4_1_Models

April 20, 2020

1 Standardization + ML Classification Model with/without Top Features

1. No oversampling techniques applied
2. No feature engineering applied

2 1. Import Necessary Libraries

[17]:

```
# For Computational and random seed purpose
import numpy as np
np.random.seed(42)
# To read csv file
import pandas as pd
# To Split data into train and cv data
from sklearn.model_selection import train_test_split
# To compute AUROC score
# For AUROC Score (Ref: https://scikit-learn.org/stable/modules/generated/
 ↪sklearn.metrics.roc_auc_score.html)
from sklearn.metrics import  roc_curve, auc
# For Hyperparameter and CV Fold
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold,␣
 ↪cross_val_score
# For plot AUROC graph
import matplotlib.pyplot as plt
# Data is umbalance, we need Calibrated Model to ive confidence probabilities␣
 ↪result
from sklearn.calibration import CalibratedClassifierCV
# For heatmap
import seaborn as sns
# To ignore warninga
import warnings
warnings.filterwarnings('ignore')
# To stndardize the data
from sklearn.preprocessing import StandardScaler
import tqdm
```

3 2. Read train data

[2]:

```
# Locate parent directory
data_dir = "./"

# Read csv file and display top 5 rows
df_train = pd.read_csv(data_dir+'/train.csv')
df_train.head(5)
```

[2]:

```
   id  target      0      1      2      3      4      5      6      7  …  \
0   0     1.0 -0.098  2.165  0.681 -0.614  1.309 -0.455 -0.236  0.276  …
1   1     0.0  1.081 -0.973 -0.383  0.326 -0.428  0.317  1.172  0.352  …
2   2     1.0 -0.523 -0.089 -0.348  0.148 -0.022  0.404 -0.023 -0.172  …
3   3     1.0  0.067 -0.021  0.392 -1.637 -0.446 -0.725 -1.035  0.834  …
4   4     1.0  2.347 -0.831  0.511 -0.021  1.225  1.594  0.585  1.509  …

     290    291    292    293    294    295    296    297    298    299
0  0.867  1.347  0.504 -0.649  0.672 -2.097  1.051 -0.414  1.038 -1.065
1 -0.165 -1.695 -1.257  1.359 -0.808 -1.624 -0.458 -1.099 -0.936  0.973
2  0.013  0.263 -1.222  0.726  1.444 -1.165 -1.544  0.004  0.800 -1.211
3 -0.404  0.640 -0.595 -0.966  0.900  0.467 -0.562 -0.254 -0.533  0.238
4  0.898  0.134  2.415 -0.996 -1.006  1.378  1.246  1.478  0.428  0.253

[5 rows x 302 columns]
```

[3]:

```
df_test = pd.read_csv(data_dir+'/test.csv')
df_test.head(5)
```

[3]:

```
    id      0      1      2      3      4      5      6      7      8  …  \
0  250  0.500 -1.033 -1.595  0.309 -0.714  0.502  0.535 -0.129 -0.687  …
1  251  0.776  0.914 -0.494  1.347 -0.867  0.480  0.578 -0.313  0.203  …
2  252  1.750  0.509 -0.057  0.835 -0.476  1.428 -0.701 -2.009 -1.378  …
3  253 -0.556 -1.855 -0.682  0.578  1.592  0.512 -1.419  0.722  0.511  …
4  254  0.754 -0.245  1.173 -1.623  0.009  0.370  0.781 -1.763 -1.432  …

     290    291    292    293    294    295    296    297    298    299
0 -0.088 -2.628 -0.845  2.078 -0.277  2.132  0.609 -0.104  0.312  0.979
1 -0.683 -0.066  0.025  0.606 -0.353 -1.133 -3.138  0.281 -0.625 -0.761
2 -0.094  0.351 -0.607 -0.737 -0.031  0.701  0.976  0.135 -1.327  2.463
3 -0.336 -0.787  0.255 -0.031 -0.836  0.916  2.411  1.053 -1.601 -1.529
4  2.184 -1.090  0.216  1.186 -0.143  0.322 -0.068 -0.156 -1.153  0.825

[5 rows x 301 columns]
```

4 3. Take train and test values from DataFrame

[4]:

```
# Take separate for features value
tr_X = df_train.drop(['id','target'], axis=1)
# Take separate for class value
tr_y = df_train['target'].values
# Take test feature value
ts_X = df_test.drop(['id'], axis=1)
```

Note: Don't worry about splitting train data into train and cv. I apply Stratify CV technique while modelling

5 4. Standardization

[5]:

```
stand_vec = StandardScaler()
tr_X = stand_vec.fit_transform(tr_X)
pd.DataFrame(tr_X).head(5)
```

[5]:

```
          0         1         2         3         4         5         6  \
0 -0.121736  2.176002  0.503692 -0.609972  1.265232 -0.469388 -0.266814
1  1.061577 -0.939278 -0.539790  0.320974 -0.415729  0.340017  1.134681
2 -0.548290 -0.061678 -0.505465  0.144689 -0.022827  0.431232 -0.054798
3  0.043868  0.005829  0.220265 -1.623118 -0.433148 -0.752470 -1.062122
4  2.332208 -0.798306  0.336970 -0.022683  1.183942  1.678890  0.550393

          7         8         9  …       290       291       292       293  \
0  0.210682 -2.296917  1.758518  …  0.814697  1.257605  0.509875 -0.664341
1  0.291718  0.042547 -0.320787  … -0.207701 -1.876475 -1.339295  1.430147
2 -0.267006  0.180835  0.144993  … -0.031358  0.140793 -1.302542  0.769883
3  0.805660  0.561388  0.234415  … -0.444478  0.529204 -0.644150 -0.994994
4  1.525391  0.025911  2.125050  …  0.845409  0.007888  2.516556 -1.026286

        294       295       296       297       298       299
0  0.699064 -1.921131  1.040182 -0.421724  1.022411 -0.965720
1 -0.703385 -1.464704 -0.522377 -1.108796 -0.939658  1.136883
2  1.430611 -1.021786 -1.646923 -0.002459  0.785850 -1.116348
3  0.915116  0.553031 -0.630069 -0.261240 -0.539094  0.378584
4 -0.891009  1.432111  1.242103  1.476000  0.416098  0.394060

[5 rows x 300 columns]
```

[6]:

```
ts_X = stand_vec.transform(ts_X)
pd.DataFrame(ts_X).head(5)
```

[6]:

```
          0         1         2         3         4         5         6  \
0  0.478452 -0.998843 -1.728417  0.304138 -0.692502  0.533981  0.500624
1  0.755461  0.934060 -0.648649  1.332140 -0.840565  0.510915  0.543426
```

```
2  1.733024  0.531992 -0.220077  0.825072 -0.462180  1.504847 -0.729665
3 -0.581411 -1.814892 -0.833024  0.570547  1.539102  0.544465 -1.444348
4  0.733381 -0.216549  0.986204 -1.609253  0.007173  0.395585  0.745488

         7         8         9  …       290       291       292       293  \
0 -0.221157 -0.675928  1.233779  … -0.131418 -2.837717 -0.906667  2.180115
1 -0.417350  0.249460  1.297652  … -0.720882 -0.198166  0.006893  0.644714
2 -2.225742 -1.394404  0.129271  … -0.137362  0.231456 -0.656750 -0.756131
3  0.686238  0.569706  0.522334  … -0.377110 -0.940990  0.248408 -0.019722
4 -1.963439 -1.450551 -0.948706  …  2.119444 -1.253162  0.207456  1.249696

       294       295       296       297       298       299
0 -0.200209  2.159692  0.582494 -0.110786  0.300799  1.143073
1 -0.272227 -0.990908 -3.297498  0.275379 -0.630538 -0.652084
2  0.032901  0.778832  0.962520  0.128937 -1.328295  2.674115
3 -0.729917  0.986299  2.448452  1.049714 -1.600639 -1.444429
4 -0.073231  0.413112 -0.118535 -0.162943 -1.155347  0.984192

[5 rows x 300 columns]
```

6 5. Apply ML Models (with hyperparameter)

[8]:

```python
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
    '''
    # Random shuffle after every iteration with stratify
    str_cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5, random_state=42)
    # Find the right hyperparameter for the model
    grid_clf = GridSearchCV(models, params, cv=str_cv, return_train_score=True,␣
 ↪scoring='roc_auc')
    # Fit on train data
    grid_clf.fit(tr_X, tr_y)
    return grid_clf

def plot_feature_importance(model, model_name, top_n = 10):
    '''
    Compute ROC curve and ROC area for each class
    Parameters:
    try_true: train true label
    try_pred: train predict probabilities value
    cvy_true: cv true label
    cvy_pred: cv predict probabilities value
    n_classes: number of unique classes

    Return:
    Plot of ROC Curve for train and cv data
    '''
    column_name = df_train.drop(['id','target'], axis=1).columns
    if model_name == 'log_model':
        feat_imp_coef = model.coef_.ravel()
    else:
        feat_imp_coef = model.feature_importances_
    temp = pd.DataFrame(data=np.column_stack((column_name, feat_imp_coef)),␣
↪columns=['col_name','coef'])
    temp = temp.sort_values(by='coef', ascending=False).reset_index()
    df = temp
    temp = temp[:top_n]
    plt.figure(figsize=(20,5))
    sns.barplot(data=temp, y='coef', x='col_name', order=temp['col_name'])
    plt.grid()
    plt.show()
    return df
```

[12]:

```
def forward_selection_model(model, top_n=10):
    top_column = []
    exist_score = 0
    for n in range(top_n):
        print('for {} feature'.format(n+1))
        flag = 0
        for i in tqdm.tqdm_notebook(range(tr_X.shape[1])):
            # Finding for first top feature
            if len(top_column) == 0:
                str_cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5,␣
↪random_state=42)
                score = cross_val_score(model,tr_X[:
↪,[i]],tr_y,cv=str_cv,scoring='roc_auc')
                if exist_score < np.mean(score):
                    top_current = i
                    exist_score = np.mean(score)
                    flag = 1
            # Excluded apart from top column
            elif i not in top_column:
                str_cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=5,␣
 ↪random_state=42)
                score = cross_val_score(model,tr_X[:,np.
 ↪concatenate((top_column,[i]))],tr_y,cv=str_cv,scoring='roc_auc')
                if exist_score < np.mean(score):
                    top_current = i
                    exist_score = np.mean(score)
                    flag = 1

        if flag == 1:
            print('Current top feature {} and score: {}'.
 ↪format(top_current,exist_score))
            print('Appended to top column')
            top_column.append(top_current)
            print(top_column)
        else:
            break
    return top_column
```

7 5.1 kNN

[10]:

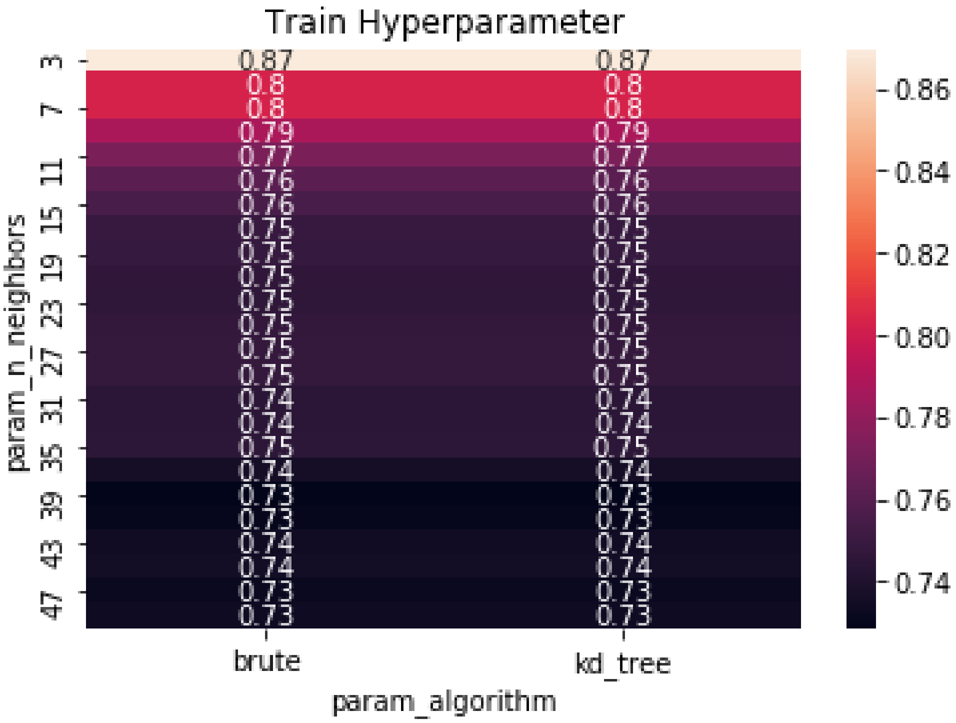
```
# Import KNN
from sklearn.neighbors import KNeighborsClassifier
```

