Over Sampling+FE+Forward_Feature_Selection+Standardization+Model

1. Import necessary Libraries

In [4]:

```
#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 [5]:

```
#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 [6]:

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

Out[6]:

	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 [7]:

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

In [8]:

```
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)
```

```
  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)
```

```python
    return df
```

In [9]:

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

Out[9]:

	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 [10]:

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

Out[10]:

	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

In [11]:

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

y_train = df_train['target']
```

In [12]:

```python
X_train.shape
```

Out[12]:

```
(250, 363)
```

In [13]:

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

In [14]:

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

Out[14]:

	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	-0.210
1	1.061577	-0.939278	-0.539790	0.320974	-0.415729	0.340017	1.134681	0.291718	0.042547	-0.320787	2.689063	0.942
2	-0.548290	-0.061678	-0.505465	0.144689	-0.022827	0.431232	-0.054798	-0.267006	0.180835	0.144993	0.428502	0.355
3	0.043868	0.005829	0.220265	-1.623118	-0.433148	-0.752470	-1.062122	0.805660	0.561388	0.234415	0.313996	-1.216
4	2.332208	-0.798306	0.336970	-0.022683	1.183942	1.678890	0.550393	1.525391	0.025911	2.125050	0.180099	0.331

5 rows × 363 columns

In [15]:

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

Out[15]:

	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	-0.41
1	0.755461	0.934060	-0.648649	1.332140	-0.840565	0.510915	0.543426	-0.417350	0.249460	1.297652	-0.998200	0.20
2	1.733024	0.531992	-0.220077	0.825072	-0.462180	1.504847	-0.729665	-2.225742	-1.394404	0.129271	-0.117246	0.33
3	-0.581411	-1.814892	-0.833024	0.570547	1.539102	0.544465	-1.444348	0.686238	0.569706	0.522334	0.333388	-0.16
4	0.733381	-0.216549	0.986204	-1.609253	0.007173	0.395585	0.745488	-1.963439	-1.450551	-0.948706	-0.085850	0.75

5 rows × 363 columns

In [16]:

```
X_train.shape
```

Out[16]:

(250, 363)

In [17]:

```
X_train.shape
```

Out[17]:

(250, 363)

I will do hyper-parameter tuning

In [18]:

```
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 [19]:

```
#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 [51]:

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

```
{'algorithm': 'kd_tree', 'n_neighbors': 49}
```

In [52]:

```
#ref https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-ho
w-to-select-the-right-variables/#:~:text=Forward%20Selection%3A%20Forward%20selection%20is,the%20performance%20of
%20the%20model.
knn= KNeighborsClassifier()
selector = SequentialFeatureSelector(knn,k_features=(10, 50),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

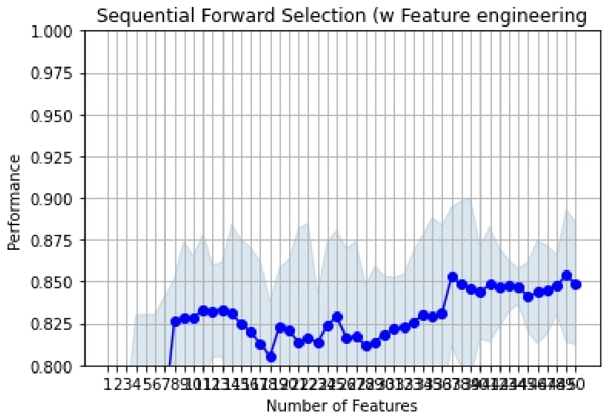
Out[52]:

```
SequentialFeatureSelector(estimator=KNeighborsClassifier(), k_features=(10, 50),
                          scoring='roc_auc')
```

In [53]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w Feature engineering')
plt.grid()
plt.show()
```



In [63]:

```
knntop_50_features = list(selector.k_feature_names_)
print(top_50_features)
```

```
[4, 7, 14, 30, 33, 46, 49, 65, 67, 82, 90, 92, 95, 98, 100, 111, 115, 117, 120, 145, 147, 167, 207,
239, 260, 266, 272, 280, 286, 301, 312, 315, 318, 320, 331, 343, 347, 350, 351, 352, 353, 354, 355,
356, 358, 359, 360, 361, 362]
```

In [26]:

```
knntop_50_features = [4, 7, 14, 30, 33, 46, 49, 65, 67, 82, 90, 92, 95, 98, 100, 111, 115, 117, 120, 145, 147, 16
7, 207, 239, 260, 266, 272, 280,
286, 301, 312, 315, 318, 320, 331, 343, 347, 350, 351, 352, 353, 354, 355, 356, 358, 359, 360, 361, 362]
```

In [65]:

```
df_Xtrain[knntop_50_features]
```

Out[65]:

	4	7	14	30	33	46	49	65	67	82	90	
0	1.265232	0.210682	0.910638	1.162883	0.512526	0.793763	0.665237	-0.788053	-1.550872	-0.344336	-1.736100	0.22
1	-0.415729	0.291718	0.648998	-0.177127	-2.552011	-0.919554	0.139402	1.239630	1.107783	0.532991	-0.034617	0.15
2	-0.022827	-0.267006	1.201470	-1.966120	1.044330	-1.528443	-1.481126	0.956508	0.563348	-0.032919	0.716311	-1.3
3	-0.433148	0.805660	1.150656	1.038001	0.521405	1.354503	-0.892191	-0.722874	1.607856	2.142540	-0.628326	-1.6
4	1.183942	1.525391	-1.508995	-0.257409	0.169171	0.191896	0.381286	0.359712	-0.177687	0.019542	-1.975032	0.64
...	...	...	...	...	...	...	...	...	...	...	...	...
245	1.576844	-0.778815	-2.008490	-1.512182	-1.264432	0.929184	0.461595	-0.339946	0.100579	0.797528	-1.083434	-0.2
246	0.810395	-0.078277	-0.009428	0.344208	0.133652	-1.477284	0.093511	-1.041640	-0.838066	-0.323129	-0.544545	-0.5
247	1.579747	1.645879	-1.211676	2.025168	0.108985	-0.026783	-0.490644	-0.198385	1.530224	-0.244995	0.026409	-0.0
248	-0.085730	-0.673255	1.057676	1.065752	-0.488925	-0.620626	0.949187	-0.383738	-0.374289	-1.001774	-0.828987	0.21
249	1.300071	0.606267	0.331137	-0.086934	-0.459325	-0.020764	0.622214	0.489051	1.104758	0.072003	0.390495	-0.3

250 rows × 49 columns

In [66]:

```
knn_model = KNeighborsClassifier(**knn_clf.best_params_)
knn_model.fit(df_Xtrain[knntop_50_features],y_train)
```

Out[66]:

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

In [67]:

```
y_pred = knn_model.predict(df_Xtrain[knntop_50_features])
print(y_pred)
```

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

In [ ]:

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

```
0.54375
```

In [ ]:

```
y_predict = knn_model.predict_proba(df_Xtest[knntop_50_features])[:,1]
```

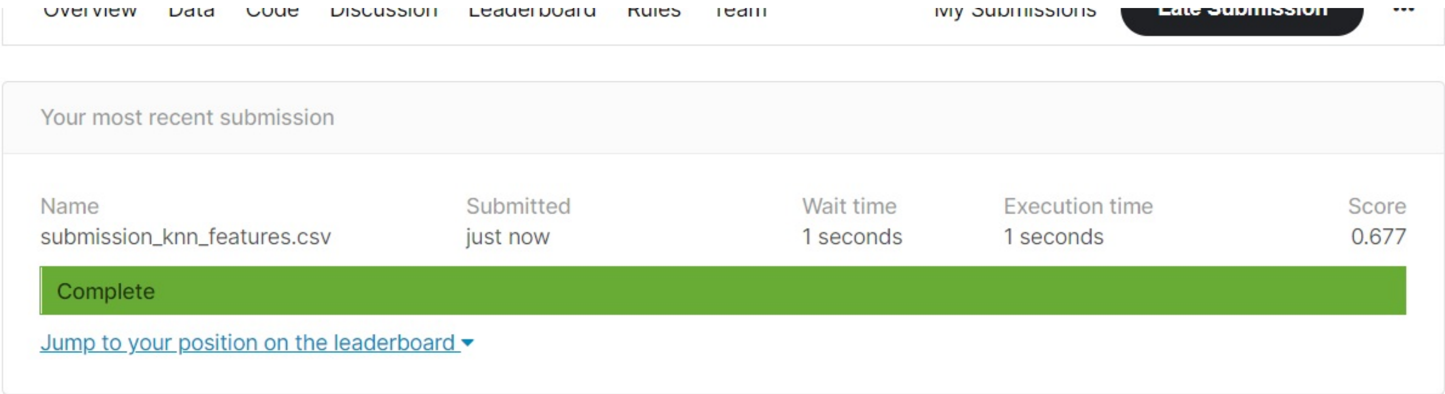
In [ ]:

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


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

Out[ ]:

	ID	Target
0	250	0.489796
1	251	0.693878
2	252	0.571429
3	253	0.653061
4	254	0.551020
5	255	0.489796
6	256	0.612245
7	257	0.673469
8	258	0.693878
9	259	0.673469
10	260	0.693878
11	261	0.612245
12	262	0.591837
13	263	0.653061
14	264	0.632653
15	265	0.632653
16	266	0.632653
17	267	0.653061
18	268	0.530612
19	269	0.489796



test_auc = 0.677

Let's take top 100 features

In [68]:

```
knn= KNeighborsClassifier()
#refhttps://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html
selector = SequentialFeatureSelector(knn,k_features=(10, 100),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

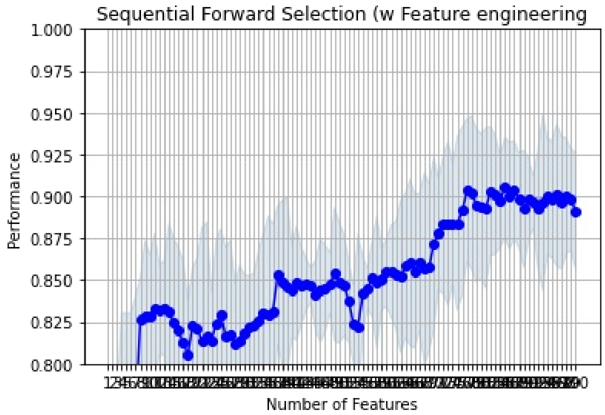
Out[68]:

```
SequentialFeatureSelector(estimator=KNeighborsClassifier(),
                          k_features=(10, 100), scoring='roc_auc')
```

In [69]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w Feature engineering')
plt.grid()
plt.show()
```



In [71]:

```
knntop_100_features = list(selector.k_feature_names_)
print(knntop_100_features)
```

```
['4', '5', '7', '14', '30', '33', '46', '48', '49', '60', '65', '67', '68', '69', '73', '75', '80',
'82', '90', '92', '95', '98', '100', '111', '114', '115', '116', '117', '120', '140', '145', '147',
'161', '163', '167', '168', '171', '172', '193', '195', '202', '207', '217', '224', '239', '253', '2
60', '266', '272', '280', '282', '286', '295', '297', '300', '301', '305', '306', '309', '310', '312
', '315', '316', '317', '318', '320', '329', '331', '343', '344', '347', '349', '350', '351', '352',
'353', '354', '355', '356', '357', '358', '359', '360', '361', '362']
```

In [27]:

```
knntop_100_features = [4, 5, 7, 14, 30, 33, 46, 48, 49, 60, 65, 67, 68, 69, 73, 75, 80,
                82, 90, 92, 95, 98, 100, 111, 114, 115, 116, 117, 120, 140, 145, 147,
                161, 163, 167, 168, 171, 172, 193, 195, 202, 207, 217, 224, 239, 253,
                260, 266, 272, 280, 282, 286, 295, 297, 300, 301, 305, 306, 309, 310,
                312, 315, 316, 317, 318, 320, 329, 331, 343, 344, 347, 349, 350, 351,
                352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362]
```

In [ ]:

```
knn_model = KNeighborsClassifier(**knn_clf.best_params_)
knn_model.fit(df_Xtrain[knntop_100_features],y_train)
```

Out[ ]:

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

In [ ]:

```
y_pred = knn_model.predict(df_Xtrain[knntop_100_features])
print(y_pred)
```

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

In [ ]:

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

```
0.5381944444444444
```

In [ ]:

```
y_predict = knn_model.predict_proba(df_Xtest[knntop_100_features])[:,1]
print(y_predict)
```

