model with best parameter (algorithm='kd_tree',n_neighbours = 49) and got training AUC = 0.53 and Test_AUC = 0.634. It is less accurate but model is not overfitted.

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(class_weight='balanced',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. 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. 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. 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. 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)
```

```
[0.05845677 0.30684152 0.84719694 ... 0.74726676 0.98934884 0.25190355]
```

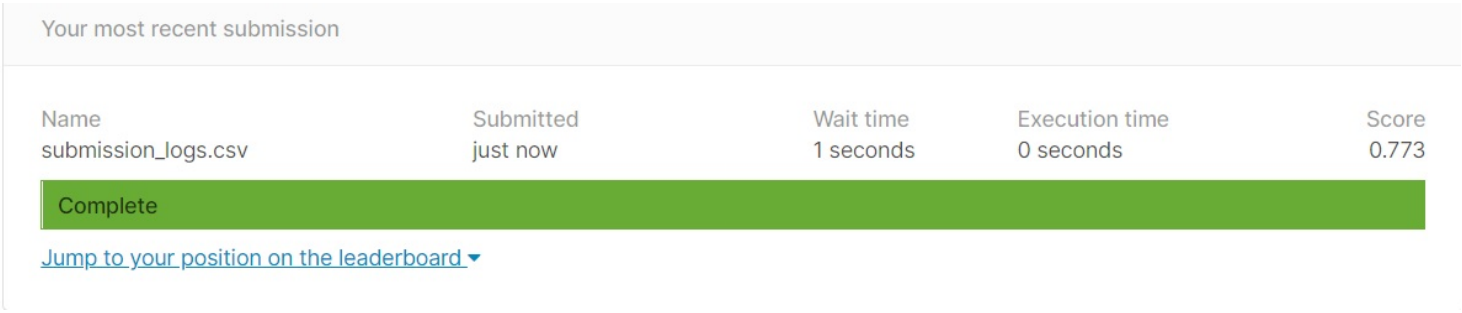
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.058457
1	251	0.306842
2	252	0.847197
3	253	0.995235
4	254	0.719528
5	255	0.368864
6	256	0.182455
7	257	0.530183
8	258	0.970734
9	259	0.284543
10	260	0.597124
11	261	0.219657
12	262	0.074346
13	263	0.868023
14	264	0.594500
15	265	0.933693
16	266	0.931524
17	267	0.708268
18	268	0.348039
19	269	0.781881



logistic_test_auc = 0.773

Observation

We have used Logistic Regression algorithm to trained the model with best parameter (c=1,penalty=l1,solver=saga) and got training AUC = 1.0 and Test_AUC = 0.773. Training AUC is good and test auc litte bit down compare to training AUC.But is decent.

Support Vector Machine

In [ ]:

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

In [ ]:

```
#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 [ ]:

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

```
{'C': 0.1, 'kernel': 'linear'}
```

In [ ]:

```
svc_clf = SVC(C = 0.1, kernel = 'linear',probability=True)
svc_clf.fit(X_train,y_train)
```

Out[ ]:

```
SVC(C=0.1, kernel='linear', probability=True)
```

In [ ]:

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

```
1.0
```

In [ ]:

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

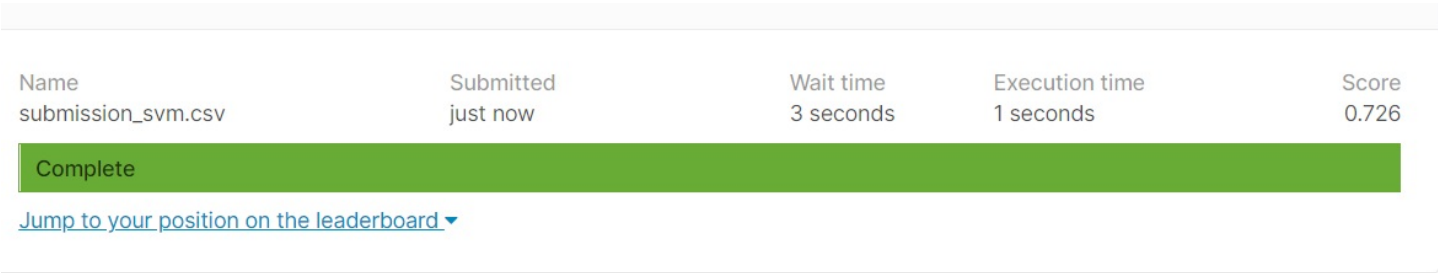
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.280172
1	251	0.333487
2	252	0.635371
3	253	0.934723
4	254	0.509205
5	255	0.614908
6	256	0.391440
7	257	0.725442
8	258	0.851081
9	259	0.469446
10	260	0.661825
11	261	0.434793
12	262	0.461862
13	263	0.760738
14	264	0.654198
15	265	0.673483
16	266	0.767051
17	267	0.485905
18	268	0.360502
19	269	0.646387