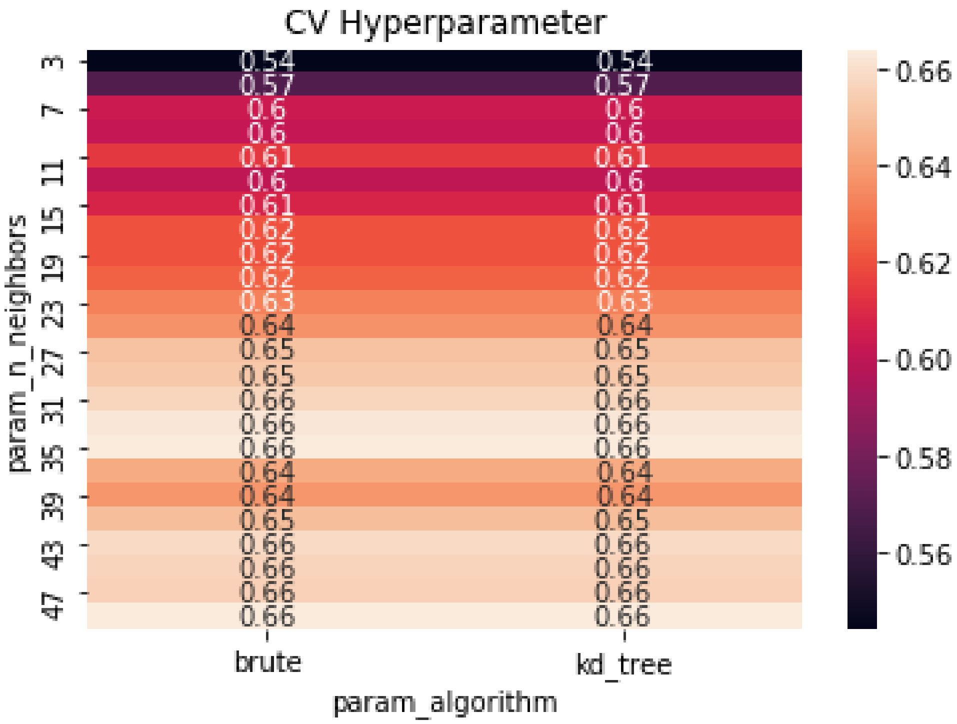
[11]:

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

# List of params
params = {'n_neighbors':np.arange(3,51,2).tolist(), 'algorithm': ['kd_tree',␣
 ↪'brute']}
# Instance of knn model
knn_model = KNeighborsClassifier()
# Call hyperparameter for find the best params as possible
knn_clf = hyperparameter_model(knn_model, params)
```

[13]:

```
cv_pvt = pd.pivot_table(pd.DataFrame(knn_clf.cv_results_),␣
 ↪values='mean_test_score', index='param_n_neighbors', \
                     columns='param_algorithm')
tr_pvt = pd.pivot_table(pd.DataFrame(knn_clf.cv_results_),␣
 ↪values='mean_train_score', index='param_n_neighbors', \
                     columns='param_algorithm')

plt.title('Train Hyperparameter')
sns.heatmap(tr_pvt, annot=True)
plt.show()
plt.title('CV Hyperparameter')
sns.heatmap(cv_pvt, annot=True)
plt.show()
```





```
[15]: print(knn_clf.best_params_)
```

```
{'algorithm': 'kd_tree', 'n_neighbors': 35}
```

```
[18]: clf = CalibratedClassifierCV(knn_clf, cv=3)
      clf.fit(tr_X,tr_y)
```

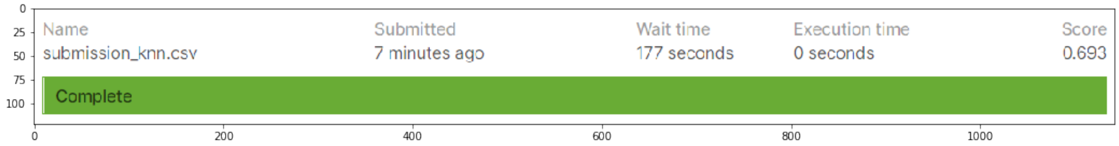
```
[18]: CalibratedClassifierCV(base_estimator=GridSearchCV(cv=RepeatedStratifiedKFold(n_
      repeats=5, n_splits=10, random_state=42),
                                                           error_score=nan,
      estimator=KNeighborsClassifier(algorithm='auto',
        leaf_size=30,
        metric='minkowski',
        metric_params=None,
        n_jobs=None,
        n_neighbors=5,
        p=2,
        weights='uniform'),
                                                           iid='deprecated',
                                                           n_jobs=None,
                                                           param_grid={'algorithm':
      ['kd_tree',
      'brute'],
                                                                    'n_neighbors':
[3,
5,
7,
9,
11,
13,
15,
17,
19,
21,
23,
25,
27,
29,
31,
33,
35,
37,
39,
41,
43,
45,
47,
49]},
                                                       pre_dispatch='2*n_jobs',
                                                       refit=True,
                                                       return_train_score=True,
                                                       scoring='roc_auc',
                                                       verbose=0),
                         cv=3, method='sigmoid')
```

8 5.1.1 Kaggle Score without top features

[19]:
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
knn_csv = clf.predict_proba(ts_X)[:,1]
knn_df = pd.DataFrame(np.column_stack((temp_id,knn_csv)),␣
 ↪columns=['id','target'])
knn_df['id'] = knn_df['id'].astype('int32')
knn_df.to_csv(data_dir+'/submission_knn.csv', index=False)
```

[21]:
```
image = plt.imread(data_dir+'/submission_knn.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

```
[21]: <matplotlib.image.AxesImage at 0x266dc241548>
```



Name	Submitted	Wait time	Execution time	Score
submission_knn.csv	7 minutes ago	177 seconds	0 seconds	0.693

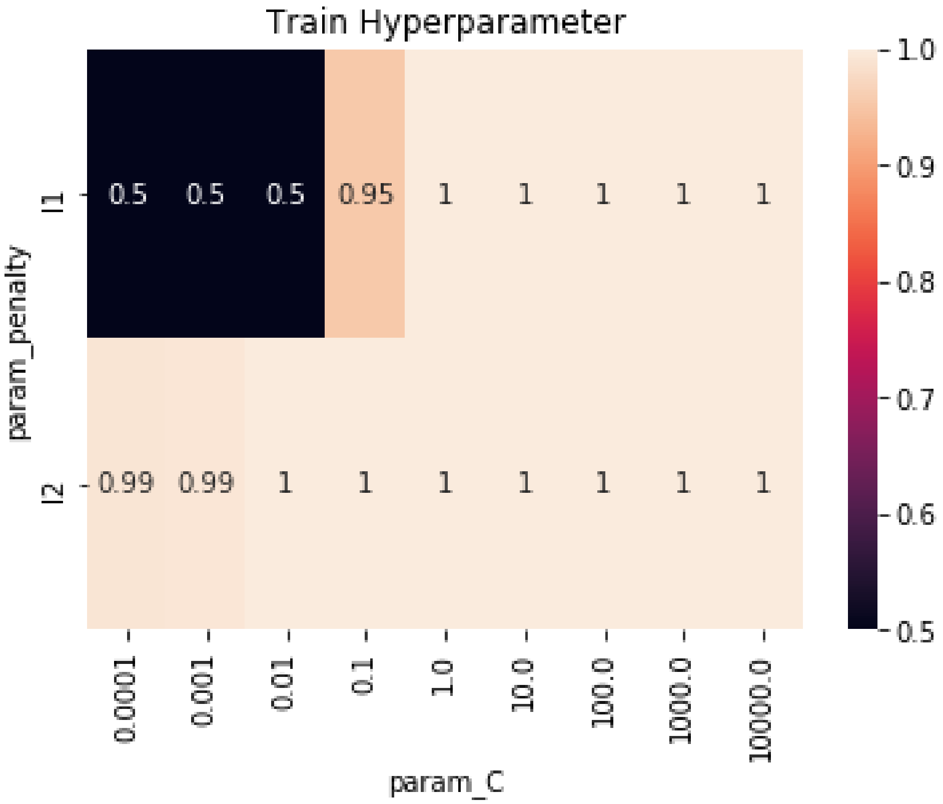
Complete

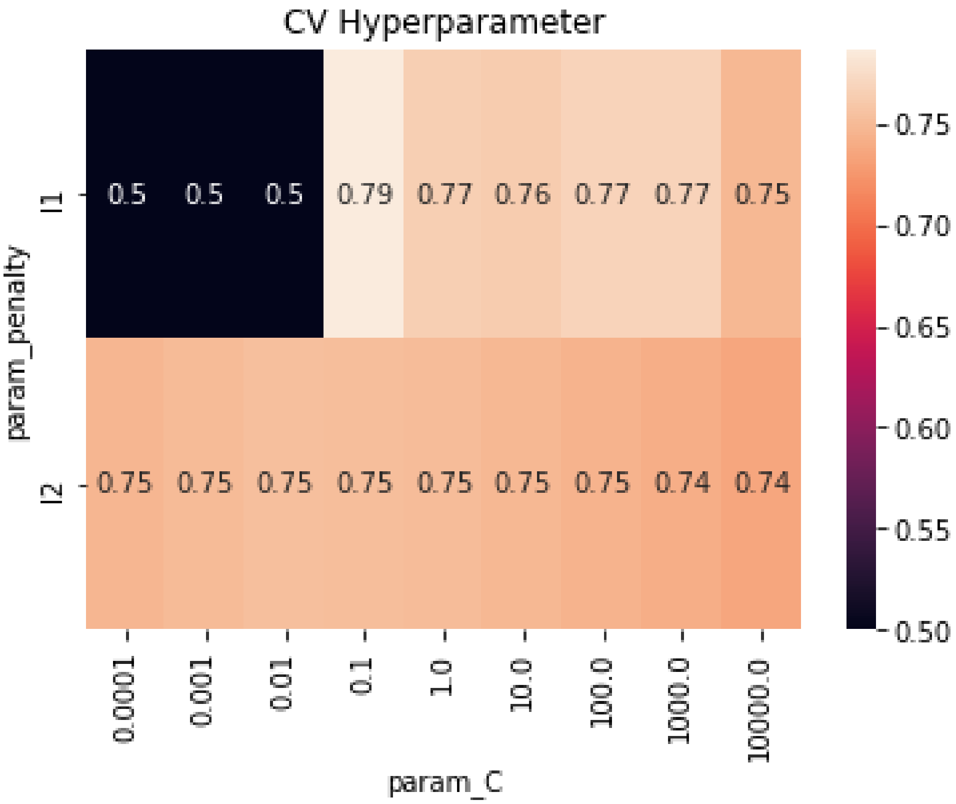
8.1 5.2 Logistic Regression

```
[22]: # Import Logistic Regression
      from sklearn.linear_model import LogisticRegression
```

```
[23]: # LogisticRegression (See Docs: https://scikit-learn.org/stable/modules/
       ↪generated/sklearn.linear_model.LogisticRegression.html)

      # List of hyperparameter that has to be tuned
      params = {'penalty':['l1', 'l2', 'elasticnet'], 'C':[10**i for i in␣
       ↪range(-4,5)], 'solver':['liblinear','sag']}
      # Instance of Logsitic Regression
      log_model = LogisticRegression(random_state=42, class_weight='balanced')
      # Call hyperparameter to get the best parameters of this model
      log_clf = hyperparameter_model(log_model, params)
```

```
[24]: cv_pvt = pd.pivot_table(pd.DataFrame(log_clf.cv_results_),␣
       ↪values='mean_test_score', index='param_penalty', \
                             columns='param_C')
      tr_pvt = pd.pivot_table(pd.DataFrame(log_clf.cv_results_),␣
       ↪values='mean_train_score', index='param_penalty', \
                             columns='param_C')

      plt.title('Train Hyperparameter')
      sns.heatmap(tr_pvt, annot=True)
      plt.show()

      plt.title('CV Hyperparameter')
      sns.heatmap(cv_pvt, annot=True)
      plt.show()
```


Train Hyperparameter
param_penalty
l1
l2
0.5
0.5
0.5
0.95
1
1
1
1
1
0.99
0.99
1
1
1
1
1
1
1
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
1.0
0.9
0.8
0.7
0.6
0.5



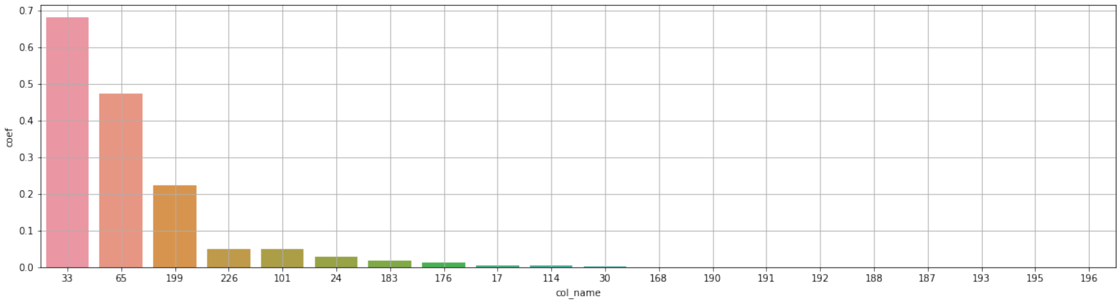
```
[25]: print(log_clf.best_params_)
```

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