```
[0.53061224 0.69387755 0.71428571 ... 0.73469388 0.65306122 0.57142857]
```

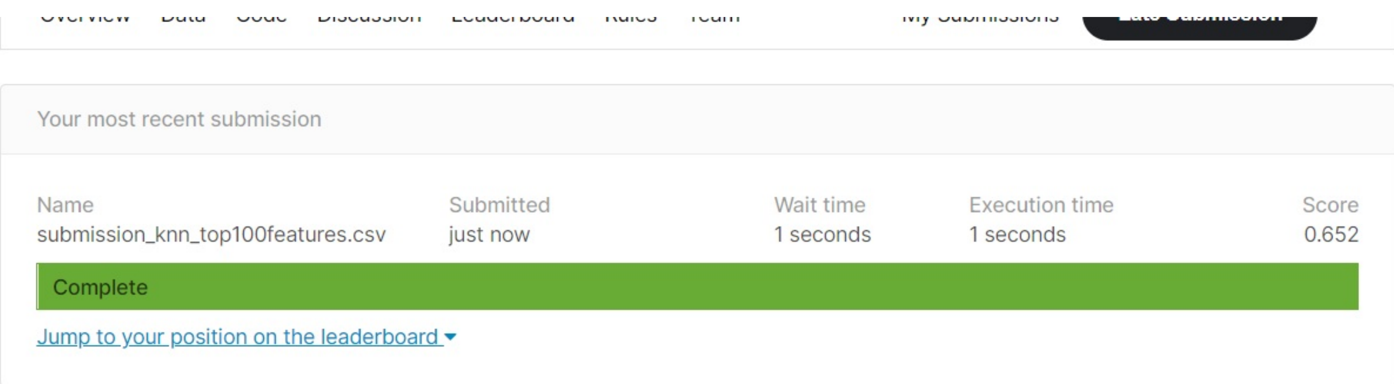
In [ ]:

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


y_pred_lr_test.to_csv('submission_knn_top100features.csv', index=False)
y_pred_lr_test.head(20)
```

Out[ ]:

	ID	Target
0	250	0.530612
1	251	0.693878
2	252	0.714286
3	253	0.693878
4	254	0.612245
5	255	0.591837
6	256	0.612245
7	257	0.612245
8	258	0.755102
9	259	0.591837
10	260	0.653061
11	261	0.612245
12	262	0.551020
13	263	0.612245
14	264	0.551020
15	265	0.632653
16	266	0.653061
17	267	0.693878
18	268	0.551020
19	269	0.489796



Logistic Regresstion Applied

In [20]:

```
#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 [74]:

```
selector = SequentialFeatureSelector(log_model,k_features=(10, 50),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

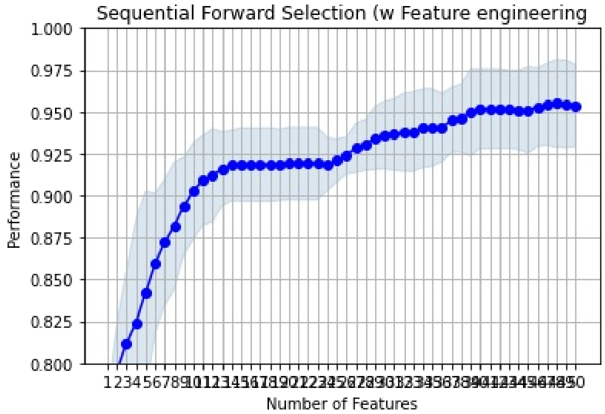
Out[74]:

```
SequentialFeatureSelector(estimator=LogisticRegression(random_state=42),
                          k_features=(10, 50), scoring='roc_auc')
```

In [75]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w Feature engineering')
plt.grid()
plt.show()
```



In [76]:

```
logistictop_50_features = list(selector.k_feature_names_)
print(top_50_features)
```

```
[4, 7, 14, 30, 33, 46, 49, 65, 67, 82, 90, 92, 95, 98, 100, 111, 115, 117, 120, 145, 147, 167, 207,
239, 260, 266, 272, 280, 286, 301, 312, 315, 318, 320, 331, 343, 347, 350, 351, 352, 353, 354, 355,
356, 358, 359, 360, 361, 362]
```

In [28]:

```
logistictop_50_features =  [4, 7, 13, 16, 24, 33, 49, 51, 64, 65, 67, 73, 90, 91,
                     104, 114, 116, 117, 123, 143, 149, 156, 164, 183, 188, 192, 217, 221, 226, 228, 253, 256,
                     260, 268, 302, 311, 313, 328, 330, 337, 338, 339, 355, 356, 358, 360, 361, 362]
```

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(df_Xtrain[top_50_features],y_train)
```

Out[ ]:

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

In [ ]:

```
y_pred = model.predict(df_Xtrain[logistictop_50_features])
print(y_pred)
```

```
[1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 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. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 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. 1. 1. 1. 0. 0. 1.
 0. 1. 0. 1. 0. 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. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 0. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1.
 1. 0. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 1. 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)
```

```
0.9027777777777778
```

In [ ]:

```
y_pred_lr_test = model.predict_proba(df_Xtest[logistictop_50_features])[:,1]
print(y_pred_lr_test)
```

```
[0.0445232  0.33663622 0.80638496 ... 0.53660329 0.99484058 0.12661038]
```

In [ ]:

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


y_pred_lr_test.to_csv('submission_logs_50features.csv', index=False)
y_pred_lr_test.head(20)
```

Out[ ]:

	ID	Target
0	250	0.044523
1	251	0.336636
2	252	0.806385
3	253	0.932935
4	254	0.138762
5	255	0.101394
6	256	0.280308
7	257	0.028632
8	258	0.872992
9	259	0.096193
10	260	0.285565
11	261	0.245310
12	262	0.112716
13	263	0.997687
14	264	0.339740
15	265	0.991135
16	266	0.877559
17	267	0.968095
18	268	0.238123
19	269	0.013618

Name	Submitted	Wait time	Execution time	Score
submission_logs_50features.csv	just now	1 seconds	0 seconds	0.818

Complete

Jump to your position on the leaderboard

logistic_test_auc = 0.818

Let's take top 100 features

In [21]:

```
selector = SequentialFeatureSelector(log_model,k_features=(10, 100),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

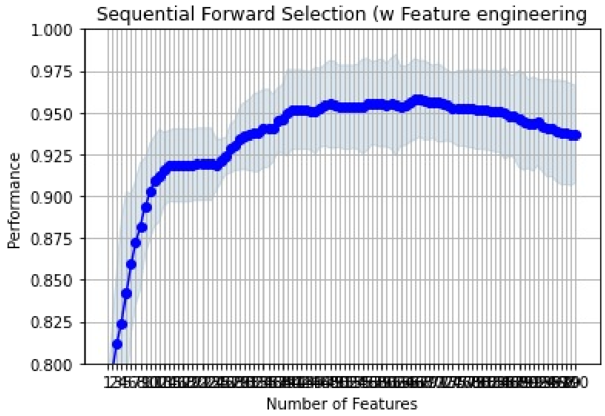
Out[21]:

```
SequentialFeatureSelector(estimator=LogisticRegression(random_state=42),
                          k_features=(10, 100), scoring='roc_auc')
```

In [ ]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w Feature engineering')
plt.grid()
plt.show()
```



In [ ]:

```
logistictop_100_features = list(selector.k_feature_names_)
print(top_100_features)
```