Test_SVM_auc = 0.726

Observation

We have used Support Vector algorithm to trained the model with best parameter ('C': 0.1, 'kernel': 'linear') and got training AUC = 1.0 and Test_AUC = 0.73. Training AUC is good and test auc litte bit down compare to training AUC.But it is decent.

Ensemble Model : Random Forest

In [ ]:

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

In [ ]:

```
#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(class_weight='balanced',random_state=42)

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

In [ ]:

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

```
{'max_depth': 9, 'n_estimators': 200}
```

In [ ]:

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

Out[ ]:

```
RandomForestClassifier(max_depth=9, n_estimators=200)
```

In [ ]:

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

```
1.0
```

In [ ]:

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

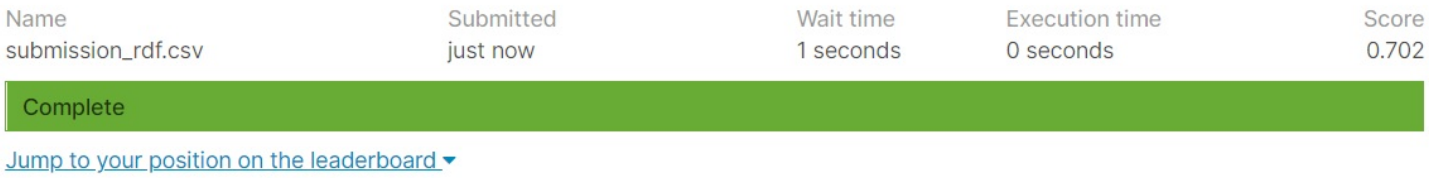
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.459459
1	251	0.652641
2	252	0.660377
3	253	0.740546
4	254	0.511131
5	255	0.601763
6	256	0.543740
7	257	0.716401
8	258	0.761362
9	259	0.608635
10	260	0.606904
11	261	0.563769
12	262	0.557147
13	263	0.595823
14	264	0.555836
15	265	0.750869
16	266	0.588763
17	267	0.635116
18	268	0.568679
19	269	0.557250



Test_rdf_auc : 0.702

Observation

We have used Random Forest algorithm to trained the model with best parameter ('max_depth': 9, 'n_estimators': 200) and got training AUC = 1.0 and Test_AUC = 0.70. Training AUC is good and test auc litte bit down compare to training AUC.But it is decent.

Decision Tree Classifier

In [ ]:

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

In [ ]:

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

params =  {'max_depth':[2,3,5,7,9]}

#The instance of Decision Tree Classifier

tree_model = DecisionTreeClassifier(class_weight='balanced',random_state=42)

#Call Hyperparameter function to get best parameter

tree_clf = hyperparameter_model(tree_model,params)
```

In [ ]:

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

```
{'max_depth': 2}
```

In [ ]:

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

Out[ ]:

```
DecisionTreeClassifier(max_depth=2)
```

In [ ]:

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

```
0.6621527777777778
```

In [ ]:

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

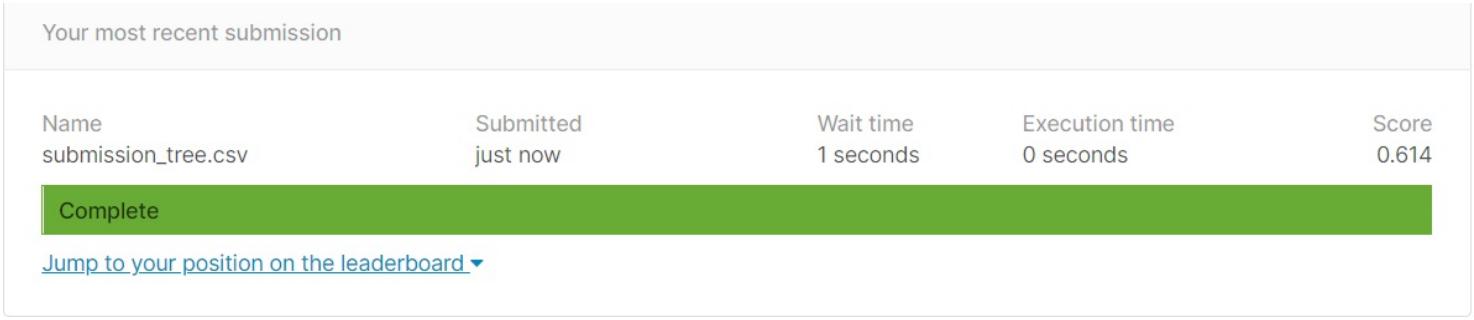
In [ ]:

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


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

Out[ ]:

	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
10	260	0.142857
11	261	0.561798
12	262	0.846774
13	263	0.561798
14	264	0.561798
15	265	0.846774
16	266	0.846774
17	267	0.561798
18	268	0.133333
19	269	0.133333



Observation

We have used Decision Tree algorithm to trained the model with best parameter (max_depth=2) and got training AUC = 0.66 and Test_AUC = 0.61. Model is not overfitted.

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(scale_pos_weight=0.5)
# call hyparameter function to get best parameter

xg_clf = hyperparameter_model(xg_model,params)
```

In [ ]:

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

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

In [ ]:

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

Out[ ]:

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

```
[0.6067806  0.8061706  0.5926346  ... 0.23538436 0.98913246 0.20229056]
```

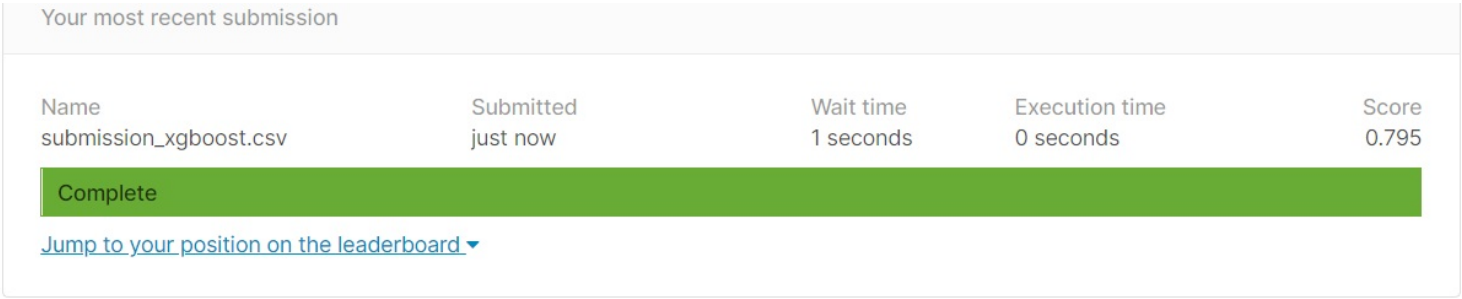
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.606781
1	251	0.806171
2	252	0.592635
3	253	0.998114
4	254	0.786182
5	255	0.507155
6	256	0.293211
7	257	0.383449
8	258	0.996120
9	259	0.372154



xgboost_test_auc = 0.795

Observation

We have used Xgboost algorithm to trained the model with best parameter (max_depth': 2, 'n_estimators': 500) and got training AUC = 1.0 and Test_AUC = 0.80. Model is accurate and it is not overfitted.

Stacking Classifier

In [ ]:

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

In [ ]:

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

In [ ]:

```
#classifier 1
knn_model = KNeighborsClassifier(algorithm ='kd_tree',n_neighbors = 49)
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 = 0.1, kernel = 'linear',probability=True)
svc_clf.fit(X_train,y_train)

#classifier 3

rdf_clf = RandomForestClassifier(max_depth=9, n_estimators=200)
rdf_clf.fit(X_train,y_train)
#classifier 4

tree_clf = DecisionTreeClassifier(max_depth = 2)
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 [137]:

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

```
1.0
```

In [138]:

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

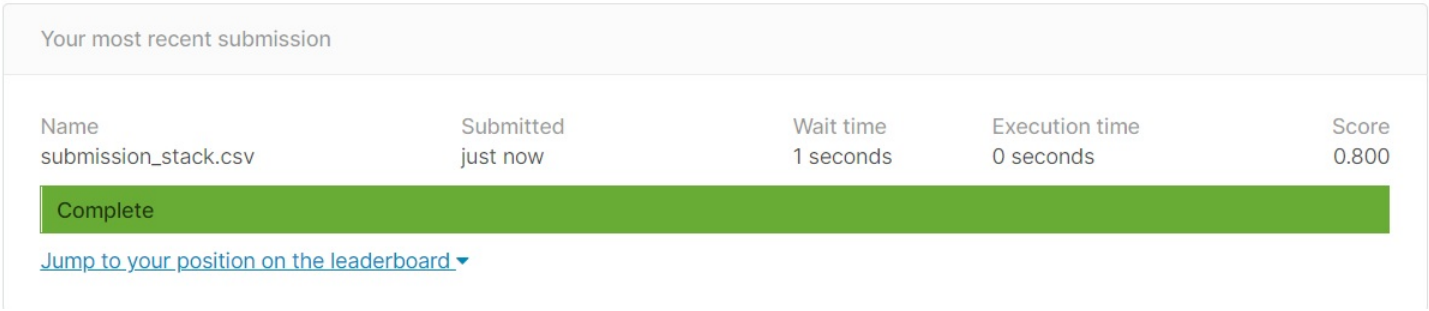
In [139]:

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


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

Out[139]:

	ID	Target
0	250	0.731490
1	251	0.950876
2	252	0.784372
3	253	0.993796
4	254	0.951728
5	255	0.563323
6	256	0.142108
7	257	0.308055
8	258	0.993594
9	259	0.264156
10	260	0.961857
11	261	0.802760
12	262	0.067171
13	263	0.991998
14	264	0.684720
15	265	0.993270
16	266	0.302832
17	267	0.991602
18	268	0.826895
19	269	0.976837



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': 49}",0.53,0.634],
           ['logistic Regresstion',r"{'C': 1, 'penalty': 'l1', 'solver': 'saga'}",1.0,0.773],['Support Vector Mac
hine',
           r"{'C': 0.1, 'kernel': 'linear'}",1.00,0.72],['XGBoost Classifier',r"{'max_depth': 2, 'n_estimators':
500}",1.00,0.80],
            ['Decision_tree Model',r"{'max_depth': 2}",0.66,0.61],['Random forest',r"{'max_depth': 9, 'n_estimato
rs': 200}",1.00,0.70],
            ['Calibrated Model',"--",1.00,0.800]])

print(t.draw())
```

```
+------------------------+------------------------------+-----------+----------+
|         Model          |       Hyper-parameter        | Train AUC | Test AUC |
+========================+==============================+===========+==========+
| Knn_Model              | {'algorithm': 'kd_tree',     | 0.530     | 0.634    |
|                        | 'n_neighbors': 49}           |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | {'C': 1, 'penalty': 'l1',    | 1         | 0.773    |
|                        | 'solver': 'saga'}            |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.1, 'kernel':         | 1         | 0.720    |
|                        | 'linear'}                    |           |          |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 2,             | 1         | 0.800    |
|                        | 'n_estimators': 500}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Decision_tree Model    | {'max_depth': 2}             | 0.660     | 0.610    |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 9,             | 1         | 0.700    |
|                        | 'n_estimators': 200}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Calibrated Model       | --                           | 1         | 0.800    |
+------------------------+------------------------------+-----------+----------+
```

Observation

We have used Calibrated Classifier and got training AUC = 1.0 and test AUC = 0.80. Model is accurate and it is not overfitted.

XGboost and calibrated model is working very well from above applied algorithm