```
[26]: log_model = LogisticRegression(**log_clf.best_params_, class_weight='balanced',␣
       ↪random_state=42)
      log_model.fit(tr_X, tr_y)
```

```
[26]: LogisticRegression(C=0.1, class_weight='balanced', dual=False,
                         fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                         max_iter=100, multi_class='auto', n_jobs=None, penalty='l1',
                         random_state=42, solver='liblinear', tol=0.0001, verbose=0,
                         warm_start=False)
```

```
[27]: df = plot_feature_importance(log_model, 'log_model', 20)
```



[28]:
```
clf = CalibratedClassifierCV(log_clf, cv=3)
clf.fit(tr_X,tr_y)
```

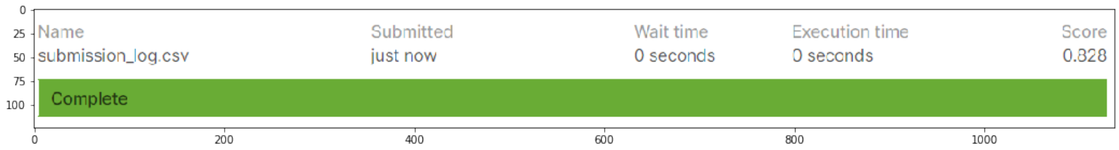
[28]:
```
CalibratedClassifierCV(base_estimator=GridSearchCV(cv=RepeatedStratifiedKFold(n_
repeats=5, n_splits=10, random_state=42),
                                                   error_score=nan,
estimator=LogisticRegression(C=1.0,
class_weight='balanced',
dual=False,
fit_intercept=True,
intercept_scaling=1,
l1_ratio=None,
max_iter=100,
multi_class='auto',
n_jobs=None,
penalty='l2',
random_state=42,
solver='lbfgs',
tol=0.0001,
verbose=0,
warm_start=False),
                                                   iid='deprecated',
                                                   n_jobs=None,
                                                   param_grid={'C': [0.0001,
                                                                     0.001,
                                                                     0.01, 0.1,
                                                                     1, 10, 100,
                                                                     1000,
                                                                     10000],
                                                               'penalty': ['l1',
                                                                           'l2',
'elasticnet'],
                                                               'solver':
['liblinear',
'sag']},
                                                   pre_dispatch='2*n_jobs',
                                                   refit=True,
                                                   return_train_score=True,
                                                   scoring='roc_auc',
                                                   verbose=0),
                       cv=3, method='sigmoid')
```

9 5.2.1 Kaggle Score without top features

[29]:
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
log_csv = clf.predict_proba(ts_X)[:,1]
log_df = pd.DataFrame(np.column_stack((temp_id,log_csv)),␣
 ↪columns=['id','target'])
log_df['id'] = log_df['id'].astype('int32')
log_df.to_csv(data_dir+'/submission_log.csv', index=False)
```

[30]:
```
image = plt.imread(data_dir+'/submission_log.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[30]:
```
<matplotlib.image.AxesImage at 0x266dedfdd08>
```



Name	Submitted	Wait time	Execution time	Score
submission_log.csv	just now	0 seconds	0 seconds	0.828

Complete

10 5.2.2 Kaggle Score Using top 10 features based on Logistic Regression Model

[31]:
```
top_column = forward_selection_model(log_model)
```

```
for 1 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 33 and score: 0.7347222222222222
Appended to top column
[33]
for 2 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 65 and score: 0.7876388888888887
Appended to top column
[33, 65]
for 3 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 217 and score: 0.8005555555555557
Appended to top column
[33, 65, 217]
for 4 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 117 and score: 0.8156944444444443
Appended to top column
[33, 65, 217, 117]
for 5 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 91 and score: 0.8325
Appended to top column
[33, 65, 217, 117, 91]
for 6 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 73 and score: 0.8423611111111111
Appended to top column
[33, 65, 217, 117, 91, 73]
for 7 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 16 and score: 0.8493055555555555
Appended to top column
[33, 65, 217, 117, 91, 73, 16]
for 8 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 226 and score: 0.8527777777777778
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226]
for 9 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 82 and score: 0.8559722222222222
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226, 82]
for 10 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 90 and score: 0.8590277777777778
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226, 82, 90]
```

[37]:

```
# Store the top column name into set
log_top10_feat = set(top_column)
# Fit Logistic Regression on top features only
log_model_top = LogisticRegression(**log_clf.best_params_,␣
 ↪class_weight='balanced', random_state=42)
log_model_top.fit(tr_X[:,top_column],tr_y)
# Calibrate it in top features only
clf = CalibratedClassifierCV(log_model_top, cv=3)
clf.fit(tr_X[:,top_column],tr_y)
# Predict the probabilities of 1
ts_pred = clf.predict_proba(ts_X[:,top_column])[:,1]
```

[34]:

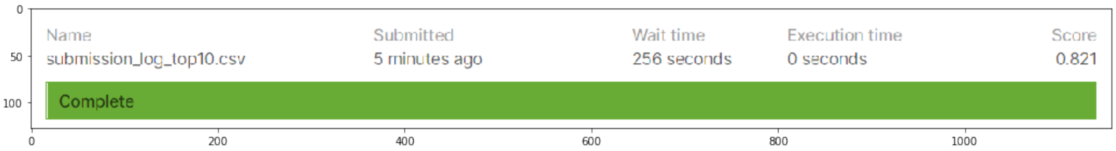
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
df_tspred = pd.DataFrame(np.column_stack((temp_id,ts_pred)),␣
 ↪columns=['id','target'])
df_tspred['id'] = df_tspred['id'].astype('int32')
df_tspred.to_csv(data_dir+'/submission_log_top10.csv', index=False)
```

[36]:

```
image = plt.imread(data_dir+'/log_csv_top10.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[36]:

```
<matplotlib.image.AxesImage at 0x266e864dd08>
```



Name	Submitted	Wait time	Execution time	Score
submission_log_top10.csv	5 minutes ago	256 seconds	0 seconds	0.821

Complete

11 5.2.3 Kaggle Score Using top 20 features based on Logistic Regression Model

[39]:
```
top_column = forward_selection_model(log_model,20)
```

```
for 1 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 33 and score: 0.7347222222222222
Appended to top column
[33]
for 2 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 65 and score: 0.7876388888888887
Appended to top column
[33, 65]
for 3 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 217 and score: 0.8005555555555557
Appended to top column
[33, 65, 217]
for 4 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 117 and score: 0.8156944444444443
Appended to top column
[33, 65, 217, 117]
for 5 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 91 and score: 0.8325
Appended to top column
[33, 65, 217, 117, 91]
for 6 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 73 and score: 0.8423611111111111
Appended to top column
[33, 65, 217, 117, 91, 73]
for 7 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 16 and score: 0.8493055555555555
Appended to top column
[33, 65, 217, 117, 91, 73, 16]
for 8 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 226 and score: 0.8527777777777778
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226]
for 9 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 82 and score: 0.8559722222222222
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226, 82]
for 10 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 90 and score: 0.8590277777777778
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226, 82, 90]
for 11 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 45 and score: 0.8591666666666669
Appended to top column
[33, 65, 217, 117, 91, 73, 16, 226, 82, 90, 45]
for 12 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

[40]:

```
# Store the top column name into set
log_top20_feat = set(top_column)
# Fit Logistic Regression on top features only
log_model_top = LogisticRegression(**log_clf.best_params_,␣
 ↪class_weight='balanced', random_state=42)
log_model_top.fit(tr_X[:,top_column],tr_y)
# Calibrate it in top features only
clf = CalibratedClassifierCV(log_model_top, cv=3)
clf.fit(tr_X[:,top_column],tr_y)
# Predict the probabilities of 1
ts_pred = clf.predict_proba(ts_X[:,top_column])[:,1]
```

[41]:

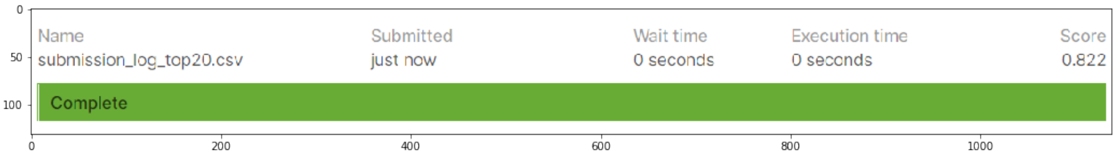
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
df_tspred = pd.DataFrame(np.column_stack((temp_id,ts_pred)),␣
 ↪columns=['id','target'])
df_tspred['id'] = df_tspred['id'].astype('int32')
df_tspred.to_csv(data_dir+'/submission_log_top20.csv', index=False)
```

[42]:

```
image = plt.imread(data_dir+'/log_csv_top20.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[42]:

```
<matplotlib.image.AxesImage at 0x266e9ff0b48>
```



Name	Submitted	Wait time	Execution time	Score
submission_log_top20.csv	just now	0 seconds	0 seconds	0.822

Complete

12 5.3 SVC

[43]:

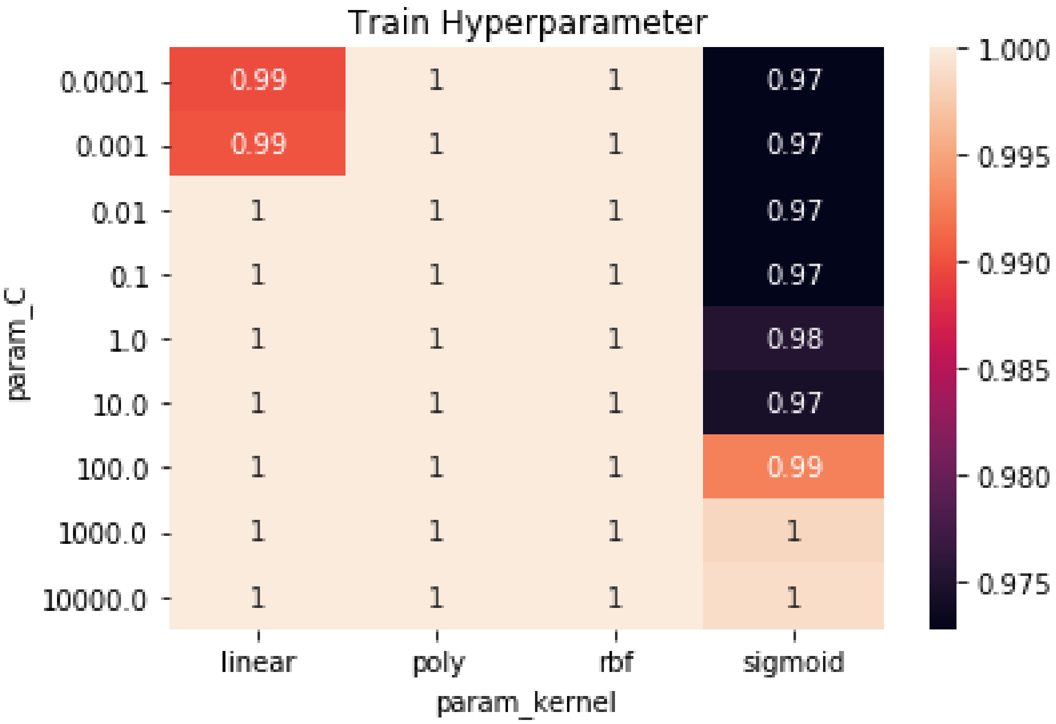
```
# Import SVC
from sklearn.svm import SVC
```