```
['4', '7', '13', '16', '21', '24', '33', '49', '51', '61', '64', '65', '67', '73', '79', '90', '91',
'104', '105', '114', '116', '117', '123', '129', '135', '143', '149', '156', '161', '164', '183', '1
88', '192', '196', '199', '213', '217', '221', '226', '228', '253', '256', '260', '265', '268', '269
', '274', '279', '285', '302', '311', '313', '319', '320', '328', '330', '337', '338', '339', '355',
'356', '358', '359', '360', '361', '362']
```

In [29]:

```
logistictop_100_features = [4, 7, 13, 16, 21, 24, 33, 49, 51, 61, 64, 65, 67, 73,
                    79, 90, 91, 104, 105, 114, 116, 117, 123, 129, 135, 143,
                    149, 156, 161, 164, 183, 188, 192, 196, 199, 213, 217, 221,
                    226, 228, 253, 256, 260, 265, 268, 269, 274, 279, 285, 302,
                    311, 313, 319, 320, 328, 330, 337, 338, 339, 355, 356, 358,
                    359, 360, 361, 362]
```

In [ ]:

```
from sklearn import linear_model

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

model.fit(df_Xtrain[logistictop_100_features],y_train)
```

Out[ ]:

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

In [ ]:

```
y_pred = model.predict(df_Xtrain[logistictop_100_features])
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. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 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. 1. 0. 1. 1. 1. 0. 1.
 0. 1. 0. 1. 0. 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. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1.
 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0.
 0. 1. 1. 0. 0. 1. 0. 0. 1. 0.]
```

In [ ]:

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

```
0.9392361111111113
```

In [ ]:

```
y_predict = model.predict_proba(df_Xtest[logistictop_100_features])[:,1]
print(y_predict)
```

```
[0.06088335 0.38840025 0.65453023 ... 0.38744559 0.99585269 0.25010157]
```

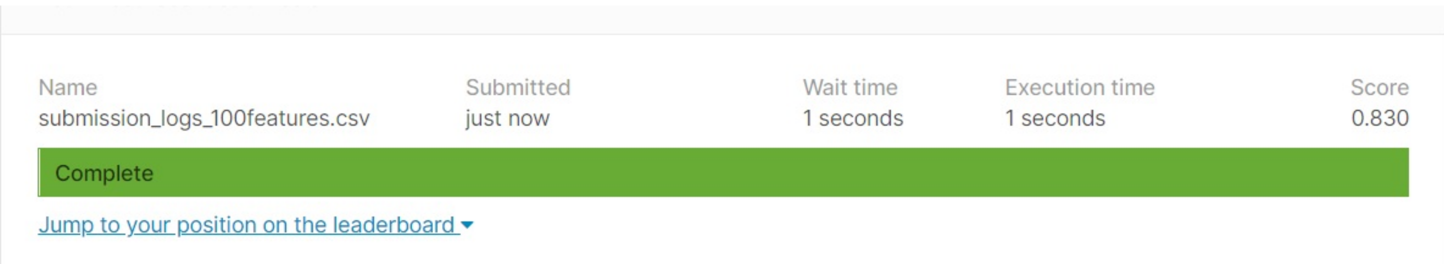
In [ ]:

```
y_pred_lr_test_100 = pd.DataFrame({"ID": df_test['id'],"Target": y_predict})


y_pred_lr_test_100.to_csv('submission_logs_100features.csv', index=False)
y_pred_lr_test_100.head(20)
```

Out[ ]:

	ID	Target
0	250	0.060883
1	251	0.388400
2	252	0.654530
3	253	0.981406
4	254	0.044851
5	255	0.105564
6	256	0.366869
7	257	0.018923
8	258	0.892434
9	259	0.117482
10	260	0.355460
11	261	0.125471
12	262	0.038672
13	263	0.997493
14	264	0.241079
15	265	0.996510
16	266	0.873278
17	267	0.986878
18	268	0.451083
19	269	0.011177



test_logistic_auc = 0.83

Support Vector Machine

In [38]:

```
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(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 [ ]:

```
selector = SequentialFeatureSelector(svc_model,k_features=(10, 50),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

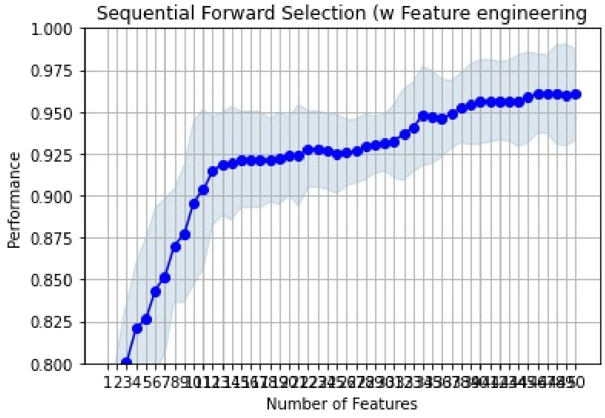
Out[ ]:

```
SequentialFeatureSelector(estimator=SVC(random_state=42), k_features=(10, 50),
                          scoring='roc_auc')
```

In [ ]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w Feature engineering')
plt.grid()
plt.show()
```



In [ ]:

```
svctop_50_features = list(selector.k_feature_names_)
print(top_50_features)
```

```
['9', '28', '31', '33', '45', '46', '62', '65', '66', '68', '73', '80', '91', '95', '100', '114', '1
23', '131', '132', '133', '141', '146', '159', '168', '169', '181', '194', '195', '217', '226', '233
', '249', '253', '258', '264', '272', '282', '288', '304', '350', '354', '355', '356', '358', '359',
'362']
```

In [23]:

```
svctop_50_features = [9, 28, 31, 33, 45, 46, 62, 65, 66, 68,
                      73, 80, 91, 95, 100, 114, 123, 131, 132,
                      133, 141, 146, 159, 168, 169, 181, 194, 195,
                      217, 226, 233, 249, 253, 258, 264, 272, 282,
                      288, 304, 350, 354, 355, 356, 358, 359, 362]
```

In [ ]:

```
from sklearn import linear_model

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

model.fit(df_Xtrain[svctop_50_features],y_train)
```

Out[ ]:

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

In [ ]:

```
y_pred = model.predict(df_Xtrain[svctop_50_features])
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. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1.
 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 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. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0.
 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1.
 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 0. 1. 1. 0. 0. 0. 0. 0. 1. 0.]
```

In [ ]:

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

```
0.8798611111111111
```

In [ ]:

```
y_pred_svc = model.predict_proba(df_Xtest[svctop_50_features])[:,1]
print(y_pred_svc)
```

```
[0.14506099 0.4656034  0.92775101 ... 0.86146204 0.91354606 0.65519584]
```

