Over Sampling+FE+Standardization+Model

1. Import necessary Libraries

In [38]:

```
#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 [39]:

```
#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 [40]:

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

Out[40]:

	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 [41]:

```
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 [42]:

```
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 [43]:

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

Out[43]:

	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 [44]:

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

Out[44]:

	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 [45]:

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

y_train = df_train['target']

smote = SMOTE()
X_train,y_train = smote.fit_resample(X_train,y_train)
```

In [46]:

```
X_train.shape
```

Out[46]:

```
(320, 363)
```

Observation

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

In [47]:

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

In [48]:

```
X_train.shape
```

Out[48]:

```
(320, 363)
```

In [49]:

```
X_train.shape
```

Out[49]:

```
(320, 363)
```

Observation

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

Used GridSearch for hyper-parameter tuning

In [50]:

```
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 [14]:

```
#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 [16]:

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

```
{'algorithm': 'kd_tree', 'n_neighbors': 39}
```

In [17]:

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

Out[17]:

```
KNeighborsClassifier(algorithm='kd_tree', n_neighbors=39)
```

In [18]:

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

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

In [19]:

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

```
0.503125
```

In [20]:

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

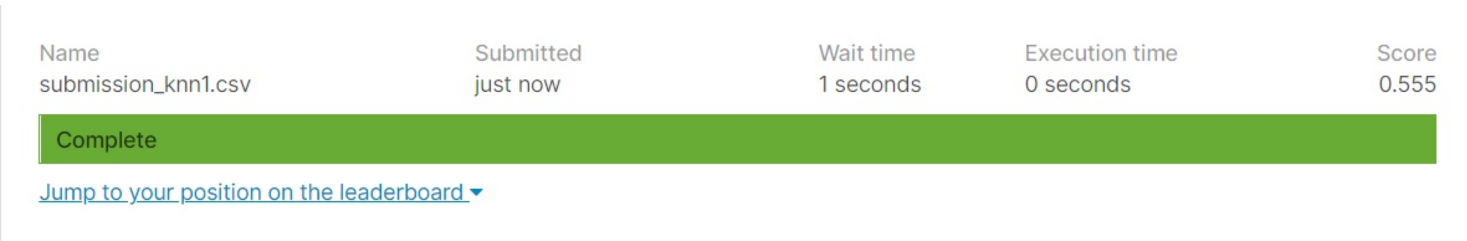
In [21]:

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


y_pred_lr_test.to_csv('submission_knn1.csv', index=False)
y_pred_lr_test.head(20)
```

Out[21]:

	ID	Target
0	250	0.333333
1	251	0.051282
2	252	0.051282
3	253	0.025641
4	254	0.128205
5	255	0.051282
6	256	0.025641
7	257	0.025641
8	258	0.102564
9	259	0.102564
10	260	0.076923
11	261	0.128205
12	262	0.025641
13	263	0.102564
14	264	0.051282
15	265	0.076923
16	266	0.128205
17	267	0.076923
18	268	0.025641
19	269	0.076923



test_auc = 0.555

Observation

As the first model, We have used KNN algorithm to trained the model with best parameter (algorithm='kd_tree',n_neighbours = 39) and got training AUC = 0.50 and Test_AUC = 0.55. It is less accurate but model is not overfitted.

Logistic Regresstion Applied

In [51]:

```
#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 [52]:

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

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

In [53]:

```
from sklearn import linear_model

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

model.fit(X_train,y_train)
```

Out[53]:

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

In [54]:

```
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. 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. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [55]:

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

```
1.0
```

In [56]:

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

```
[0.0296196  0.33341024 0.86344733 ... 0.87014075 0.99319125 0.17739898]
```

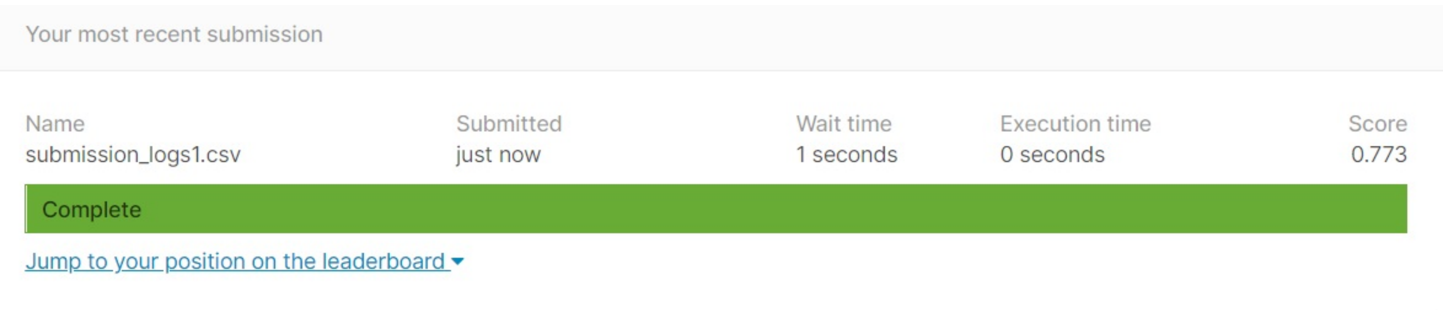
In [57]:

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


y_pred_lr_test.to_csv('submission_logs1.csv', index=False)
y_pred_lr_test.head(20)
```

Out[57]:

	ID	Target
0	250	0.029620
1	251	0.333410
2	252	0.863447
3	253	0.997246
4	254	0.644034
5	255	0.386384
6	256	0.197760
7	257	0.420377
8	258	0.975555
9	259	0.187157
10	260	0.601244
11	261	0.171481
12	262	0.039011
13	263	0.879241
14	264	0.529214
15	265	0.940078
16	266	0.966850
17	267	0.685696
18	268	0.309618
19	269	0.760107



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 it is decent.

Support Vector Machine

In [58]:

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

In [60]:

```
#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 [61]:

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

```
{'C': 1, 'kernel': 'poly'}
```

In [62]:

```
svc_clf = SVC(C = 1, kernel = 'poly',probability=True)
svc_clf.fit(X_train,y_train)
```

Out[62]:

```
SVC(C=1, kernel='poly', probability=True)
```

In [63]:

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

```
1.0
```

In [64]:

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

In [65]:

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


y_pred_svc_test.to_csv('submission_svm1.csv', index=False)
y_pred_svc_test.head(20)
```

Out[65]:

	ID	Target
0	250	0.010239
1	251	0.799375
2	252	0.782953
3	253	0.776982
4	254	0.719545
5	255	0.762792
6	256	0.761592
7	257	0.770146
8	258	0.851095
9	259	0.745677
10	260	0.765826
11	261	0.558469
12	262	0.722451
13	263	0.758217
14	264	0.755097
15	265	0.784517
16	266	0.867675
17	267	0.792677
18	268	0.729581
19	269	0.715112

Name	Submitted	Wait time	Execution time	Score
submission_svm1.csv	just now	1 seconds	0 seconds	0.644

Complete

Jump to your position on the leaderboard

Test_SVM_auc = 0.644

Observation

We have used Support Vector algorithm to trained the model with best parameter ('C': 1, 'kernel': 'Poly') and got training AUC = 1.0 and Test_AUC = 0.64. Training AUC is good and test auc a bit down compare to training AUC.But it is decent.

Ensemble Model : Random Forest

In [66]:

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

In [67]:

```
#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 [68]:

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

```
{'max_depth': 5, 'n_estimators': 400}
```

In [69]:

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

Out[69]:

```
RandomForestClassifier(max_depth=5, n_estimators=400)
```

In [70]:

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

```
1.0
```

In [71]:

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

In [72]:

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


y_pred_rdf_test.to_csv('submission_rdf1.csv', index=False)
y_pred_rdf_test.head(20)
```

Out[72]:

	ID	Target
0	250	0.516006
1	251	0.568699
2	252	0.575418
3	253	0.579637
4	254	0.535900
5	255	0.507414
6	256	0.512040
7	257	0.527653
8	258	0.630538
9	259	0.463896
10	260	0.604303
11	261	0.430581
12	262	0.465354
13	263	0.519210
14	264	0.471482
15	265	0.606225
16	266	0.621462
17	267	0.566494
18	268	0.514597
19	269	0.463538

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_rdf1.csv	just now	1 seconds	1 seconds	0.715

Complete

Jump to your position on the leaderboard

Test_rdf_auc : 0.715

Observation

We have used Random Forest algorithm to trained the model with best parameter ('max_depth': 5, 'n_estimators': 400) and got training AUC = 1.0 and Test_AUC = 0.72. Model is accurate and it is not overfitted.

Decision Tree Classifier

In [95]:

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

In [96]:

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

params =  {'max_depth':[2,3,5,7,9],'criterion':['gini','entropy'],'min_samples_split':[2,3,4,5,6,]}

#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 [97]:

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

```
{'criterion': 'gini', 'max_depth': 9, 'min_samples_split': 6}
```

In [98]:

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

Out[98]:

```
DecisionTreeClassifier(max_depth=9, min_samples_split=6)
```

In [99]:

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

```
0.99375
```

In [100]:

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

In [101]:

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


y_pred_tree_test.to_csv('submission_tree1.csv', index=False)
y_pred_tree_test.head(20)
```

Out[101]:

	ID	Target
0	250	1.000000
1	251	0.000000
2	252	0.000000
3	253	1.000000
4	254	0.000000
5	255	0.000000
6	256	1.000000
7	257	1.000000
8	258	1.000000
9	259	0.000000
10	260	0.000000
11	261	0.000000
12	262	0.333333
13	263	1.000000
14	264	0.000000
15	265	1.000000
16	266	1.000000
17	267	1.000000
18	268	0.000000
19	269	1.000000

Name	Submitted	Wait time	Execution time	Score
submission_tree1.csv	just now	1 seconds	0 seconds	0.599

Complete

Jump to your position on the leaderboard

Observation

Decision Tree is working poorly in this case

XGBoost Classifier

In [102]:

```
from xgboost import XGBClassifier
```

In [103]:

```
#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()
# call hyparameter function to get best parameter

xg_clf = hyperparameter_model(xg_model,params)
```

In [104]:

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

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

In [105]:

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

Out[105]:

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

In [107]:

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

In [108]:

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

```
1.0
```

In [109]:

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

```
[0.7702675  0.4989509  0.2832494  ... 0.35014024 0.9550946  0.3850446 ]
```

In [110]:

```
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[110]:

	ID	Target
0	250	0.770267
1	251	0.498951
2	252	0.283249
3	253	0.995090
4	254	0.698269
5	255	0.321344
6	256	0.348547
7	257	0.270397
8	258	0.997484
9	259	0.552730

Name	Submitted	Wait time	Execution time	Score
submission_xgboost1.csv	just now	1 seconds	0 seconds	0.784

Complete

Jump to your position on the leaderboard

xgboost_test_auc = 0.784

Observation

We have used Xgboost algorithm to trained the model with best parameter ('max_depth': 2, 'n_estimators': 400) and got training AUC = 1.0 and Test_AUC = 0.78. Model is accurate and it is not overfitted.

Stacking Classifier

In [111]:

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

In [112]:

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

In [113]:

```
#classifier 1
knn_model = KNeighborsClassifier(algorithm ='kd_tree',n_neighbors = 39)
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 =1, kernel = 'poly',probability=True)
svc_clf.fit(X_train,y_train)

#classifier 3

rdf_clf = RandomForestClassifier(max_depth=5, n_estimators=400)
rdf_clf.fit(X_train,y_train)
#classifier 4

tree_clf = DecisionTreeClassifier(criterion = 'gini', max_depth = 9, min_samples_split = 6)
tree_clf.fit(X_train,y_train)

#classifier 5
xg_clf = XGBClassifier(max_depth = 2, n_estimators = 400)
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 [114]:

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

1.0

In [115]:

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

In [116]:

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


y_pred_stack_test.to_csv('submission_stack1.csv', index=False)
y_pred_stack_test.head(20)
```

Out[116]:

	ID	Target
0	250	0.384830
1	251	0.664552
2	252	0.444153
3	253	0.984780
4	254	0.739978
5	255	0.819734
6	256	0.833805
7	257	0.795113
8	258	0.989374
9	259	0.650674
10	260	0.886006
11	261	0.119933
12	262	0.340979
13	263	0.982276
14	264	0.264743
15	265	0.984879
16	266	0.712580
17	267	0.982666
18	268	0.381315
19	269	0.941461

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission_stack1.csv	just now	1 seconds	0 seconds	0.759

Complete

Jump to your position on the leaderboard

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_neighbors=3
9)",0.50,0.55],
           ['logistic Regresstion',r"{'C': 1, 'penalty': 'l1', 'solver': 'saga'}",1.0,0.77],['Support Vector Mach
ine',
           r"{'C': 1, 'kernel': 'poly'}",1.0,0.64],['XGBoost Classifier',r"{'max_depth': 2, 'n_estimators': 400}"
,1.00,0.78],
            ['Decision_tree Model',r"{'criterion': 'gini', 'max_depth': 9, 'min_samples_split': 6}",0.99,0.59],['
Random forest',r"{'max_depth': 5, 'n_estimators': 400}",1.00,0.71],
            ['Calibrated Model',"--",1.00,0.76]])

print(t.draw())
```

```
+------------------------+------------------------------+-----------+----------+
|         Model          |       Hyper-parameter        | Train AUC | Test AUC |
+========================+==============================+===========+==========+
| Knn_Model              | (algorithm='kd_tree',        | 0.500     | 0.550    |
|                        | n_neighbors=39)              |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | {'C': 1, 'penalty': 'l1',    | 1         | 0.770    |
|                        | 'solver': 'saga'}            |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 1, 'kernel': 'poly'}   | 1         | 0.640    |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 2,             | 1         | 0.780    |
|                        | 'n_estimators': 400}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Decision_tree Model    | {'criterion': 'gini',        | 0.990     | 0.590    |
|                        | 'max_depth': 9,              |           |          |
|                        | 'min_samples_split': 6}      |           |          |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 5,             | 1         | 0.710    |
|                        | 'n_estimators': 400}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Calibrated Model       | --                           | 1         | 0.760    |
+------------------------+------------------------------+-----------+----------+
```

We have used Calibrated Classifier and got training AUC = 1.0 and test AUC = 0.76

As per observation, XGboost is giving Good Accuracy from above applied algorithm