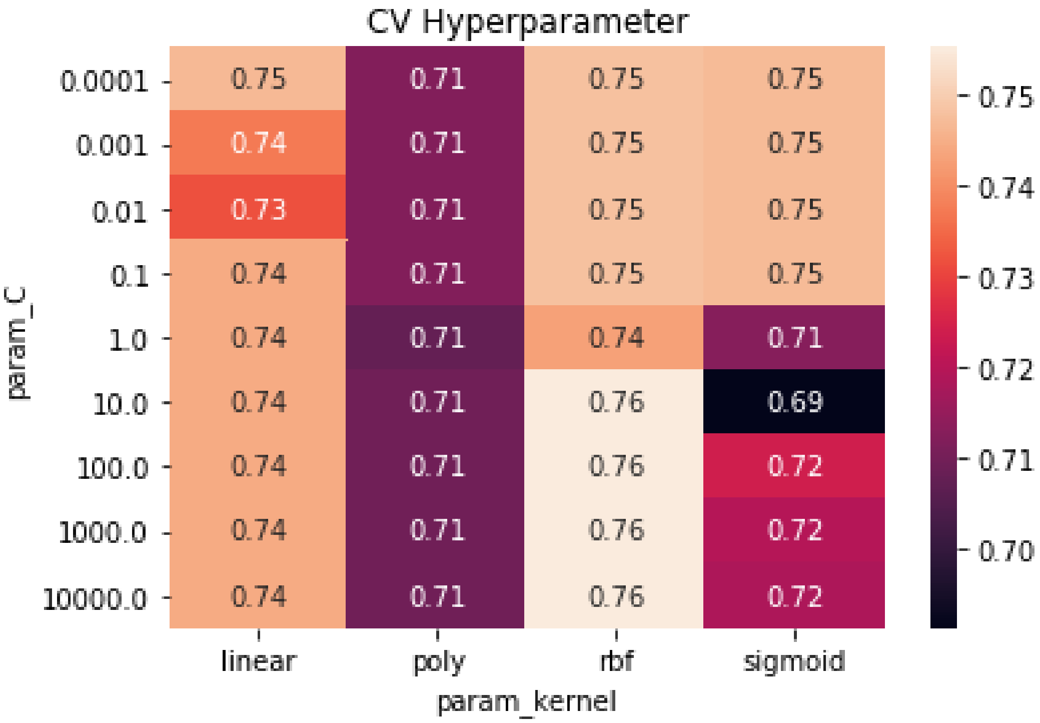
[44]:

```
# SVC (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.svm.
↪SVC.html)

# List of hyperparameter that has to be tuned
params = {'C':[10**i for i in range(-4,5)], 'kernel':
↪['linear','poly','sigmoid','rbf']}
# Instance of SVC
svc_model = SVC(class_weight='balanced', random_state=42, probability=True)
# Call hyperparameter to find the best parameters
svc_clf = hyperparameter_model(svc_model, params)
```

[45]:

```
cv_pvt = pd.pivot_table(pd.DataFrame(svc_clf.cv_results_),␣
↪values='mean_test_score', index='param_C', \
                        columns='param_kernel')
tr_pvt = pd.pivot_table(pd.DataFrame(svc_clf.cv_results_),␣
↪values='mean_train_score', index='param_C', \
                        columns='param_kernel')

plt.title('Train Hyperparameter')
sns.heatmap(tr_pvt, annot=True)
plt.show()

plt.title('CV Hyperparameter')
sns.heatmap(cv_pvt, annot=True)
plt.show()
```


Train Hyperparameter


CV Hyperparameter

```
[46]: print(svc_clf.best_params_)
```

```
{'C': 10, 'kernel': 'rbf'}
```

```
[47]: svc_model = SVC(**svc_clf.best_params_, class_weight='balanced',␣
       ↪random_state=42, probability=True)
      svc_model.fit(tr_X, tr_y)

      clf = CalibratedClassifierCV(svc_clf, cv=3)
      clf.fit(tr_X,tr_y)
```

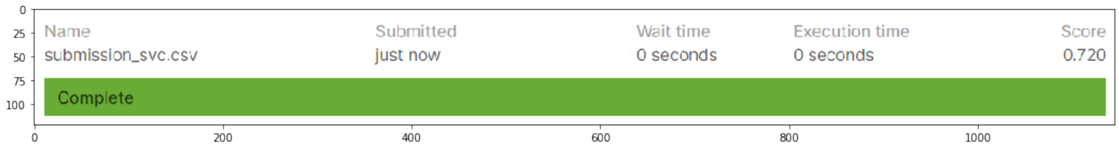
```
[47]: CalibratedClassifierCV(base_estimator=GridSearchCV(cv=RepeatedStratifiedKFold(n_
      repeats=5, n_splits=10, random_state=42),
                                                         error_score=nan,
                                                         estimator=SVC(C=1.0,
      break_ties=False,
                                                                       cache_size=200,
      class_weight='balanced',
                                                                       coef0=0.0,
      decision_function_shape='ovr',
                                                                       degree=3,
                                                                       gamma='scale',
                                                                       kernel='rbf',
                                                                       max_iter=-1,
      probability=True,
      random_state=42,
                                                                       shrinking=True,
                                                                       tol=0.001,
                                                                       verbose=False),
                                                         iid='deprecated',
                                                         n_jobs=None,
                                                         param_grid={'C': [0.0001,
                                                                           0.001,
                                                                           0.01, 0.1,
                                                                           1, 10, 100,
                                                                           1000,
                                                                           10000],
                                                                     'kernel':
      ['linear',
      'poly',
      'sigmoid',
      'rbf']},
                                                         pre_dispatch='2*n_jobs',
                                                         refit=True,
                                                         return_train_score=True,
                                                         scoring='roc_auc',
                                                         verbose=0),
                                 cv=3, method='sigmoid')
```

13 5.3.1 Kaggle Score without top features

```
[48]: # Create a submssion format to make submission in Kaggle
      temp_id = df_test['id']
      svc_csv = clf.predict_proba(ts_X)[:,1]
      svc_df = pd.DataFrame(np.column_stack((temp_id,svc_csv)),␣
       ↪columns=['id','target'])
      svc_df['id'] = svc_df['id'].astype('int32')
      svc_df.to_csv(data_dir+'/submission_svc.csv', index=False)
```

```
[49]: image = plt.imread(data_dir+'/submission_svc.png')
      plt.figure(figsize=(18,5))
      plt.imshow(image)
```

```
[49]: <matplotlib.image.AxesImage at 0x266e9bc10c8>
```



Name	Submitted	Wait time	Execution time	Score
submission_svc.csv	just now	0 seconds	0 seconds	0.720

Complete

14 5.3.2 Kaggle Score Using top 10 features based on SVC Model

```
[50]: top_column = forward_selection_model(svc_model)
```

```
for 1 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 33 and score: 0.7088888888888889
Appended to top column
[33]
for 2 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 167 and score: 0.7579166666666667
Appended to top column
```

```
[33, 167]
for 3 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 199 and score: 0.7611111111111112
Appended to top column
[33, 167, 199]
for 4 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 65 and score: 0.7808333333333333
Appended to top column
[33, 167, 199, 65]
for 5 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 224 and score: 0.8034722222222223
Appended to top column
[33, 167, 199, 65, 224]
for 6 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 1 and score: 0.8180555555555555
Appended to top column
[33, 167, 199, 65, 224, 1]
for 7 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 208 and score: 0.8381944444444445
Appended to top column
[33, 167, 199, 65, 224, 1, 208]
for 8 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 201 and score: 0.867638888888889
Appended to top column
```

```
[33, 167, 199, 65, 224, 1, 208, 201]
for 9 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 253 and score: 0.8869444444444444
Appended to top column
[33, 167, 199, 65, 224, 1, 208, 201, 253]
for 10 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

[51]:

```
# Store the top column name into set
svc_top10_feat = set(top_column)
# Fit SVC on top features only
svc_model_top = SVC(**svc_clf.best_params_, class_weight='balanced',␣
 ↪random_state=42, probability=True)
svc_model_top.fit(tr_X[:,top_column],tr_y)
# Calibrate it in top features only
clf = CalibratedClassifierCV(svc_model_top, cv=3)
clf.fit(tr_X[:,top_column],tr_y)
# Predict the probabilities of 1
ts_pred = clf.predict_proba(ts_X[:,top_column])[:,1]
```

[52]:

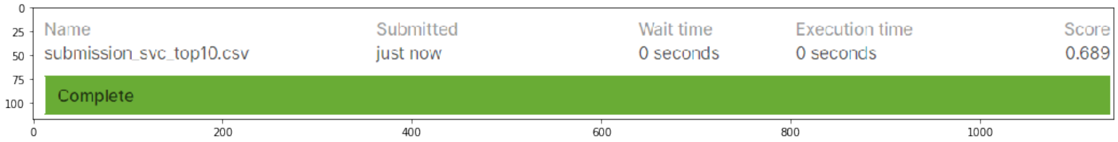
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
df_tspred = pd.DataFrame(np.column_stack((temp_id,ts_pred)),␣
 ↪columns=['id','target'])
df_tspred['id'] = df_tspred['id'].astype('int32')
df_tspred.to_csv(data_dir+'/submission_svc_top10.csv', index=False)
```

[54]:

```
image = plt.imread(data_dir+'/svc_csv_top10.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[54]:

```
<matplotlib.image.AxesImage at 0x266e8512f88>
```



Name	Submitted	Wait time	Execution time	Score
submission_svc_top10.csv	just now	0 seconds	0 seconds	0.689

Complete

15 5.4 RandomForest

[55]:

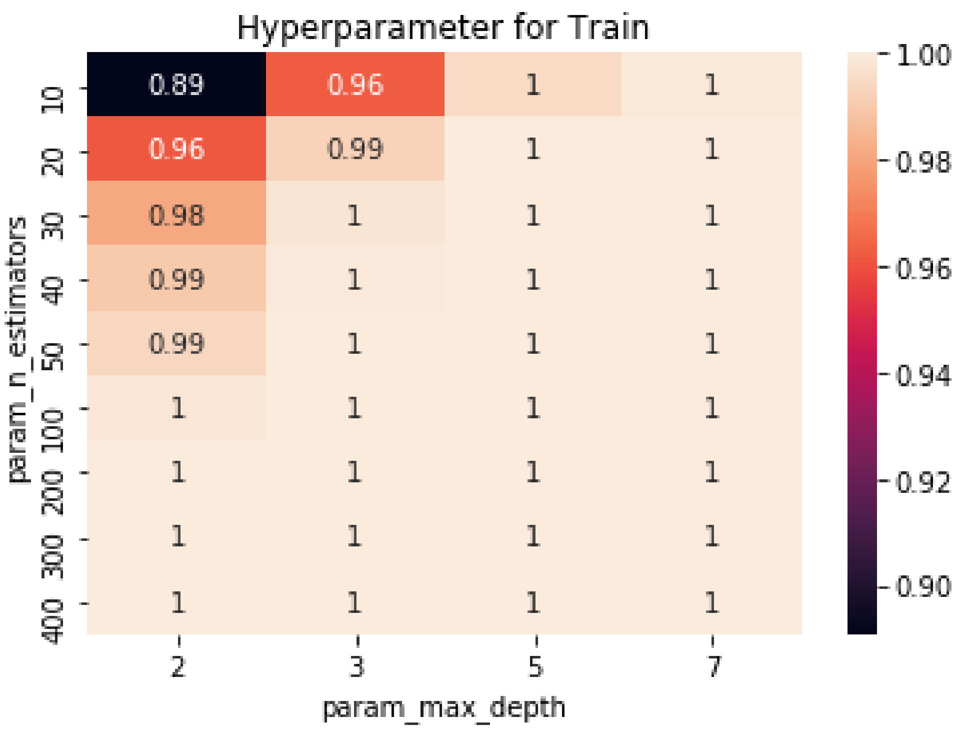
```
# Impoer Random Forest
from sklearn.ensemble import RandomForestClassifier
```

[56]:

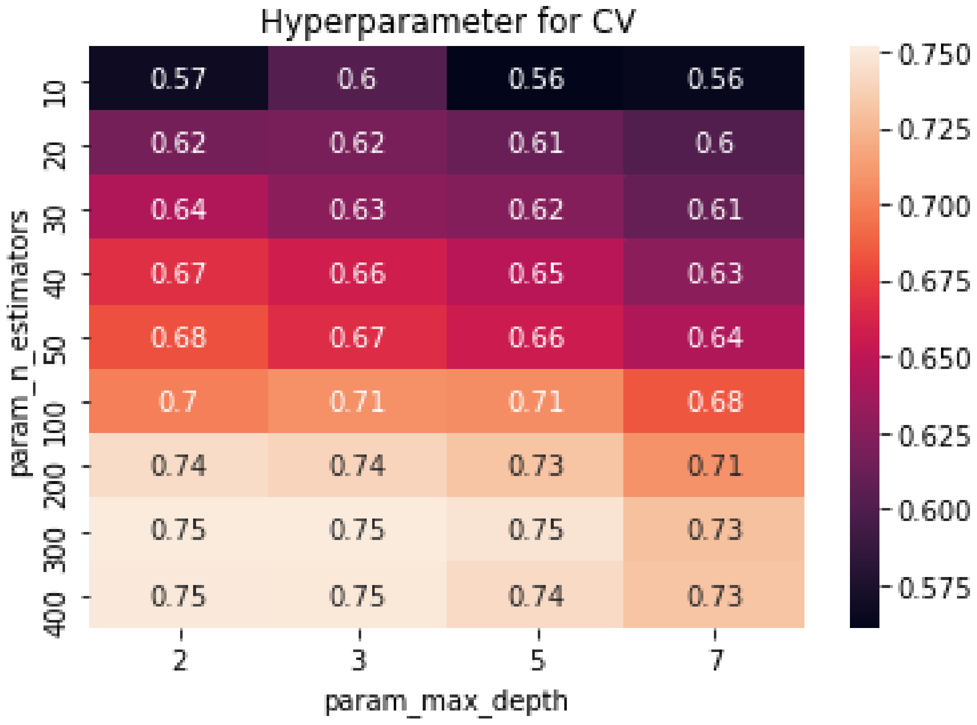
```
# RandomForest (See Docs: https://scikit-learn.org/stable/modules/generated/
 ↪sklearn.ensemble.RandomForestClassifier.html)