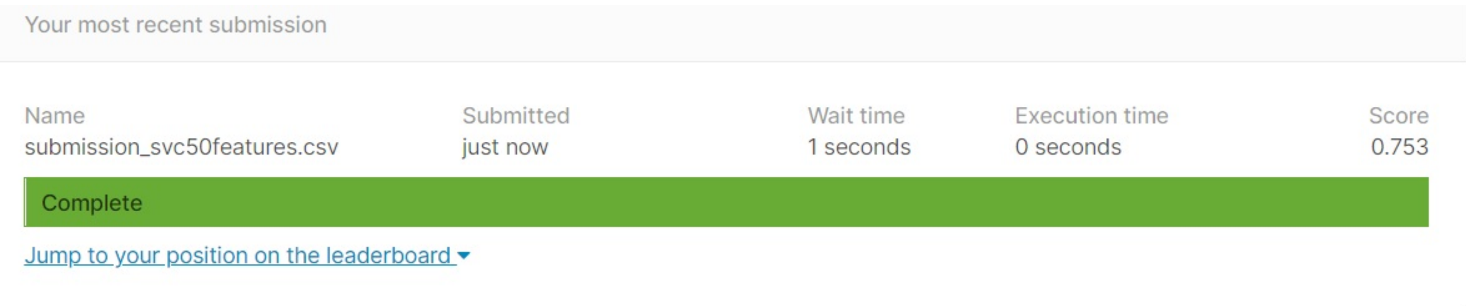
In [ ]:

```
y_pred_svc_test_50 = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_svc})


y_pred_svc_test_50.to_csv('submission_svc50features.csv', index=False)
y_pred_svc_test_50.head(20)
```

Out[ ]:

	ID	Target
0	250	0.145061
1	251	0.465603
2	252	0.927751
3	253	0.896267
4	254	0.653264
5	255	0.630216
6	256	0.500000
7	257	0.160114
8	258	0.903210
9	259	0.086692
10	260	0.757806
11	261	0.199269
12	262	0.770502
13	263	0.935361
14	264	0.018196
15	265	0.507099
16	266	0.234485
17	267	0.895184
18	268	0.863253
19	269	0.148971



Test_SVM_auc = 0.753

Let's take top 100 features

In [ ]:

```
selector = SequentialFeatureSelector(svc_model,k_features=(10, 100),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

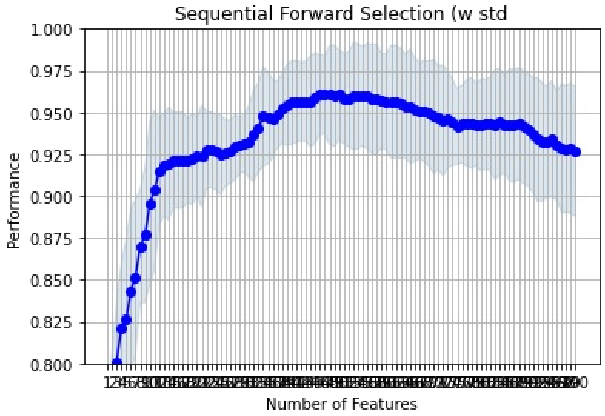
Out[ ]:

```
SequentialFeatureSelector(estimator=SVC(random_state=42), k_features=(10, 100),
                          scoring='roc_auc')
```

In [ ]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w std')
plt.grid()
plt.show()
```



In [ ]:

```
svctop_100_features = list(selector.k_feature_names_)
print(top_100_features)
```

```
['9', '28', '31', '33', '45', '46', '62', '65', '66', '68', '73', '80', '91', '95', '100', '114', '1
23', '131', '132', '133', '141', '146', '159', '168', '169', '181', '194', '195', '217', '226', '233
', '249', '253', '258', '264', '272', '282', '288', '304', '350', '354', '355', '356', '358', '359',
'362']
```

In [24]:

```
svctop_100_features = [9, 28, 31, 33, 45, 46, 62, 65, 66, 68,
                       73, 80, 91, 95, 100, 114, 123, 131, 132,
                       133, 141, 146, 159, 168, 169, 181, 194, 195,
                       217, 226, 233, 249, 253, 258, 264, 272, 282,
                       288, 304, 350, 354, 355, 356, 358, 359, 362]
```

In [ ]:

```
from sklearn import linear_model

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

model.fit(df_Xtrain[svctop_100_features],y_train)
```

Out[ ]:

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

In [ ]:

```
y_pred = model.predict(df_Xtrain[svctop_100_features])
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. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1.
 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 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. 1. 1. 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 1. 0.
 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1.
 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 0. 1. 1. 0. 0. 0. 0. 0. 1. 0.]
```

In [ ]:

```
train_svc_auc_100features = roc_auc_score(y_train,y_pred)
print(train_svc_auc_100features)
```

```
0.8798611111111111
```

In [ ]:

```
y_pred_svc_top100 = model.predict_proba(df_Xtest[svctop_100_features])[:,1]
```

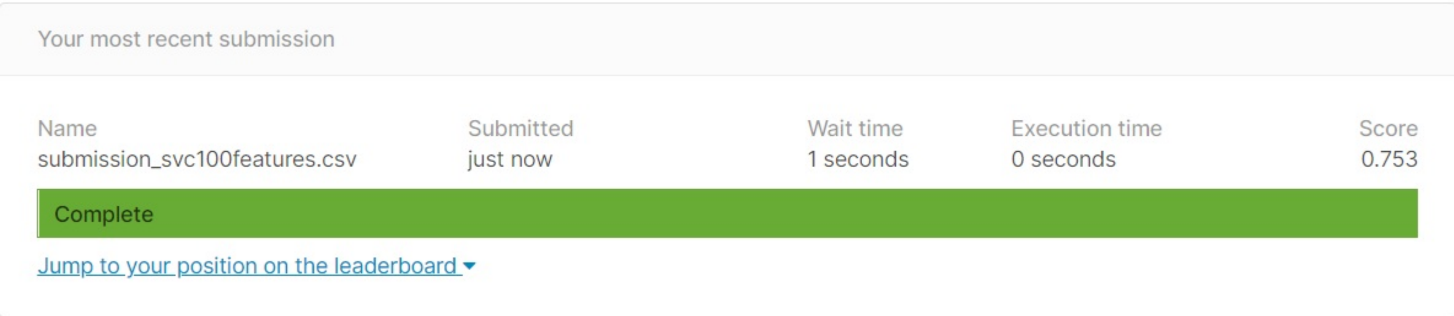
In [ ]:

```
y_pred_svc_test_100 = pd.DataFrame({"ID": df_test['id'],"Target": y_pred_svc_top100})


y_pred_svc_test_100.to_csv('submission_svc100features.csv', index=False)
y_pred_svc_test_100.head(10)
```

Out[ ]:

	ID	Target
0	250	0.167671
1	251	0.500000
2	252	0.935021
3	253	0.906848
4	254	0.684161
5	255	0.660151
6	256	0.532745
7	257	0.184330
8	258	0.913066
9	259	0.101990