# List of hyperparameter that has t be tuned
params = {'n_estimators':[10,20,30,40,50,100,200,300,400],'max_depth':[2,3,5,7]}
# Instance of randomforest
rf_model = RandomForestClassifier(random_state=42)
# Perform GridSearchCV to find best parameters
rf_clf = hyperparameter_model(rf_model, params)
```

[57]:

```
# Ref: https://stackoverflow.com/questions/48791709/
 ↪how-to-plot-a-heat-map-on-pivot-table-after-grid-search

# Plotting of hyperparameter of train and cv score
pvt_tr = pd.pivot_table(pd.DataFrame(rf_clf.cv_results_),␣
 ↪values='mean_train_score', index='param_n_estimators',␣
 ↪columns='param_max_depth')
pvt_cv = pd.pivot_table(pd.DataFrame(rf_clf.cv_results_),␣
 ↪values='mean_test_score', index='param_n_estimators',␣
 ↪columns='param_max_depth')
plt.figure(1)
plt.title('Hyperparameter for Train')
sns.heatmap(pvt_tr, annot=True)
plt.figure(2)
plt.title('Hyperparameter for CV')
sns.heatmap(pvt_cv, annot=True)
plt.show()
```



Hyperparameter for Train



Hyperparameter for CV

[58]:

```python
print(rf_clf.best_params_)
```

```
{'max_depth': 2, 'n_estimators': 300}
```

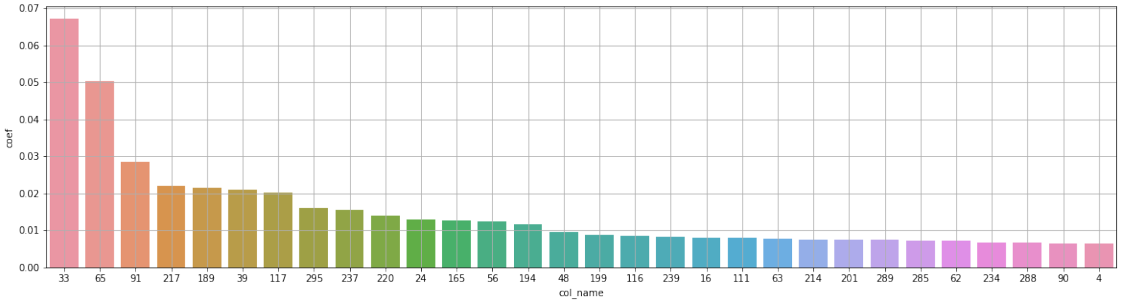
[59]:

```python
# Instance of randomforest with best parameters
rf_model = RandomForestClassifier(**rf_clf.best_params_, random_state=42)
# Fit the model
rf_model.fit(tr_X,tr_y)
# Calibrate the model
clf = CalibratedClassifierCV(rf_clf, cv=3)
clf.fit(tr_X, tr_y)
```

[59]:

```
CalibratedClassifierCV(base_estimator=GridSearchCV(cv=RepeatedStratifiedKFold(n_
repeats=5, n_splits=10, random_state=42),
                                                    error_score=nan,
estimator=RandomForestClassifier(bootstrap=True,
    ccp_alpha=0.0,
    class_weight=None,
    criterion='gini',
    max_depth=None,
    max_features='auto',
    max_leaf_nodes=None,
    max_samples=None,
    min_impurity_decrease=0.0,
    min_impurity_split=N…
    min_samples_split=2,
    min_weight_fraction_leaf=0.0,
    n_estimators=100,
    n_jobs=None,
    oob_score=False,
    random_state=42,
    verbose=0,
    warm_start=False),
                                                    iid='deprecated',
                                                    n_jobs=None,
                                                    param_grid={'max_depth': [2,
                                                                              3,
                                                                              5,
                                                                              7],
                                                                'n_estimators':
[10,
20,
30,
40,
50,
100,
200,
300,
400]},
                                                    pre_dispatch='2*n_jobs',
                                                    refit=True,
                                                    return_train_score=True,
                                                    scoring='roc_auc',
                                                    verbose=0),
                       cv=3, method='sigmoid')
```

[60]:
```
# Plot the feature importance based on this model
df = plot_feature_importance(rf_model, 'rf',30)
```

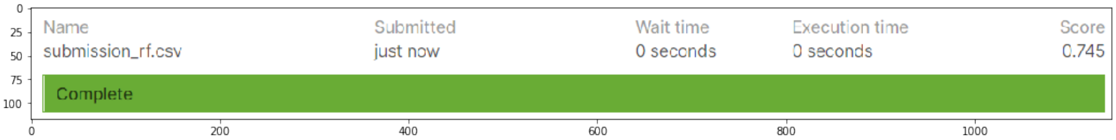


16 5.4.1 Kaggle Score without top features

[61]:
```
# Create a submission file format to submit in kaggle
temp_id = df_test['id']
rf_csv = clf.predict_proba(ts_X)[:,1]
rf_df = pd.DataFrame(np.column_stack((temp_id,rf_csv)), columns=['id','target'])
rf_df['id'] = rf_df['id'].astype('int32')
rf_df.to_csv(data_dir+'/submission_rf.csv', index=False)
```

[62]:
```
image = plt.imread(data_dir+'/submission_rf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[62]:
```
<matplotlib.image.AxesImage at 0x266e9c5f348>
```



Name	Submitted	Wait time	Execution time	Score
submission_rf.csv	just now	0 seconds	0 seconds	0.745

Complete

17 5.4.2 Kaggle Score Using top 10 features based on RandomForest Model

[63]:

```
top_column = forward_selection_model(rf_model)
```

```
for 1 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 33 and score: 0.7097222222222223
Appended to top column
[33]
for 2 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 65 and score: 0.765
Appended to top column
[33, 65]
for 3 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 91 and score: 0.7863888888888888
Appended to top column
[33, 65, 91]
for 4 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 199 and score: 0.788888888888889
Appended to top column
[33, 65, 91, 199]
for 5 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 217 and score: 0.805138888888889
Appended to top column
[33, 65, 91, 199, 217]
for 6 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

```
Current top feature 214 and score: 0.8094444444444444
Appended to top column
[33, 65, 91, 199, 217, 214]
for 7 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 189 and score: 0.8162500000000001
Appended to top column
[33, 65, 91, 199, 217, 214, 189]
for 8 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 132 and score: 0.8290277777777777
Appended to top column
[33, 65, 91, 199, 217, 214, 189, 132]
for 9 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

[64]:
```python
# Store the top column name into set
rf_top10_feat = set(top_column)
```

[65]:
```python
# Fit RF on top features only
rf_model_top = RandomForestClassifier(**rf_clf.best_params_, random_state=42)
rf_model_top.fit(tr_X[:,top_column],tr_y)
# Calibrate it in top features only
clf = CalibratedClassifierCV(rf_model_top, cv=3)
clf.fit(tr_X[:,top_column],tr_y)
# Predict the probabilities of 1
ts_pred = clf.predict_proba(ts_X[:,top_column])[:,1]
```

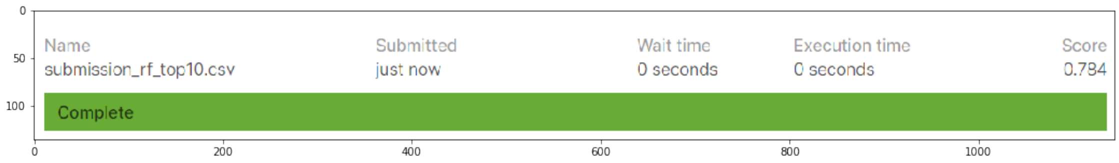
[66]:
```python
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
df_tspred = pd.DataFrame(np.column_stack((temp_id,ts_pred)),␣
 ↪columns=['id','target'])
df_tspred['id'] = df_tspred['id'].astype('int32')
df_tspred.to_csv(data_dir+'/submission_rf_top10.csv', index=False)
```

[67]:

```
image = plt.imread(data_dir+'/rf_csv_top10.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[67]:

```
<matplotlib.image.AxesImage at 0x266e9e12a08>
```



Name	Submitted	Wait time	Execution time	Score
submission_rf_top10.csv	just now	0 seconds	0 seconds	0.784

Complete

18 5.5 XGBoost

[69]:

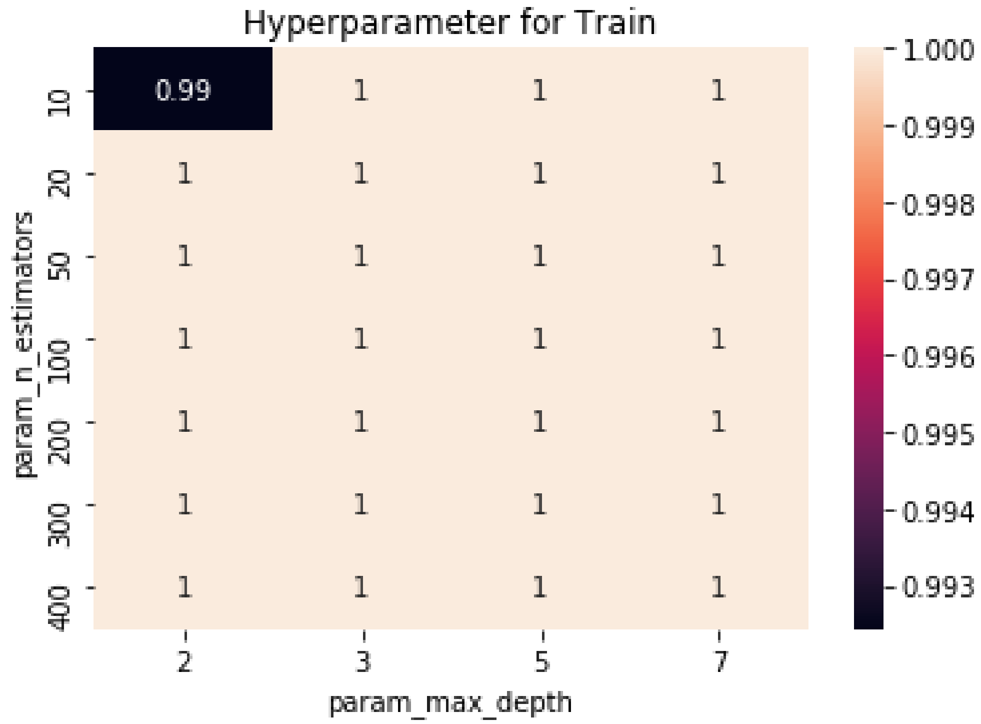
```
# Import Xgboost
from xgboost import XGBClassifier
```

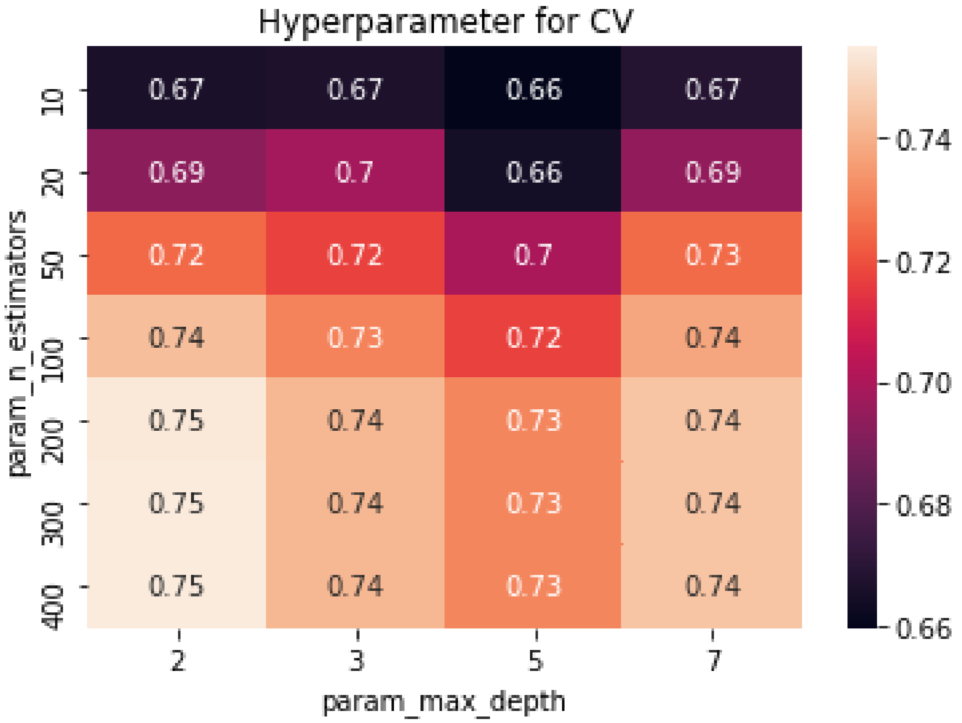
[70]:

```
# Xgboost (See Docs: https://xgboost.readthedocs.io/en/latest/python/python_api.
 ↪html)

# List of hyperparameter that has to be tuned
params = {'max_depth':[2,3,5,7], 'n_estimators':[10,20,50,100,200,300,400]}
# Instance of XGBoost Model
xgb_model = XGBClassifier(scale_pos_weight=0.5)
# Call hyperparameter to find the best parameters
xgb_clf = hyperparameter_model(xgb_model, params)
```

[71]:

```
# Ref: https://stackoverflow.com/questions/48791709/
 ↪how-to-plot-a-heat-map-on-pivot-table-after-grid-search

# Plotting of hyperparameter of train and cv score
pvt_tr = pd.pivot_table(pd.DataFrame(xgb_clf.cv_results_),␣
 ↪values='mean_train_score', index='param_n_estimators',␣
 ↪columns='param_max_depth')
pvt_cv = pd.pivot_table(pd.DataFrame(xgb_clf.cv_results_),␣
 ↪values='mean_test_score', index='param_n_estimators',␣
 ↪columns='param_max_depth')
plt.figure(1)
plt.title('Hyperparameter for Train')
sns.heatmap(pvt_tr, annot=True)
plt.figure(2)
plt.title('Hyperparameter for CV')
sns.heatmap(pvt_cv, annot=True)
plt.show()
```

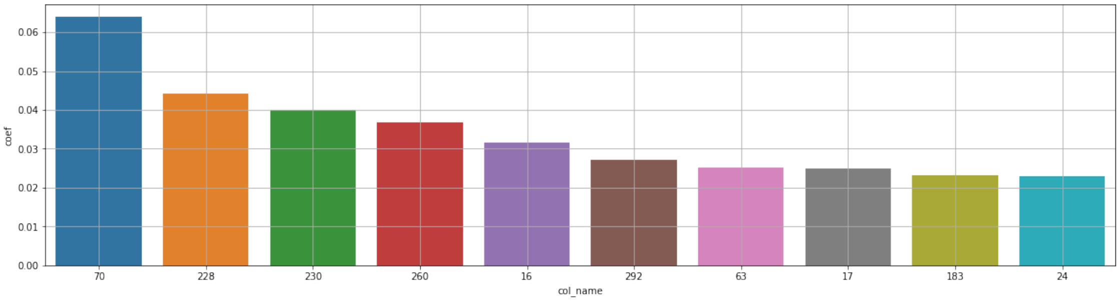

Hyperparameter for Train
param_n_estimators
10
20
50
100
200
300
400
param_max_depth
2
3
5
7
0.99 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1.000
0.999
0.998
0.997
0.996
0.995
0.994
0.993


Hyperparameter for CV
param_n_estimators
10
20
50
100
200
300
400
param_max_depth
2
3
5
7
0.67 0.67 0.66 0.67
0.69 0.7 0.66 0.69
0.72 0.72 0.7 0.73
0.74 0.73 0.72 0.74
0.75 0.74 0.73 0.74
0.75 0.74 0.73 0.74
0.75 0.74 0.73 0.74
0.74
0.72
0.70
0.68
0.66

[72]:
```
print(xgb_clf.best_params_)
```

```
{'max_depth': 2, 'n_estimators': 300}
```

[74]:
```
# Instance of randomforest with best parameters
xgb_model = XGBClassifier(**xgb_clf.best_params_, random_state=42, scale_pos_weight=0.5)
# Fit the model
xgb_model.fit(tr_X,tr_y)

# Instance of XGBoost model with best parameters
df = plot_feature_importance(xgb_model, 'xgb',10)
```



[75]:
```
# Calibrate the model
clf = CalibratedClassifierCV(xgb_clf, cv=3)
clf.fit(tr_X, tr_y)
```

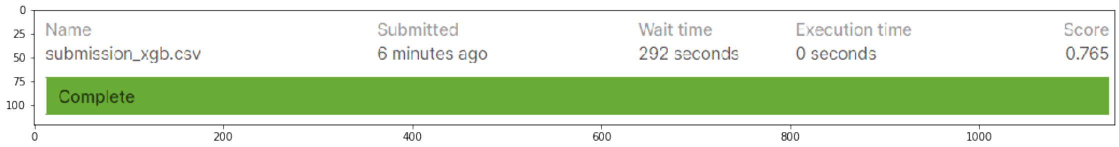
[75]:
```
CalibratedClassifierCV(base_estimator=GridSearchCV(cv=RepeatedStratifiedKFold(n_
repeats=5, n_splits=10, random_state=42),
                                                   error_score=nan,
estimator=XGBClassifier(base_score=None,
booster=None,
colsample_bylevel=None,
colsample_bynode=None,
colsample_bytree=None,
gamma=None,
gpu_id=None,
importance_type='gain',
interaction_constraints=None,
learning_rate=None,
                                                                           m…
random_state=None,
reg_alpha=None,
reg_lambda=None,
scale_pos_weight=0.5,
subsample=None,
tree_method=None,
validate_parameters=False,
verbosity=None),
                                                   iid='deprecated',
                                                   n_jobs=None,
                                                   param_grid={'max_depth': [2,
                                                                             3,
                                                                             5,
                                                                             7],
                                                               'n_estimators':
[10,
20,
50,
100,
200,
300,
400]},
                                                   pre_dispatch='2*n_jobs',
                                                   refit=True,
                                                   return_train_score=True,
                                                   scoring='roc_auc',
                                                   verbose=0),
                       cv=3, method='sigmoid')
```

19 5.5.1 Kaggle Score without top features

[76]:
```
# Create submission file format to submit in Kaggle
temp_id = df_test['id']
xgb_csv = clf.predict_proba(ts_X)[:,1]
xgb_df = pd.DataFrame(np.column_stack((temp_id,xgb_csv)),␣
 ↪columns=['id','target'])
xgb_df['id'] = xgb_df['id'].astype('int32')
xgb_df.to_csv(data_dir+'/submission_xgb.csv', index=False)
```

[77]:
```
image = plt.imread(data_dir+'/submission_xgb.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[77]:
```
<matplotlib.image.AxesImage at 0x266f1803908>
```



Name	Submitted	Wait time	Execution time	Score
submission_xgb.csv	6 minutes ago	292 seconds	0 seconds	0.765

Complete

20 5.5.2 Using top 10 features of this model

```
[78]: top_column = forward_selection_model(xgb_model)
```

```
for 1 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 91 and score: 0.6425
Appended to top column
[91]
for 2 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 120 and score: 0.7095833333333332
Appended to top column
[91, 120]
for 3 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 33 and score: 0.7473611111111111
Appended to top column
[91, 120, 33]
for 4 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 108 and score: 0.7788888888888887
Appended to top column
[91, 120, 33, 108]
for 5 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 141 and score: 0.8011111111111111
Appended to top column
[91, 120, 33, 108, 141]
for 6 feature
```

```
HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 246 and score: 0.8295833333333335
Appended to top column
[91, 120, 33, 108, 141, 246]
for 7 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 146 and score: 0.8491666666666668
Appended to top column
[91, 120, 33, 108, 141, 246, 146]
for 8 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 95 and score: 0.868888888888889
Appended to top column
[91, 120, 33, 108, 141, 246, 146, 95]
for 9 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))



Current top feature 230 and score: 0.8731944444444444
Appended to top column
[91, 120, 33, 108, 141, 246, 146, 95, 230]
for 10 feature

HBox(children=(FloatProgress(value=0.0, max=300.0), HTML(value='')))
```

[79]:

```
# Store the top column name into set
xgb_top10_feat = set(top_column)
```

[80]:

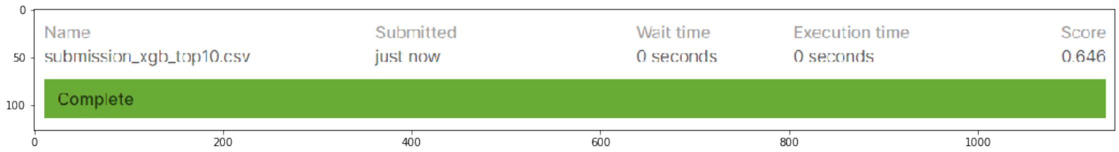
```
# Fit RF on top features only
xgb_model_top = XGBClassifier(**xgb_clf.best_params_, random_state=42,␣
 ↪scale_pos_weight=0.5)
xgb_model_top.fit(tr_X[:,top_column],tr_y)
# Calibrate it in top features only
clf = CalibratedClassifierCV(xgb_model_top, cv=3)
clf.fit(tr_X[:,top_column],tr_y)
# Predict the probabilities of 1
```

```
ts_pred = clf.predict_proba(ts_X[:,top_column])[:,1]
```

[81]:
```
# Create a submssion format to make submission in Kaggle
temp_id = df_test['id']
df_tspred = pd.DataFrame(np.column_stack((temp_id,ts_pred)),␣
↪columns=['id','target'])
df_tspred['id'] = df_tspred['id'].astype('int32')
df_tspred.to_csv(data_dir+'/submission_xgb_top10.csv', index=False)
```

[85]:
```
image = plt.imread(data_dir+'/xgb_csv_top10.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[85]:
```
<matplotlib.image.AxesImage at 0x266eaa06a88>
```



Name	Submitted	Wait time	Execution time	Score
submission_xgb_top10.csv	just now	0 seconds	0 seconds	0.646

Complete

21 5.6 Stacking Classifier

[92]:
```
# Combined all top features of all models
# Ref: https://www.geeksforgeeks.org/union-function-python/
comb_top_feat = list(log_top20_feat.union(svc_top10_feat, rf_top10_feat,␣
↪xgb_top10_feat))
```

[86]:
```
# Import Stacking Classifier
from mlxtend.classifier import StackingClassifier
```

[93]:
```
# StackClassifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
↪classifier/StackingClassifier/#methods)

# Classifier 1: Logistic Regression with best params
clf1 = LogisticRegression(C = 0.1, penalty = 'l1', solver = 'liblinear',␣
↪class_weight='balanced', random_state=42)
clf1.fit(tr_X,tr_y)
clf1 = CalibratedClassifierCV(clf1, cv=3)

# Classifier 2: SVC with best params
clf2 = SVC(C=10, kernel='rbf', random_state=42, class_weight='balanced',␣
↪probability=True)
clf2.fit(tr_X,tr_y)
```

```
clf2 = CalibratedClassifierCV(clf2, cv=3)

# Classifier 3: XGBoost with best params
clf3 = XGBClassifier(max_depth=2, n_estimators=300, scale_pos_weight=0.5)
clf3.fit(tr_X,tr_y)
clf3 = CalibratedClassifierCV(clf3, cv=3)

# Classifier 4: RF with best params
clf4 = RandomForestClassifier(max_depth=2, n_estimators=300)
clf4.fit(tr_X,tr_y)
clf4 = CalibratedClassifierCV(clf4, cv=3)

# Stack Classifier
sclf = StackingClassifier(classifiers=[clf1,clf2,clf3,clf4],␣
 ↪meta_classifier=clf1, use_probas=True)