Ensemble Model : Random Forest

In [37]:

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

In [20]:

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

```
selector = SequentialFeatureSelector(rdf_model,k_features=(10, 15),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

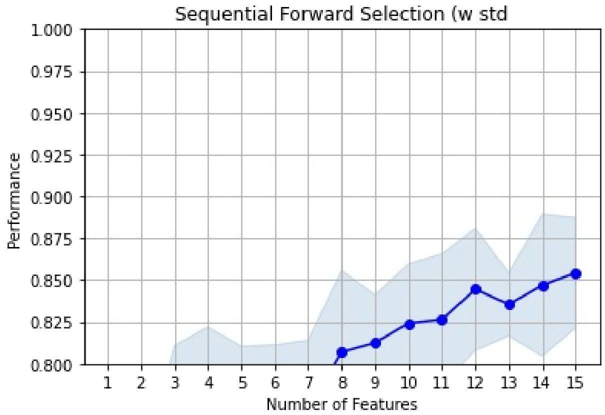
Out[ ]:

```
SequentialFeatureSelector(estimator=RandomForestClassifier(random_state=42),
                          k_features=(10, 15), scoring='roc_auc')
```

In [ ]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w std')
plt.grid()
plt.show()
```



In [ ]:

```
rdftop_15_features = list(selector.k_feature_names_)
print(top_15_features)
```

```
['2', '17', '33', '45', '65', '96', '116', '194', '199', '214', '215', '217', '315', '317', '320']
```

In [31]:

```
rdftop_15_features = [2, 17, 33, 45, 65, 96, 116, 194, 199, 214, 215, 217, 315, 317, 320]
```

In [ ]:

```
df_Xtrain[rdftop_15_features]
```

Out[ ]:

	2	17	33	45	65	96	116	194	199	214	215	
0	0.503692	-0.560123	0.512526	0.807873	-0.788053	1.142777	-0.294448	-0.306051	1.730666	-0.505591	0.810781	1
1	-0.539790	-0.128760	-2.552011	-1.184575	1.239630	-1.454985	0.651431	0.014025	-2.250726	-0.028172	0.318050	0
2	-0.505465	0.372848	1.044330	0.951000	0.956508	-1.749053	0.076352	-0.589873	0.354049	-0.084865	0.295837	0
3	0.220265	0.932829	0.521405	-0.936420	-0.722874	-1.118907	0.428757	-1.006283	1.126227	0.470135	1.409531	0
4	0.336970	-0.212856	0.169171	0.304356	0.359712	0.520447	0.037626	0.070997	0.157389	0.060350	1.964863	0
...	...	...	...	...	...	...	...	...	...	...	...	..
245	-1.054666	-0.952901	-1.264432	-1.099111	-0.339946	1.409489	-2.046792	0.159044	0.135217	0.362716	0.174673	0
246	-0.536848	-0.208898	0.133652	-1.393602	-1.041640	-0.072573	-0.445479	0.652106	0.577701	0.765539	-0.892575	1
247	-0.047471	0.256103	0.108985	0.366137	-0.198385	0.045640	-0.412562	-0.844691	0.060024	-1.304275	-1.052107	1
248	0.016276	-0.249462	-0.488925	-0.368031	-0.383738	-0.503417	0.076352	-1.472413	-0.394993	0.300054	0.613891	-
249	-0.027856	-1.509913	-0.459325	-0.175480	0.489051	-0.381296	-0.871463	0.414898	-0.707334	1.632851	0.069665	1

250 rows x 15 columns

In [ ]:

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

```
{'max_depth': 9, 'n_estimators': 400}
```

In [ ]:

```
rdf_clf = RandomForestClassifier(**rdf_clf.best_params_,bootstrap=True)
rdf_clf.fit(df_Xtrain[rdftop_15_features],y_train)
```

Out[ ]:

```
RandomForestClassifier(max_depth=9, n_estimators=400)
```

In [ ]:

```
y_pred = rdf_clf.predict(df_Xtrain[rdftop_15_features])
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(df_Xtest[rdftop_15_features])[:,1]
```

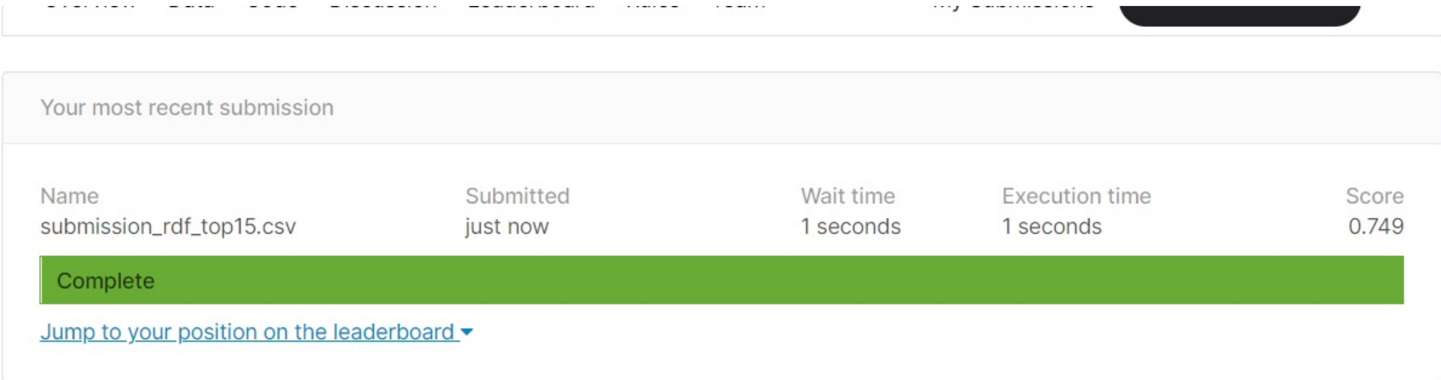
In [ ]:

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


y_pred_rdf_test.to_csv('submission_rdf_top15.csv', index=False)
y_pred_rdf_test.head(20)
```

Out[ ]:

	ID	Target
0	250	0.534700
1	251	0.777835
2	252	0.562941
3	253	0.862542
4	254	0.489986
5	255	0.412787
6	256	0.396291
7	257	0.673244
8	258	0.932938
9	259	0.536302
10	260	0.886898
11	261	0.523860
12	262	0.683837
13	263	0.658972
14	264	0.222353
15	265	0.867772
16	266	0.432188
17	267	0.579593
18	268	0.782165
19	269	0.720348



Test_rdf_auc : 0.75

XGBoost Classifier

In [39]:

```
from xgboost import XGBClassifier
```

In [20]:

```
#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 [23]:

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

```
{'max_depth': 9, 'n_estimators': 500}
```

In [22]:

```
selector = SequentialFeatureSelector(xg_model,k_features=(10, 50),forward=True,cv=5,scoring='roc_auc')
selector.fit(X_train,y_train)
```

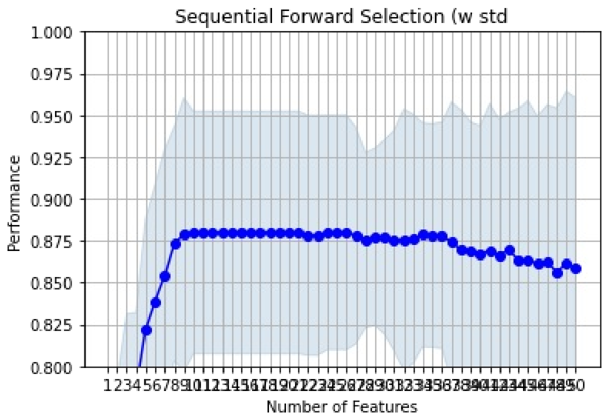
Out[22]:

```
SequentialFeatureSelector(estimator=XGBClassifier(), k_features=(10, 50),
                          scoring='roc_auc')
```

In [24]:

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1 = plot_sfs(selector.get_metric_dict(), kind='std_dev')

plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w std')
plt.grid()
plt.show()
```



In [25]:

```
xgboottop_50_features = list(selector.k_feature_names_)
print(top_50_features)
```

```
['33', '50', '65', '91', '164', '167', '230', '301', '361', '362']
```

In [30]:

```
xgboottop_50_features = [33, 50, 65, 91, 164, 167, 230, 301, 361, 362]
```

In [27]:

```
xg_clf = XGBClassifier(**xg_clf.best_params_)
xg_clf.fit(df_Xtrain[xgboottop_50_features],y_train)
```

Out[27]:

```
XGBClassifier(max_depth=9, n_estimators=500)
```

In [31]:

```
y_pred = xg_clf.predict(df_Xtrain[xgboottop_50_features])
train_auc = roc_auc_score(y_train,y_pred)
print(train_auc)
```

```
1.0
```

In [32]:

```
y_pred_xg_test = xg_clf.predict_proba(df_Xtest[xgboottop_50_features])[:,1]
print(y_pred_xg_test)
```

```
[0.29076236 0.2061423  0.01131213 ... 0.89664805 0.99611    0.26628968]
```

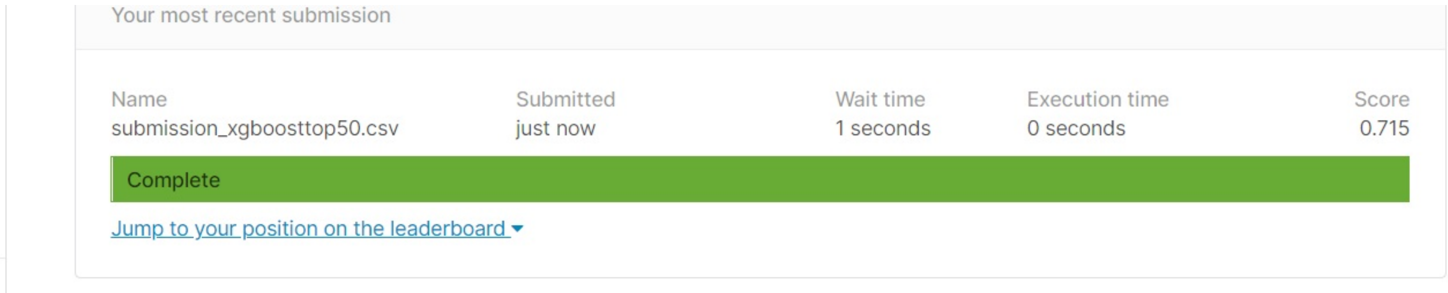
In [33]:

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


y_pred_xg_test.to_csv('submission_xgboosttop50.csv', index=False)
y_pred_xg_test.head(10)
```

Out[33]:

	ID	Target
0	250	0.290762
1	251	0.206142
2	252	0.011312
3	253	0.998524
4	254	0.536212
5	255	0.089159
6	256	0.140542
7	257	0.136743
8	258	0.999746
9	259	0.004194



xgboost_test_auc = 0.715

Stacking Classifier

In [32]:

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

In [33]:

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

In [57]:

```
# Combined all top features of all models
# Ref: https://www.geeksforgeeks.org/union-function-python/

all_top_feat = logistictop_50_features+knntop_100_features+svctop_50_features+xgboottop_50_features
```

In [61]:

```
#classifier 1
knn_model = KNeighborsClassifier(algorithm='kd_tree', n_neighbors=49)
knn_model.fit(X_train[:,all_top_feat],y_train)

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

#Classifier 3
svc_clf = SVC(C =0.1, kernel = 'linear',probability=True)
svc_clf.fit(X_train[:,all_top_feat],y_train)

#classifier 3

rdf_clf = RandomForestClassifier(max_depth=9, n_estimators=400)
rdf_clf.fit(X_train[:,all_top_feat],y_train)



#classifier 5
xg_clf = XGBClassifier(max_depth = 9, n_estimators = 500)
xg_clf.fit(X_train[:,all_top_feat],y_train)


#Stacking Classifer

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

#fit the model
sclf.fit(X_train[:,all_top_feat],y_train)

#predict in probabilities

y_pred = sclf.predict(X_train[:,all_top_feat])
```

In [62]:

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

1.0

In [66]:

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

[0.12189521 0.98701918 0.96373052 ... 0.25666822 0.99342974 0.78469862]

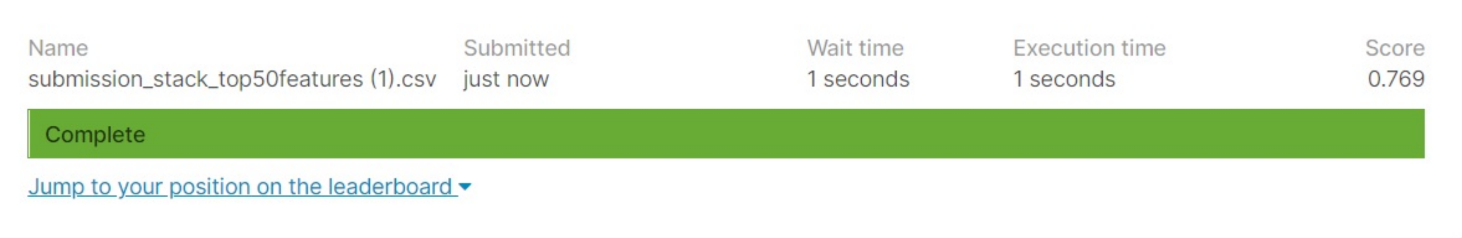
In [64]:

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


y_pred_stack_test.to_csv('submission_stack_top50features.csv', index=False)
y_pred_stack_test.head(20)
```

Out[64]:

	ID	Target
0	250	0.121895
1	251	0.987019
2	252	0.963731
3	253	0.993992
4	254	0.776488
5	255	0.991428
6	256	0.239680
7	257	0.051843
8	258	0.993739
9	259	0.099150
10	260	0.986614
11	261	0.863000
12	262	0.505329
13	263	0.992677
14	264	0.787459
15	265	0.991170
16	266	0.681878
17	267	0.993208
18	268	0.102114
19	269	0.024689



Summary of All Models Using Top 50 Features

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
9)",0.54,0.68],
           ['logistic Regresstion',r"(C=1, penalty='l1', solver='saga')",0.90,0.82],['Support Vector Machine',
           r"{'C': 0.1, 'kernel': 'linear'}",0.87,0.76],['XGBoost Classifier',r"{'max_depth': 9, 'n_estimators':
500}",1.00,0.76],
            ['Random forest',r"{'max_depth': 9, 'n_estimators': 400}",1.0,0.75],
            ['Calibrated Model',"--",1.00,0.77]])

print(t.draw())
```

```
+------------------------+------------------------------+-----------+----------+
|         Model          |       Hyper-parameter        | Train AUC | Test AUC |
+========================+==============================+===========+==========+
| Knn_Model              | (algorithm='kd_tree',        | 0.540     | 0.680    |
|                        | n_neighbors=49)              |           |          |
+------------------------+------------------------------+-----------+----------+
| logistic Regresstion   | (C=1, penalty='l1',          | 0.900     | 0.820    |
|                        | solver='saga')               |           |          |
+------------------------+------------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.1, 'kernel':         | 0.870     | 0.760    |
|                        | 'linear'}                    |           |          |
+------------------------+------------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 9,             | 1         | 0.760    |
|                        | 'n_estimators': 500}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Random forest          | {'max_depth': 9,             | 1         | 0.750    |
|                        | 'n_estimators': 400}         |           |          |
+------------------------+------------------------------+-----------+----------+
| Calibrated Model       | --                           | 1         | 0.770    |
+------------------------+------------------------------+-----------+----------+
```

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. Applied Feature Engineering to came up new features.
3. Dropped the labled data from both train and test datasets.
4. Standradized the Features using StandardScaler method.
5. Used GridSerach Validation for hyper-parameter tuning.
6. We have applied following machine learning algorithm: 1.KNN : The KNN algorithm trained the model with parameter(algorithm = 'kd_tree', and n_neighbors=49). Used forward Feature selection Method and came up top 50 Feature based on performance. Re-train the model using KNN with best parameter and got train_AUC=0.54 and Test Auc = 0.68 . 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). Used forward Feature selection Method and came up top 50 Feature based on performance. Re-traing the model with best parameter on top 50 Features and got train_AUC = 0.90 and Test_auc=0.82 which is working as expected.

   3.Support Vector Machine: The SVM algorithm trained the model with parameter(C=0.1,kernel=linear).Used forward Feature selection Method and came up top 50 Feature based on performance.Re-train the model with best parameter on top 50 features and got the train_AUC=0.87 and Test_AUC = 0.76.Model is not overfitted

   4.XGBoost Classifier: The XGBoost classifier trained the model with parameter(max_depth=9,n_estimators=500). Used forward Feature selection Method and came up top 50 Feature based on performance.Re-train the model with best parameter on top 50 features and got the train_AUC= 1.0 and Test_AUC=0.76.Model is not overfitted.

   5.Random Forest : The Random Forest classifier trained the model with parameter(max_depth=9,n_estimators=400).Used forward Feature selection Method and came up top 50 Feature based on performance. Re-train the model on top 50 features and got the train_AUC = 1.0 and test_AUC = 0.75.Model is not overfitted.

   7.Calibrated model gave train_AUC = 1.0 and Test_AUC = 0.77. Model is accurate and not overfitted.

Logistic Regression is working well from above applied algorithm.

Summary of All Models Using Top 100 Features

In [4]:

```
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
9)",0.54,0.65],
          ['logistic Regresstion',r"(C=1, penalty='l1', solver='saga')",0.93,0.83],['Support Vector Machine',
          r"{'C': 0.1, 'kernel': 'linear'}",0.88,0.75],['XGBoost Classifier',r"{'max_depth': 9, 'n_estimators':
500}",1.00,0.75]])

print(t.draw())
```

```
+------------------------+-----------------------------+-----------+----------+
|         Model          |       Hyper-parameter       | Train AUC | Test AUC |
+========================+=============================+===========+==========+
| Knn_Model              | (algorithm='kd_tree',       | 0.540     | 0.650    |
|                        | n_neighbors=49)             |           |          |
+------------------------+-----------------------------+-----------+----------+
| logistic Regresstion   | (C=1, penalty='l1',         | 0.930     | 0.830    |
|                        | solver='saga')              |           |          |
+------------------------+-----------------------------+-----------+----------+
| Support Vector Machine | {'C': 0.1, 'kernel':        | 0.880     | 0.750    |
|                        | 'linear'}                   |           |          |
+------------------------+-----------------------------+-----------+----------+
| XGBoost Classifier     | {'max_depth': 9,            | 1         | 0.750    |
|                        | 'n_estimators': 500}        |           |          |
+------------------------+-----------------------------+-----------+----------+
```

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. Applied Feature Engineering to came up new features.
3. Dropped the labled data from both train and test datasets.
4. Standradized the Features using StandardScaler method.
5. Used GridSerach Validation for hyper-parameter tuning.
6. We have applied following machine learning algorithm: 1.KNN : The KNN algorithm trained the model with parameter(algorithm = 'kd_tree', and n_neighbors=49). Used forward Feature selection Method and came up top 100 Features based on performance. Re-train the model using KNN with best parameter and got 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=1,penalty=l1,solver=saga). Used forward Feature selection Method and came up top 100 Features based on performance. Re-traing the model with best parameter on top 100 Features and got train_AUC = 0.93 and Test_auc=0.83 which is working as expected.

   3.Support Vector Machine: The SVM algorithm trained the model with parameter(C=0.1,kernel=linear).Used forward Feature selection Method and came up top 100 Feature based on performance. Re-train the model with best parameter on top 100 features and got the train_AUC=0.88 and Test_AUC = 0.75.Model is not overfitted

   4.XGBoost Classifier: The XGBoost classifier trained the model with parameter(max_depth=9,n_estimators=500). Used forward Feature selection Method and came up top 100 Feature based on performance.Re-train the model with best parameter on top 100 features and got the train_AUC= 1.0 and Test_AUC=0.75.Model is not overfitted.

Logistic Regression is working well from above applied algorithm.