# Fit the model
sclf.fit(tr_X, tr_y)
```

[93]:

```
StackingClassifier(average_probas=False,
classifiers=[CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.1,
         class_weight='balanced',
         dual=False,
         fit_intercept=True,
         intercept_scaling=1,
         l1_ratio=None,
         max_iter=100,
         multi_class='auto',
         n_jobs=None,
         penalty='l1',
         random_state=42,
         solver='liblinear',
         tol=0.0001,
         verbose=0,
         warm_start=False),
                                                       cv=3, method='si…
meta_classifier=CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.1,
            class_weight='balanced',
            dual=False,
            fit_intercept=True,
            intercept_scaling=1,
            l1_ratio=None,
            max_iter=100,
            multi_class='auto',
            n_jobs=None,
            penalty='l1',
            random_state=42,
            solver='liblinear',
            tol=0.0001,
            verbose=0,
            warm_start=False),
                                                          cv=3,
                                                          method='sigmoid'),
                   store_train_meta_features=False, use_clones=True,
                   use_features_in_secondary=False, use_probas=True, verbose=0)
```

22 5.6.1 Kaggle score without top features

[94]:

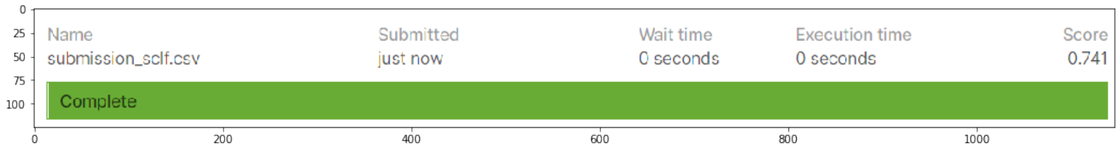
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
sclf_csv = sclf.predict_proba(ts_X)[:,1]
sclf_df = pd.DataFrame(np.column_stack((temp_id,sclf_csv)),␣
 ↪columns=['id','target'])
sclf_df['id'] = sclf_df['id'].astype('int32')
sclf_df.to_csv(data_dir+'/submission_sclf.csv', index=False)
```

[95]:

```
image = plt.imread(data_dir+'/submission_sclf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[95]:

```
<matplotlib.image.AxesImage at 0x266f2521408>
```



Name	Submitted	Wait time	Execution time	Score
submission_sclf.csv	just now	0 seconds	0 seconds	0.741

Complete

23 5.6.2 Using Top features

[97]:

```
# Fit the model
sclf.fit(tr_X[:,comb_top_feat], tr_y)
```

[97]:

```
StackingClassifier(average_probas=False,
classifiers=[CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.1,
         class_weight='balanced',
         dual=False,
         fit_intercept=True,
         intercept_scaling=1,
         l1_ratio=None,
         max_iter=100,
         multi_class='auto',
         n_jobs=None,
         penalty='l1',
         random_state=42,
         solver='liblinear',
         tol=0.0001,
         verbose=0,
         warm_start=False),
                                                        cv=3, method='si…
meta_classifier=CalibratedClassifierCV(base_estimator=LogisticRegression(C=0.1,
            class_weight='balanced',
            dual=False,
            fit_intercept=True,
            intercept_scaling=1,
            l1_ratio=None,
            max_iter=100,
            multi_class='auto',
            n_jobs=None,
            penalty='l1',
            random_state=42,
            solver='liblinear',
            tol=0.0001,
            verbose=0,
            warm_start=False),
                                                           cv=3,
                                                           method='sigmoid'),
                   store_train_meta_features=False, use_clones=True,
                   use_features_in_secondary=False, use_probas=True, verbose=0)
```

[98]:

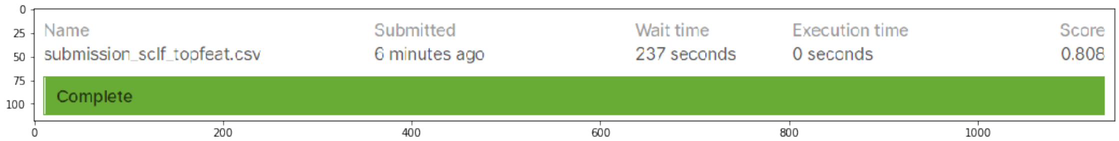
```python
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
sclf_csv = sclf.predict_proba(ts_X[:, comb_top_feat])[:,1]
sclf_df = pd.DataFrame(np.column_stack((temp_id,sclf_csv)),␣
 ↪columns=['id','target'])
sclf_df['id'] = sclf_df['id'].astype('int32')
sclf_df.to_csv(data_dir+'/submission_sclf_topfeat.csv', index=False)
```

[99]:

```python
image = plt.imread(data_dir+'/sclf_topfeat.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[99]:

```
<matplotlib.image.AxesImage at 0x266f23771c8>
```



Name	Submitted	Wait time	Execution time	Score
submission_sclf_topfeat.csv	6 minutes ago	237 seconds	0 seconds	0.808

Complete

24 5.7 Voting Classifier (Without Stack Classifier + no weights)

[100]:
```
# Import Voting Classifier
from mlxtend.classifier import EnsembleVoteClassifier
```

[102]:
```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2,clf3,clf4])
# Fit the train data
eclf.fit(tr_X,tr_y)
```

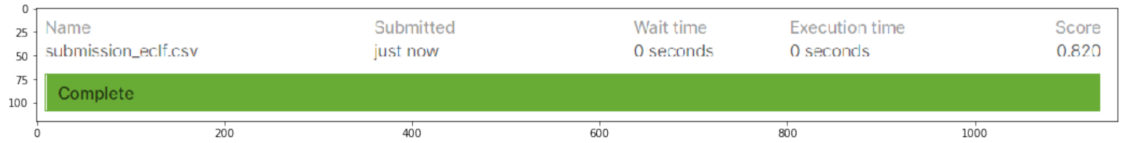
[102]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=0.1,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                             CalibratedClassi…
          max_depth=2,
          max_features='auto',
          max_leaf_nodes=None,
          max_samples=None,
          min_impurity_decrease=0.0,
          min_impurity_split=None,
          min_samples_leaf=1,
          min_samples_split=2,
          min_weight_fraction_leaf=0.0,
          n_estimators=300,
          n_jobs=None,
          oob_score=False,
          random_state=None,
          verbose=0,
          warm_start=False),
                                                    cv=3, method='sigmoid')],
                       refit=True, verbose=0, voting='hard', weights=None)
```

25 5.7.1 Kaggle Score without top features

[105]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X)[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf.csv', index=False)
```

[106]:
```
image = plt.imread(data_dir+'/submission_eclf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[106]:
```
<matplotlib.image.AxesImage at 0x266f23cd488>
```



Name	Submitted	Wait time	Execution time	Score
submission_eclf.csv	just now	0 seconds	0 seconds	0.820

Complete

26 5.7.2 Kaggle Score using top features

[107]:
```
# Fit the model
eclf.fit(tr_X[:,comb_top_feat], tr_y)
```

[107]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=0.1,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                             CalibratedClassi…
          max_depth=2,
          max_features='auto',
          max_leaf_nodes=None,
          max_samples=None,
          min_impurity_decrease=0.0,
          min_impurity_split=None,
          min_samples_leaf=1,
          min_samples_split=2,
          min_weight_fraction_leaf=0.0,
          n_estimators=300,
          n_jobs=None,
          oob_score=False,
          random_state=None,
          verbose=0,
          warm_start=False),
                                                    cv=3, method='sigmoid')],
                       refit=True, verbose=0, voting='hard', weights=None)
```

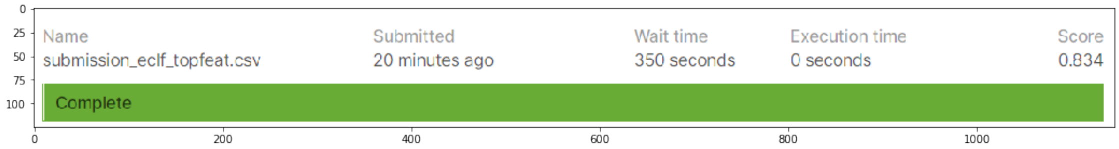
[108]:

```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X[:,comb_top_feat])[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_topfeat.csv', index=False)
```

[109]:

```
image = plt.imread(data_dir+'/eclf_topfeat.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

```
[109]: <matplotlib.image.AxesImage at 0x266f16b9ac8>
```



Name	Submitted	Wait time	Execution time	Score
submission_eclf_topfeat.csv	20 minutes ago	350 seconds	0 seconds	0.834

Complete

27 5.8 Voting Classifier (With Stack Classifier + no weights)

```
[110]: # Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
        ↪classifier/EnsembleVoteClassifier/)
       eclf = EnsembleVoteClassifier(clfs=[clf1, clf2,clf3,clf4,sclf])
       # Fit the train data
       eclf.fit(tr_X,tr_y)
```

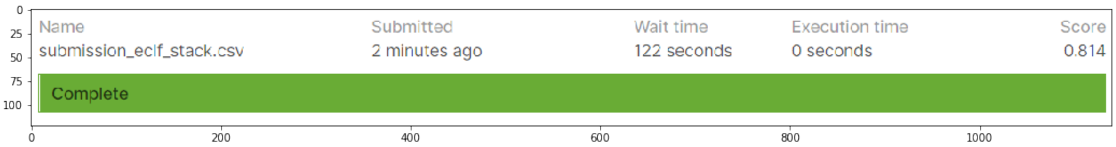
```
[110]: EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
       ssion(C=0.1,
             class_weight='balanced',
             dual=False,
             fit_intercept=True,
             intercept_scaling=1,
             l1_ratio=None,
             max_iter=100,
             multi_class='auto',
             n_jobs=None,
             penalty='l1',
             random_state=42,
             solver='liblinear',
             tol=0.0001,
             verbose=0,
             warm_start=False),
                                                           cv=3, method='sigmoid'),
                                    CalibratedClassi…
                                                fit_intercept=True,
                                                intercept_scaling=1,
                                                l1_ratio=None,
                                                max_iter=100,
                                                multi_class='auto',
                                                n_jobs=None,
                                                penalty='l1',
                                                random_state=42,
                                                solver='liblinear',
                                                tol=0.0001,
                                                verbose=0,
                                                warm_start=False),
              cv=3,
              method='sigmoid'),
                                                       store_train_meta_features=False,
                                                       use_clones=True,
                                                       use_features_in_secondary=False,
                                                       use_probas=True, verbose=0)],
                                    refit=True, verbose=0, voting='hard', weights=None)
```

28 5.8.1 Kaggle Score without top features

[111]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X)[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_stack.csv', index=False)
```

[112]:
```
image = plt.imread(data_dir+'/submission_eclf_stack.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[112]:
```
<matplotlib.image.AxesImage at 0x266f1fdfb88>
```



Name	Submitted	Wait time	Execution time	Score
submission_eclf_stack.csv	2 minutes ago	122 seconds	0 seconds	0.814

Complete

29 5.8.2 Kaggle Score using top features

[114]:
```
# Fit the model
eclf.fit(tr_X[:,comb_top_feat], tr_y)
```

[114]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=0.1,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                             CalibratedClassi…
                                         fit_intercept=True,
                                         intercept_scaling=1,
                                         l1_ratio=None,
                                         max_iter=100,
                                         multi_class='auto',
                                         n_jobs=None,
                                         penalty='l1',
                                         random_state=42,
                                         solver='liblinear',
                                         tol=0.0001,
                                         verbose=0,
                                         warm_start=False),
       cv=3,
       method='sigmoid'),
                                                store_train_meta_features=False,
                                                use_clones=True,
                                                use_features_in_secondary=False,
                                                use_probas=True, verbose=0)],
                       refit=True, verbose=0, voting='hard', weights=None)
```

[115]:

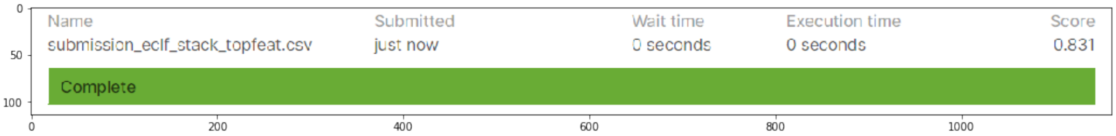
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X[:,comb_top_feat])[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_stack_topfeat.csv', index=False)
```

[116]:

```
image = plt.imread(data_dir+'/submission_eclf_stack_topfeat.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[116]:

```
<matplotlib.image.AxesImage at 0x266f1fb2608>
```



```
Name                          Submitted      Wait time    Execution time    Score
submission_eclf_stack_topfeat.csv  just now  0 seconds    0 seconds         0.831
Complete
```

30 5.9 Voting Classifier (without Stack Classifier + weights)

```
[141]: # Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
        ↪classifier/EnsembleVoteClassifier/)
       eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4], weights=[0.4,0.2,0.
        ↪2,0.2])
       # Fit the train data
       eclf.fit(tr_X,tr_y)
```

```
[141]: EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
       ssion(C=0.1,
             class_weight='balanced',
             dual=False,
             fit_intercept=True,
             intercept_scaling=1,
             l1_ratio=None,
             max_iter=100,
             multi_class='auto',
             n_jobs=None,
             penalty='l1',
             random_state=42,
             solver='liblinear',
             tol=0.0001,
             verbose=0,
             warm_start=False),
                                                           cv=3, method='sigmoid'),
                                    CalibratedClassi…
                 max_features='auto',
                 max_leaf_nodes=None,
                 max_samples=None,
                 min_impurity_decrease=0.0,
                 min_impurity_split=None,
                 min_samples_leaf=1,
                 min_samples_split=2,
                 min_weight_fraction_leaf=0.0,
                 n_estimators=300,
                 n_jobs=None,
                 oob_score=False,
                 random_state=None,
                 verbose=0,
                 warm_start=False),
                                                           cv=3, method='sigmoid')],
                              refit=True, verbose=0, voting='hard',
                              weights=[0.4, 0.2, 0.2, 0.2])
```

# 5.9.1 Kaggle Score without top features

[120]:

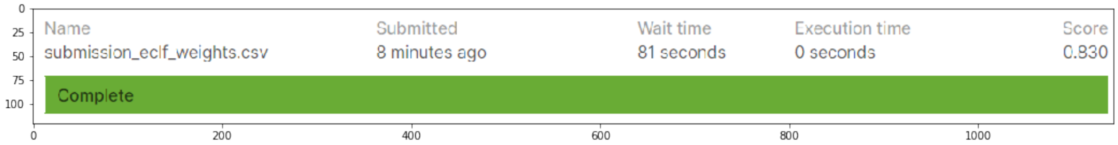
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X)[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_weights.csv', index=False)
```

[121]:

```
image = plt.imread(data_dir+'/submission_eclf_weights.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[121]:

```
<matplotlib.image.AxesImage at 0x266f2f91d08>
```



Name	Submitted	Wait time	Execution time	Score
submission_eclf_weights.csv	8 minutes ago	81 seconds	0 seconds	0.830

Complete

31 5.9.2 Kaggle Score using top features

[142]:

```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4], weights=[0.4,0.2,0.
 ↪2,0.2])
# Fit the train data
eclf.fit(tr_X[:,comb_top_feat],tr_y)
```

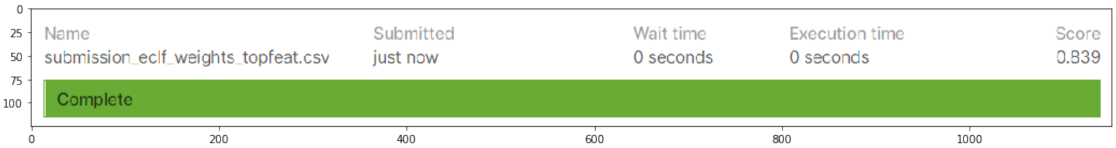
[142]:

```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=0.1,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                             CalibratedClassi…
          max_features='auto',
          max_leaf_nodes=None,
          max_samples=None,
          min_impurity_decrease=0.0,
          min_impurity_split=None,
          min_samples_leaf=1,
          min_samples_split=2,
          min_weight_fraction_leaf=0.0,
          n_estimators=300,
          n_jobs=None,
          oob_score=False,
          random_state=None,
          verbose=0,
          warm_start=False),
                                                    cv=3, method='sigmoid')],
                       refit=True, verbose=0, voting='hard',
                       weights=[0.4, 0.2, 0.2, 0.2])
```

[143]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X[:,comb_top_feat])[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_weights_topfeat.csv', index=False)
```

[145]:
```
image = plt.imread(data_dir+'/submission_eclf_weights_topfeat.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[145]:
```
<matplotlib.image.AxesImage at 0x266f4cacf08>
```



Name | Submitted | Wait time | Execution time | Score
submission_eclf_weights_topfeat.csv | just now | 0 seconds | 0 seconds | 0.839

Complete

32 5.10 Voting Classifier (with Stack Classifier + weights)

```
[135]: # Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
        ↪classifier/EnsembleVoteClassifier/)
       eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4,sclf], weights=[0.4,0.
        ↪1,0.1,0.2,0.2])
       # Fit the train data
       eclf.fit(tr_X,tr_y)
```

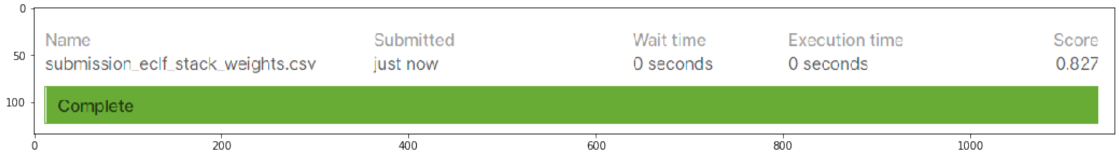
```
[135]: EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
       ssion(C=0.1,
             class_weight='balanced',
             dual=False,
             fit_intercept=True,
             intercept_scaling=1,
             l1_ratio=None,
             max_iter=100,
             multi_class='auto',
             n_jobs=None,
             penalty='l1',
             random_state=42,
             solver='liblinear',
             tol=0.0001,
             verbose=0,
             warm_start=False),
                                                           cv=3, method='sigmoid'),
                                    CalibratedClassi…
                                                intercept_scaling=1,
                                                l1_ratio=None,
                                                max_iter=100,
                                                multi_class='auto',
                                                n_jobs=None,
                                                penalty='l1',
                                                random_state=42,
                                                solver='liblinear',
                                                tol=0.0001,
                                                verbose=0,
                                                warm_start=False),
              cv=3,
              method='sigmoid'),
                                                       store_train_meta_features=False,
                                                       use_clones=True,
                                                       use_features_in_secondary=False,
                                                       use_probas=True, verbose=0)],
                              refit=True, verbose=0, voting='hard',
                              weights=[0.4, 0.1, 0.1, 0.2, 0.2])
```

33 5.10.1 Kaggle Score without top features

[136]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X)[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_stack_weights.csv', index=False)
```

[137]:
```
image = plt.imread(data_dir+'/submission_eclf_stack_weights.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[137]:
```
<matplotlib.image.AxesImage at 0x266f272d048>
```



```
Name                             Submitted   Wait time   Execution time   Score
submission_eclf_stack_weights.csv just now   0 seconds   0 seconds        0.827
Complete
```

34 5.10.2 Kaggle Score using top features

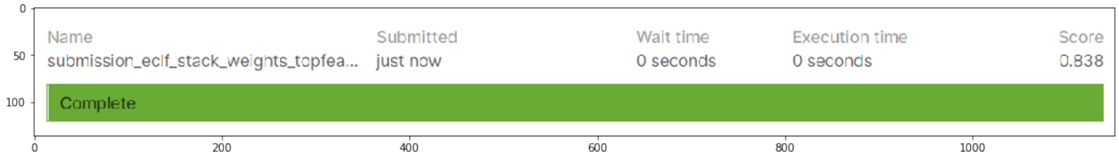
[138]:
```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4,sclf], weights=[0.4,0.
 ↪1,0.1,0.2,0.2])
# Fit the train data
eclf.fit(tr_X[:,comb_top_feat],tr_y)
```

[138]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=0.1,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                             CalibratedClassi…
                                         intercept_scaling=1,
                                         l1_ratio=None,
                                         max_iter=100,
                                         multi_class='auto',
                                         n_jobs=None,
                                         penalty='l1',
                                         random_state=42,
                                         solver='liblinear',
                                         tol=0.0001,
                                         verbose=0,
                                         warm_start=False),
       cv=3,
       method='sigmoid'),
                                                store_train_meta_features=False,
                                                use_clones=True,
                                                use_features_in_secondary=False,
                                                use_probas=True, verbose=0)],
                       refit=True, verbose=0, voting='hard',
                       weights=[0.4, 0.1, 0.1, 0.2, 0.2])
```

[139]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
eclf_csv = eclf.predict_proba(ts_X[:,comb_top_feat])[:,1]
eclf_df = pd.DataFrame(np.column_stack((temp_id,eclf_csv)),␣
 ↪columns=['id','target'])
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_stack_weights_topfeat.csv',␣
 ↪index=False)
```

[140]:
```
image = plt.imread(data_dir+'/submission_eclf_stack_weights_topfeat.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[140]:
```
<matplotlib.image.AxesImage at 0x266f3060b48>
```



Name | Submitted | Wait time | Execution time | Score
submission_eclf_stack_weights_topfea... | just now | 0 seconds | 0 seconds | 0.838

Complete

35 6. Summary of All Models

```
[151]: from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ['Model','Features','Hyperparameter','Test Score']
x.add_row(['knn','AF',r"{'algorithm': 'kd_tree', 'n_neighbors': 35}",0.693])
x.add_row(['Logistic Regression','AF',r"{'C': 0.1, 'penalty': 'l1', 'solver':␣
 ↪'liblinear'}",0.828])
x.add_row(['Logistic Regression','top10',r"{'C': 0.1, 'penalty': 'l1', 'solver':
 ↪ 'liblinear'}",0.821])
x.add_row(['Logistic Regression','top20',r"{'C': 0.1, 'penalty': 'l1', 'solver':
 ↪ 'liblinear'}",0.822])
x.add_row(['SVC','AF',r"{'C': 10, 'kernel': 'rbf'}",0.720])
x.add_row(['SVC','top10',r"{'C': 10, 'kernel': 'rbf'}",0.689])
x.add_row(['RandomForest','AF',r"{'max_depth': 2, 'n_estimators': 300}",0.745])
x.add_row(['RandomForest','top10',r"{'max_depth': 2, 'n_estimators': 300}",0.
 ↪784])
x.add_row(['XGBoost','AF',r"{'max_depth': 2, 'n_estimators': 300}",0.765])
x.add_row(['XGBoost','top10',r"{'max_depth': 2, 'n_estimators': 300}",0.646])
x.add_row(['Stacking Classifier','AF','-',0.741])
x.add_row(['Stacking Classifier','topfeatures','-',0.808])
x.add_row(['Voting Classifier(No stacking + no weights)','AF',"-",0.820])
x.add_row(['Voting Classifier(No stacking + no weights)','topfeatures',"-",0.
 ↪834])
x.add_row(['Voting Classifier(stacking + no weights)','AF',"-",0.814])
x.add_row(['Voting Classifier(stacking + no weights)','topfeatures',"-",0.831])
x.add_row(['Voting Classifier(no stacking + weights)','AF',"-",0.830])
x.add_row(['Voting Classifier(no stacking + weights)','topfeatures',"-",0.839])
x.add_row(['Voting Classifier(stacking + weights)','AF',"-",0.827])
x.add_row(['Voting Classifier(stacking + weights)','topfeatures',"-",0.838])
print(x)
```

```
+---------------------------------------------+-------------+-------------------
---------------------------------+------------+
|                    Model                    |   Features  |
Hyperparameter                               | Test Score |
+---------------------------------------------+-------------+-------------------
---------------------------------+------------+
|                     knn                     |      AF     |    {'algorithm':
'kd_tree', 'n_neighbors': 35}      |   0.693    |
|             Logistic Regression             |      AF     | {'C': 0.1,
'penalty': 'l1', 'solver': 'liblinear'} |   0.828    |
|             Logistic Regression             |    top10    | {'C': 0.1,
'penalty': 'l1', 'solver': 'liblinear'} |   0.821    |
|             Logistic Regression             |    top20    | {'C': 0.1,
'penalty': 'l1', 'solver': 'liblinear'} |   0.822    |
|                     SVC                     |      AF     |              {'C':
```

```
10, 'kernel': 'rbf'}             |    0.72    |
|                     SVC                     |    top10    |             {'C':
10, 'kernel': 'rbf'}             |   0.689    |
|                 RandomForest                |      AF     |
{'max_depth': 2, 'n_estimators': 300}        |   0.745    |
|                 RandomForest                |    top10    |
{'max_depth': 2, 'n_estimators': 300}        |   0.784    |
|                   XGBoost                   |      AF     |
{'max_depth': 2, 'n_estimators': 300}        |   0.765    |
|                   XGBoost                   |    top10    |
{'max_depth': 2, 'n_estimators': 300}        |   0.646    |
|             Stacking Classifier             |      AF     |
-                          |   0.741    |
|             Stacking Classifier             | topfeatures |
-                          |   0.808    |
| Voting Classifier(No stacking + no weights) |      AF     |
-                          |    0.82    |
| Voting Classifier(No stacking + no weights) | topfeatures |
-                          |   0.834    |
|   Voting Classifier(stacking + no weights)  |      AF     |
-                          |   0.814    |
|   Voting Classifier(stacking + no weights)  | topfeatures |
-                          |   0.831    |
|   Voting Classifier(no stacking + weights)  |      AF     |
-                          |    0.83    |
|   Voting Classifier(no stacking + weights)  | topfeatures |
-                          |   0.839    |
|    Voting Classifier(stacking + weights)    |      AF     |
-                          |   0.827    |
|    Voting Classifier(stacking + weights)    | topfeatures |
-                          |   0.838    |
+---------------------------------------------+-------------+-------------------
---------------------------------+------------+
```

Notation: 1. AF: All features 2. top10: Find top 10 features using forward feature selections of that model 3. top20: Find top 20 features using forward feature selections of that model 4. topfeatures: combining all the top features generated using forward feature selection of that model.

[ ]: