4_4_Models

April 20, 2020

1 FE + Dimension Reduction + Standardization + ML Classification Model

1. No oversampling techniques applied
2. feature engineering applied

2 1. Import Necessary Libraries

[1]:
```
# 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
# Dimension reduction
```

```
from sklearn.decomposition import PCA
```

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. Apply Feature Engg

[4]:
```
# We already saw in 2_FE.ipynb file that we created a feat_enng function. We␣
↪just put it here

def feature_engg(df, if_test = False):
    '''
    Perform Feature Engg in Basic Stats, Trigometrics, Hyperbolic and␣
↪Exponential Function

    Parameters:
    df: Pass DataFrame (all features much be in numric values)
    if_test: If the DataFrame is test data or train data. Ig it is test data,␣
↪put if_test=True

    Return:
    DataFrame with feature engineering appended
    '''

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

    # Mean and Std FE
    df['mean'] = np.mean(temp, axis=1)
    df['std'] = np.std(temp, axis=1)

    # Trigometric FE
    sin_temp = np.sin(temp)
    cos_temp = np.cos(temp)
    tan_temp = np.tan(temp)
    df['mean_sin'] = np.mean(sin_temp, axis=1)
    df['mean_cos'] = np.mean(cos_temp, axis=1)
    df['mean_tan'] = np.mean(tan_temp, axis=1)

    # Hyperbolic FE
    sinh_temp = np.sinh(temp)
    cosh_temp = np.cosh(temp)
    tanh_temp = np.tanh(temp)
    df['mean_sinh'] = np.mean(sin_temp, axis=1)
    df['mean_cosh'] = np.mean(cos_temp, axis=1)
    df['mean_tanh'] = np.mean(tan_temp, axis=1)

    # Exponents FE
    exp_temp = np.exp(temp)
    expm1_temp = np.expm1(temp)
```

```
    exp2_temp = np.exp2(temp)
    df['mean_exp'] = np.mean(exp_temp, axis=1)
    df['mean_expm1'] = np.mean(expm1_temp, axis=1)
    df['mean_exp2'] = np.mean(exp2_temp, axis=1)

    # Polynomial FE
    # X**2
    df['mean_x2'] = np.mean(np.power(temp,2), axis=1)
    # X**3
    df['mean_x3'] = np.mean(np.power(temp,3), axis=1)
    # X**4
    df['mean_x4'] = np.mean(np.power(temp,4), axis=1)

    return df
```

[5]:
```
df_train = feature_engg(df_train)
df_train.head(5)
```

[5]:

	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	…	

	mean_tan	mean_sinh	mean_cosh	mean_tanh	mean_exp	mean_expm1	mean_exp2	\
0	-0.315591	-0.010536	0.537968	-0.315591	1.760647	0.760647	1.315869	
1	0.607457	0.075490	0.611600	0.607457	1.712292	0.712292	1.324817	
2	0.104777	-0.005509	0.599358	0.104777	1.749107	0.749107	1.313960	
3	0.891722	0.046067	0.645721	0.891722	1.752101	0.752101	1.326229	
4	0.274261	0.059548	0.643508	0.274261	1.861741	0.861741	1.377569	

	mean_x2	mean_x3	mean_x4
0	1.182425	0.015243	3.584848
1	0.976056	0.047272	2.766570
2	1.023024	0.266454	3.092631
3	0.887980	0.371308	2.553467
4	0.901115	0.613952	2.671541

[5 rows x 316 columns]

[6]:
```
df_test = feature_engg(df_test, True)
df_test.head(5)
```

[6]:

	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  …

   mean_tan  mean_sinh  mean_cosh  mean_tanh  mean_exp  mean_expm1  mean_exp2  \
0  0.565830   0.094378   0.609398   0.565830  1.904397    0.904397   1.404195
1 -1.641918  -0.018425   0.570495  -1.641918  1.642217    0.642217   1.265487
2 -0.516155  -0.012641   0.611053  -0.516155  1.517775    0.517775   1.214393
3 -0.816079   0.002689   0.610619  -0.816079  1.566765    0.566765   1.243412
4 -1.547172   0.067329   0.611907  -1.547172  1.849024    0.849024   1.374870

    mean_x2   mean_x3   mean_x4
0  0.985912  0.477020  2.913247
1  1.094274 -0.128315  3.281111
2  0.994294 -0.330590  3.062801
3  0.956136 -0.076546  2.382968
4  0.988710  0.371320  3.079160

[5 rows x 315 columns]
```

5 4. Take train and test values from DataFrame

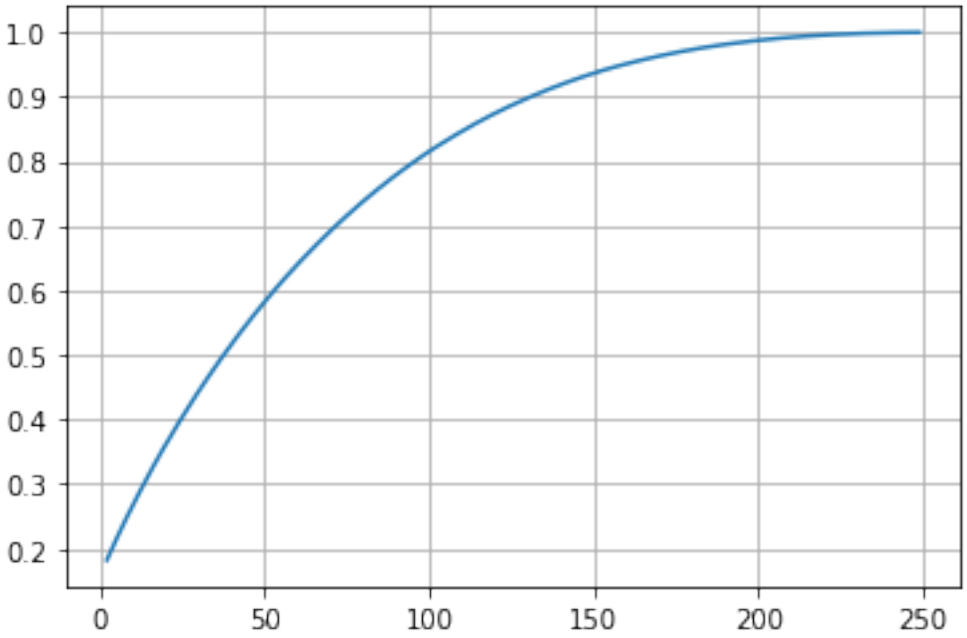
[7]:
```
# Take separate for features value
tr_X = df_train.drop(['id','target'], axis=1)
# Take separate for class value
tr_y = df_train['target'].values
# Take test feature value
ts_X = df_test.drop(['id'], axis=1)
```

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

6 5. PCA - Principal Components Analysis

[8]:
```
exp_rat = []
for i in range(2,min(tr_X.shape[0],tr_X.shape[1])):
    pca = PCA(n_components=i)
    pca.fit(tr_X,tr_y)
    exp_rat.append(np.sum(pca.explained_variance_ratio_))
```

[9]:
```
plt.plot(np.arange(2,min(tr_X.shape[0],tr_X.shape[1])),exp_rat)
plt.grid()
plt.show()
```



Let take n_components = 175 which retain 90-100% data

[10]:
```
# Fit and transform on train data and transform on cv and test data
pca = PCA(n_components=175)
tr_X = pca.fit_transform(tr_X,tr_y)
ts_X = pca.transform(ts_X)
```

7 6. Standardization

[11]:
```
stand_vec = StandardScaler()
tr_X = stand_vec.fit_transform(tr_X)
pd.DataFrame(tr_X).head(5)
```

[11]:

	0	1	2	3	4	5	6	\
0	-0.166673	0.716303	1.061006	0.283135	0.765347	1.253518	-1.967637	
1	0.031105	0.688296	-0.679376	-1.075485	1.484971	-1.484536	-0.954770	
2	-0.037655	0.759798	0.240590	-1.494130	0.680533	-1.507534	-1.475887	
3	0.089884	-0.794913	0.652919	-0.124737	-1.415833	0.182715	1.361817	
4	-0.048058	0.464460	0.932549	0.062235	1.730730	-0.003666	-1.875208	

	7	8	9	…	165	166	167	168	\
0	1.739057	-1.551768	2.025589	…	-0.635075	-2.100014	0.666909	0.043840	
1	-1.640496	0.533955	-1.032526	…	-0.354963	-0.529771	-1.564648	-1.346141	
2	1.563435	-0.133188	1.023150	…	-0.289336	0.295746	1.614245	0.119328	
3	-0.364036	1.076568	-0.197581	…	0.101799	0.308904	-0.120174	-1.617262	

```
4  0.016811  0.794978  1.687170  … -0.657138  0.750889 -1.080992 -2.022388

        169       170       171       172       173       174
0  1.901291 -0.418972  0.861888  1.142312 -1.561428 -1.269437
1 -0.058692 -2.748382 -0.072904 -0.391150  1.329331 -0.870928
2 -0.662576 -0.278453 -0.349016 -0.037971  0.322065  0.447964
3 -0.443145 -0.092754  0.702214  1.046393  1.584612 -0.887656
4 -0.096655  0.270118  0.472265 -0.426247 -0.855543 -0.276437

[5 rows x 175 columns]
```

[12]:
```
ts_X = stand_vec.transform(ts_X)
pd.DataFrame(ts_X).head(5)
```

[12]:
```
        0         1         2         3         4         5         6    \
0  0.019110  0.276096 -0.061850 -0.436308  0.152002 -0.063529  0.282660
1 -0.361623 -0.459394  0.611061 -0.043825 -0.601051 -0.794353 -0.333588
2 -0.174794  0.385567 -0.876048  0.592629  0.129279 -0.328676 -0.401293
3 -0.220446  0.983462 -0.596268  0.007514 -0.104571  0.285299 -0.541471
4 -0.327378 -0.630620 -1.088273 -1.305239 -0.065958  0.116384 -0.027180

        7         8         9    …       165       166       167       168  \
0 -0.963953  0.269206  0.077995  …  1.782189  0.742038 -0.075163 -0.608261
1  0.045550  0.197833  0.466416  …  1.775805 -1.325259 -1.648324 -0.715455
2  1.146829 -0.305653 -0.500269  …  0.574671  0.101536 -1.079702 -0.955873
3  0.045604 -0.163915  0.472175  …  0.925415 -0.736115  1.138512 -0.137684
4  0.027114 -0.500452 -0.846949  …  0.412247  2.257494  0.348214 -0.385650

        169       170       171       172       173       174
0  0.615582  0.414376  1.553202  2.300787 -0.908851 -1.048736
1  1.736466  0.115542 -0.388295  1.037530  0.711408  1.480681
2 -0.431163  0.158387  1.789018 -0.489373 -1.481817  0.170374
3  0.302283 -2.858745 -4.126849 -1.287239  0.360416  4.214549
4  1.625903  1.358772  1.950546  0.156232  1.116639  1.933765

[5 rows x 175 columns]
```

8 7. Apply ML Models (with hyperparameter)

[13]:
```
def hyperparameter_model(models, params):
    '''
    Hyperparameter tuning with StratifiedKFold follow by GridSearchCV follow by␣
 ↪CalibratedClassifier

    Parameters:
    models: Instance of the model
```

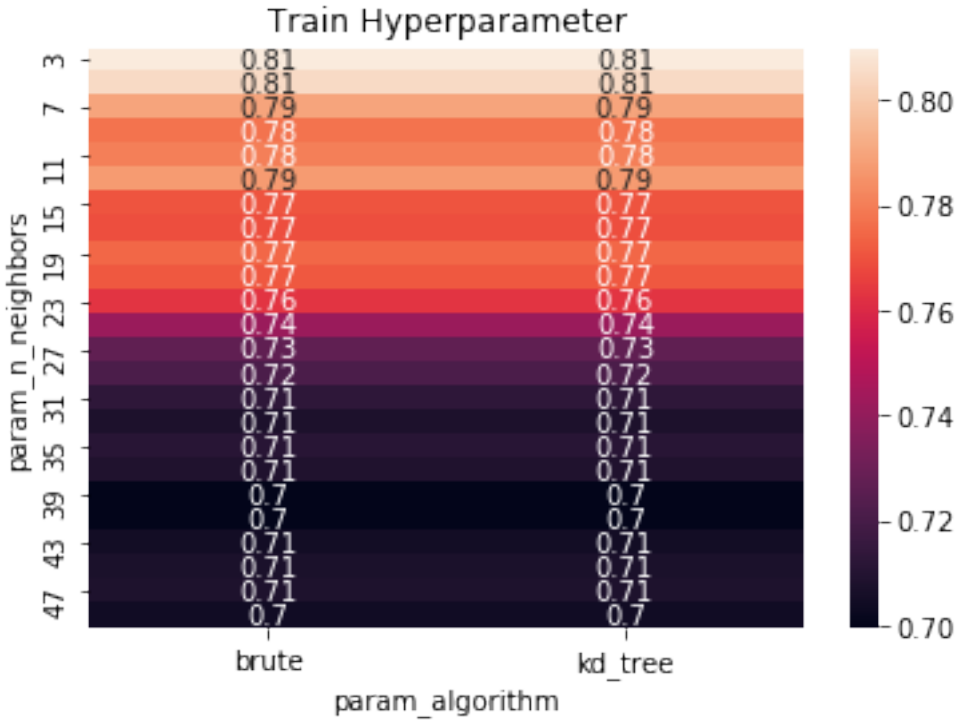
```
    params: list of parameters with value fr tuning (dict)

    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
```

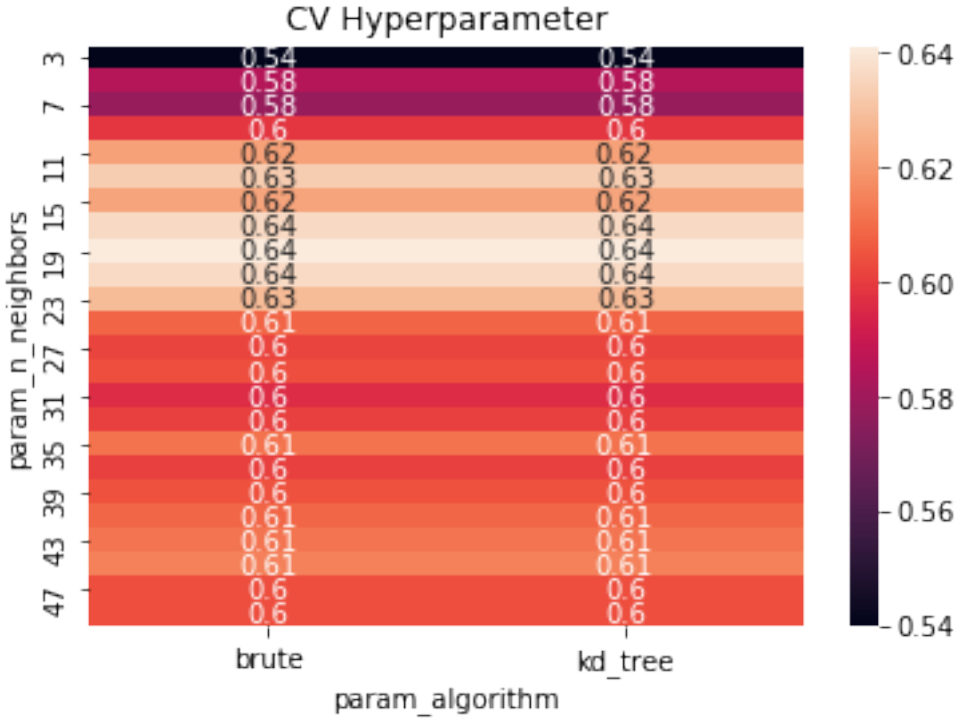
9 7.1 kNN

[14]:
```
# Import KNN
from sklearn.neighbors import KNeighborsClassifier
```

[15]:
```
# 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)
```

[16]:
```
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()
```


Train Hyperparameter
param_n_neighbors
3
7
11
15
19
23
27
31
35
39
43
47
0.81 0.81
0.81 0.81
0.79 0.79
0.78 0.78
0.78 0.78
0.79 0.79
0.77 0.77
0.77 0.77
0.77 0.77
0.77 0.77
0.76 0.76
0.74 0.74
0.73 0.73
0.72 0.72
0.71 0.71
0.71 0.71
0.71 0.71
0.71 0.71
0.7 0.7
0.7 0.7
0.71 0.71
0.71 0.71
0.71 0.71
0.7 0.7
brute
kd_tree
param_algorithm
0.80
0.78
0.76
0.74
0.72
0.70


CV Hyperparameter
param_n_neighbors
3
7
11
15
19
23
27
31
35
39
43
47
0.54 0.54
0.58 0.58
0.58 0.58
0.6 0.6
0.62 0.62
0.63 0.63
0.62 0.62
0.64 0.64
0.64 0.64
0.64 0.64
0.63 0.63
0.61 0.61
0.6 0.6
0.6 0.6
0.6 0.6
0.6 0.6
0.61 0.61
0.6 0.6
0.6 0.6
0.61 0.61
0.61 0.61
0.61 0.61
0.6 0.6
0.6 0.6
brute
kd_tree
param_algorithm
0.64
0.62
0.60
0.58
0.56
0.54

[17]:
```
print(knn_clf.best_params_)
```

```
{'algorithm': 'kd_tree', 'n_neighbors': 19}
```

[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,
```

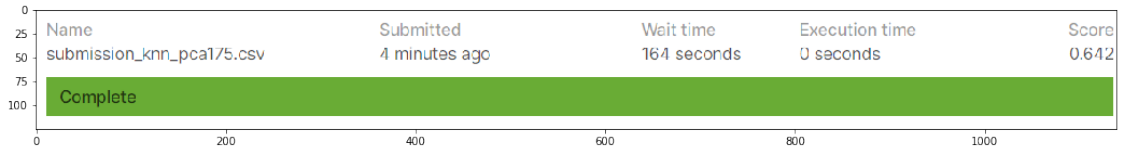
```
49]},
                                                    pre_dispatch='2*n_jobs',
                                                    refit=True,
                                                    return_train_score=True,
                                                    scoring='roc_auc',
                                                    verbose=0),
                         cv=3, method='sigmoid')
```

10 7.1.1 Kaggle Score

[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_pca175.csv', index=False)
```

[20]:
```
image = plt.imread(data_dir+'/submission_knn_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[20]:
```
<matplotlib.image.AxesImage at 0x21853affc48>
```



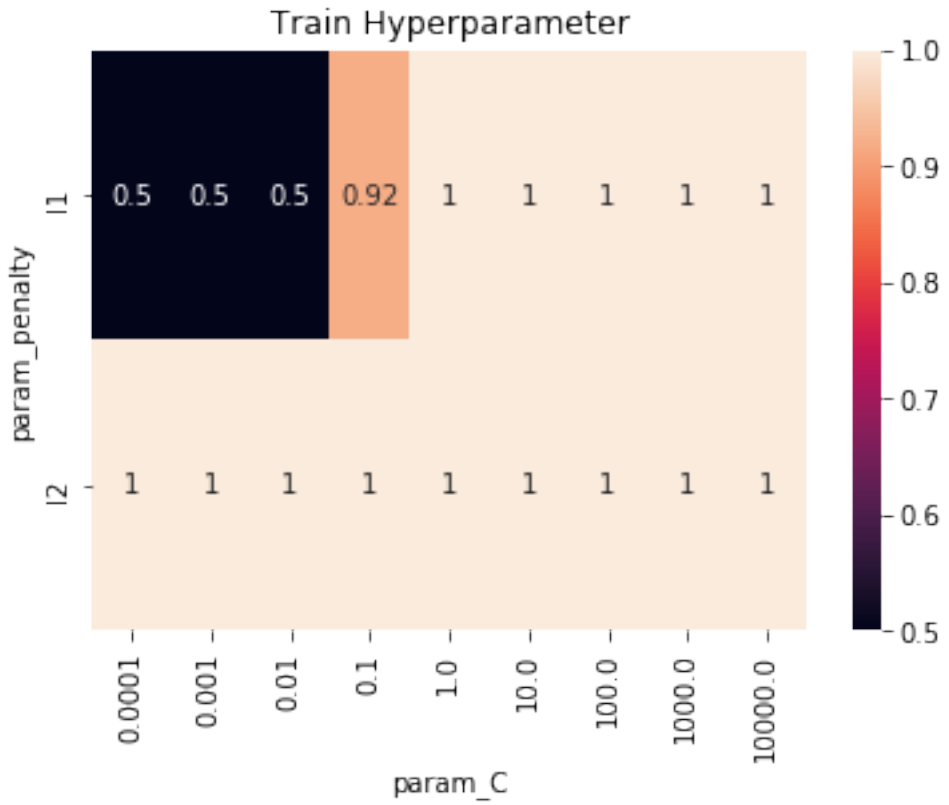
10.1 7.2 Logistic Regression

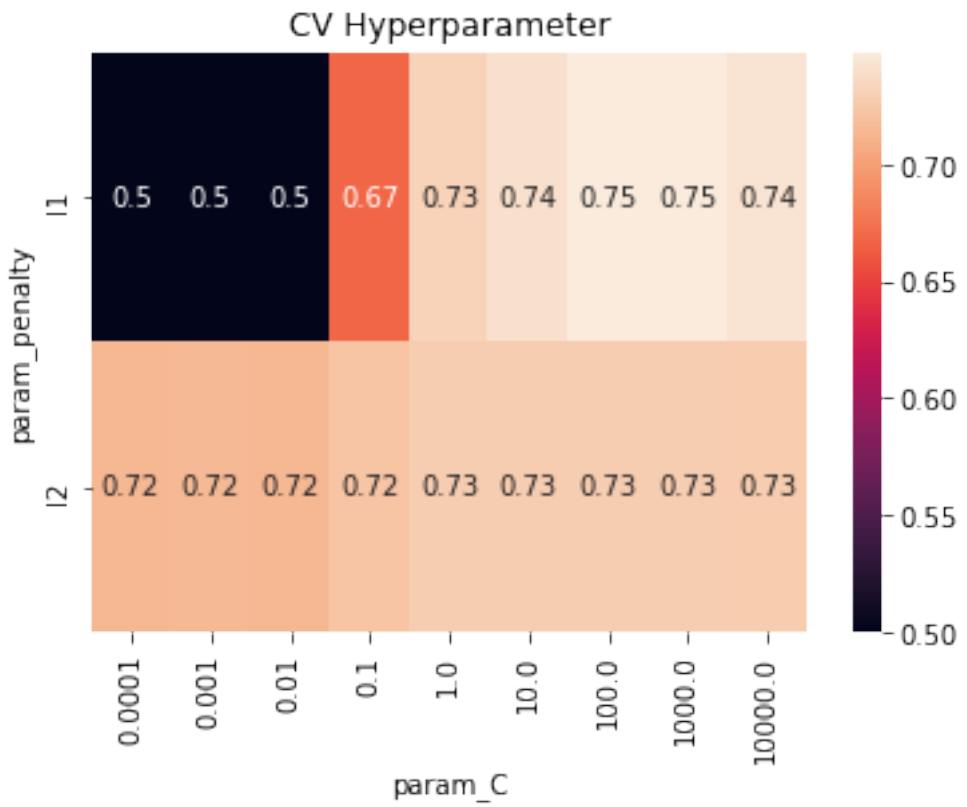
[21]:
```
# Import Logistic Regression
from sklearn.linear_model import LogisticRegression
```

[22]:
```
# 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)
```

[23]:
```
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()
```





[24]:
```
print(log_clf.best_params_)
```

```
{'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}
```

[25]:
```
clf = CalibratedClassifierCV(log_clf, cv=3)
clf.fit(tr_X,tr_y)
```

[25]:
```
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,
```

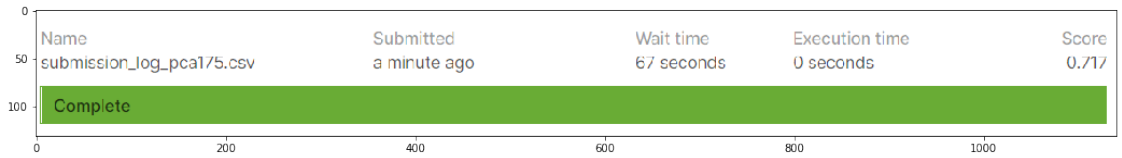
```
solver='lbfgs',
tol=0.0001,
verbose=0,
warm_start=False),
                                                    iid='deprecated',
                                                    n_jobs=None,
                                                    param_grid={'C': [0.0001,
                                                                      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')
```

11 7.2.1 Kaggle Score

[26]:
```
# 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_pca175.csv', index=False)
```

[27]:
```
image = plt.imread(data_dir+'/submission_log_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[27]:
```
<matplotlib.image.AxesImage at 0x21853b52d48>
```

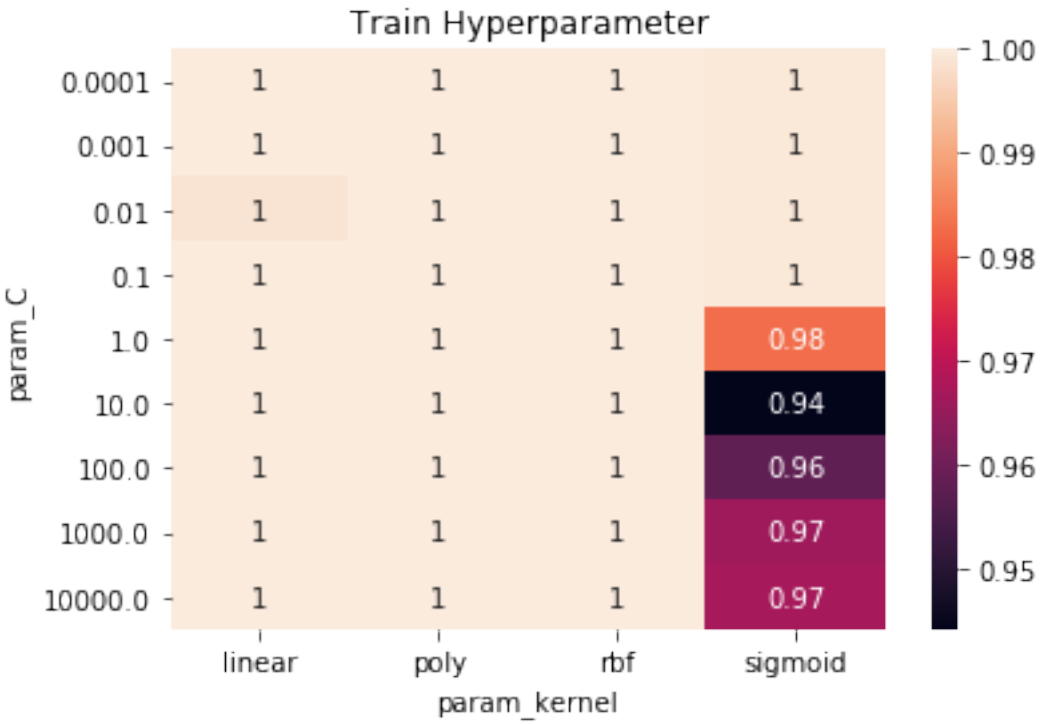


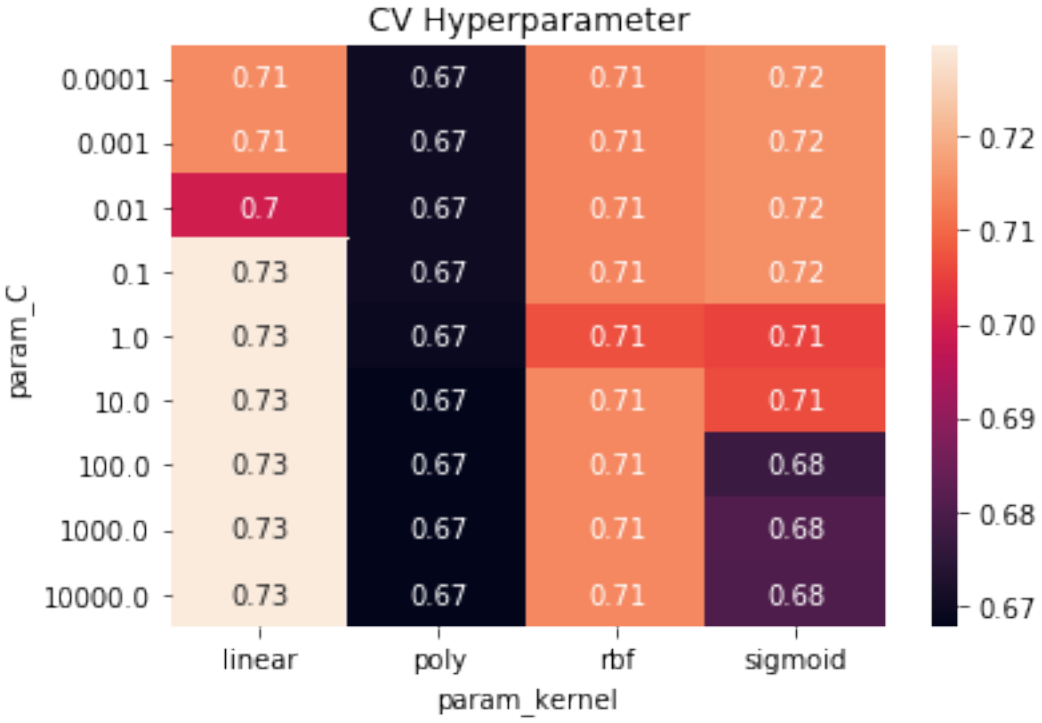
12 7.3 SVC

[28]:
```
# Import SVC
from sklearn.svm import SVC
```

[29]:
```
# SVC (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.sum.
 ↪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)
```

[30]:
```
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
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 0.98
1 1 1 0.94
1 1 1 0.96
1 1 1 0.97
1 1 1 0.97
linear
poly
rbf
sigmoid
param_kernel
1.00
0.99
0.98
0.97
0.96
0.95


CV Hyperparameter
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
0.71 0.67 0.71 0.72
0.71 0.67 0.71 0.72
0.7 0.67 0.71 0.72
0.73 0.67 0.71 0.72
0.73 0.67 0.71 0.71
0.73 0.67 0.71 0.71
0.73 0.67 0.71 0.68
0.73 0.67 0.71 0.68
0.73 0.67 0.71 0.68
linear
poly
rbf
sigmoid
param_kernel
0.72
0.71
0.70
0.69
0.68
0.67

[31]:
```
print(svc_clf.best_params_)
```

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

[32]:
```
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)
```

[32]:
```
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),
```

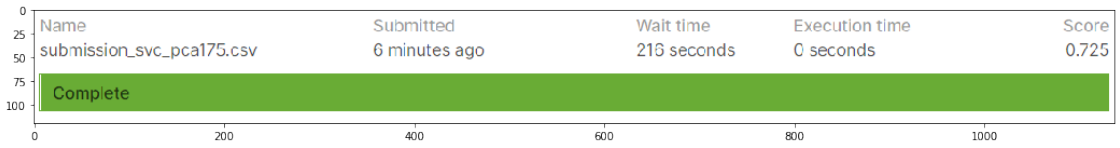
```
cv=3, method='sigmoid')
```

13 7.3.1 Kaggle Score

[33]:
```
# 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_pca175.csv', index=False)
```

[34]:
```
image = plt.imread(data_dir+'/submission_svc_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[34]:
```
<matplotlib.image.AxesImage at 0x218539c5dc8>
```



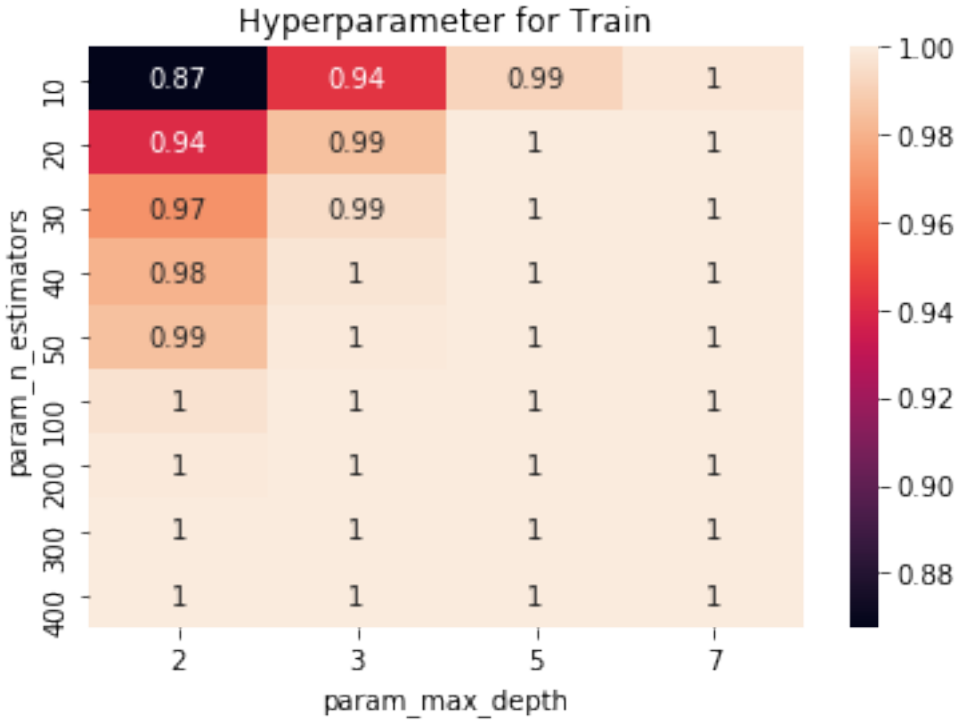
14 7.4 RandomForest

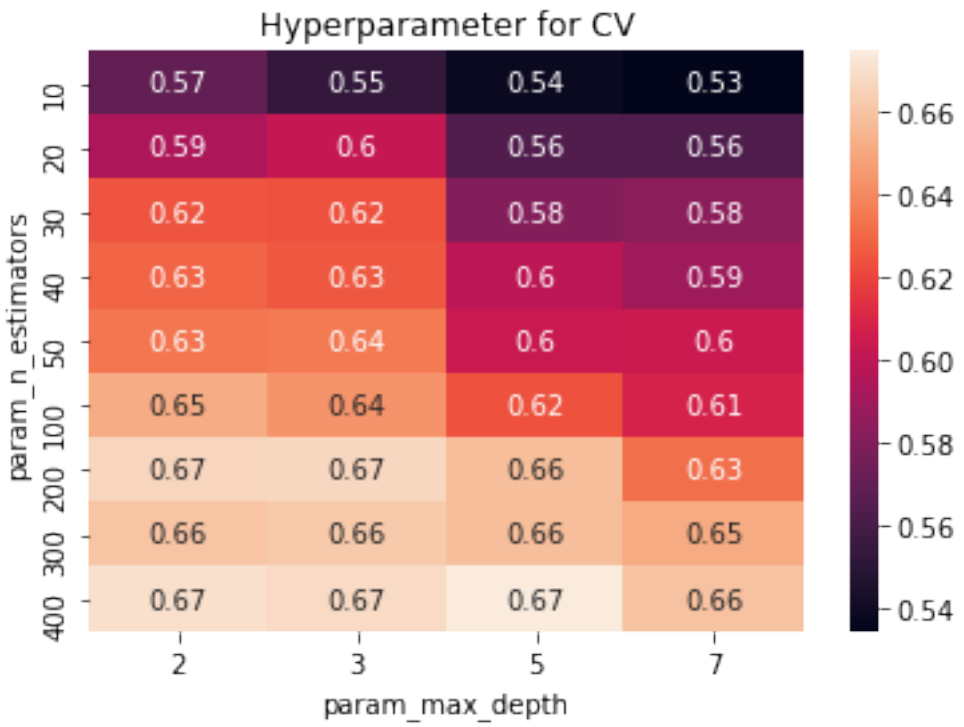
[35]:
```
# Impoer Random Forest
from sklearn.ensemble import RandomForestClassifier
```

[36]:
```
# 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)
```

[37]:
```
# Ref: https://stackoverflow.com/questions/48791709/
 ↪how-to-plot-a-heat-map-on-pivot-table-after-grid-search

# Plotting of hyperpameter 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()
```





[38]:
```
print(rf_clf.best_params_)
```

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

[39]:
```
# Calibrate the model
clf = CalibratedClassifierCV(rf_clf, cv=3)
clf.fit(tr_X, tr_y)
```

[39]:
```
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,
```

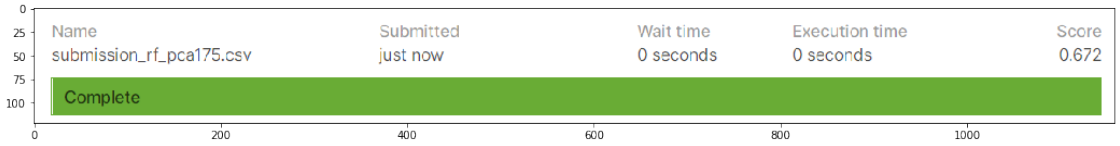
```
    n_estimators=100,
    n_jobs=None,
    oob_score=False,
    random_state=42,
    verbose=0,
    warm_start=False),
                                                    iid='deprecated',
                                                    n_jobs=None,
                                                    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')
```

15 7.4.1 Kaggle Score

[40]:
```
# 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_pca175.csv', index=False)
```

[41]:
```
image = plt.imread(data_dir+'/submission_rf_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[41]:
```
<matplotlib.image.AxesImage at 0x21851c3a748>
```

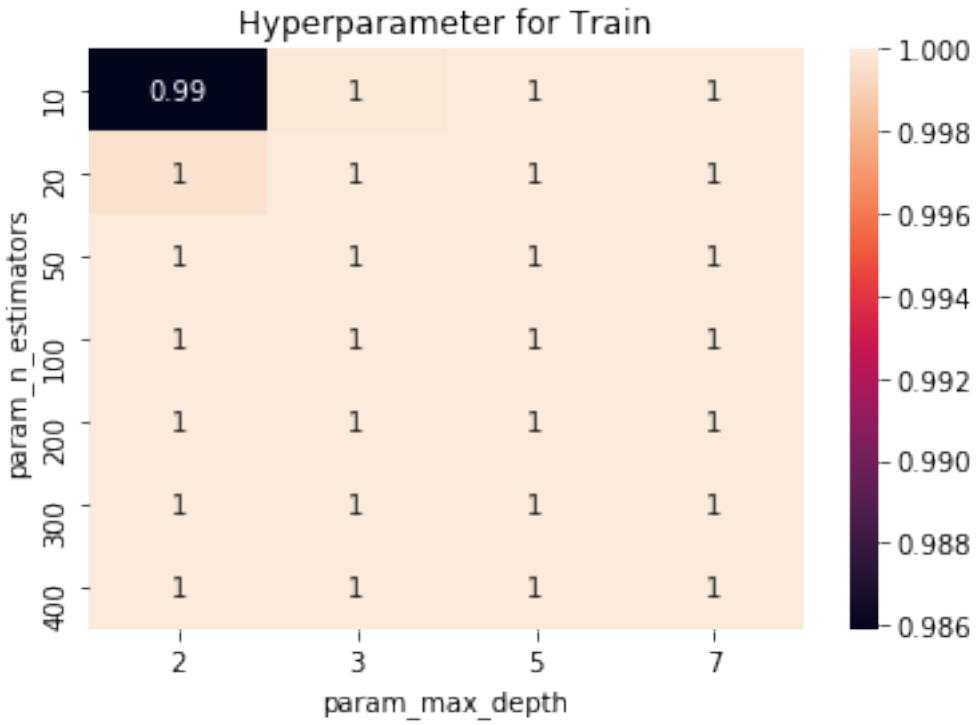


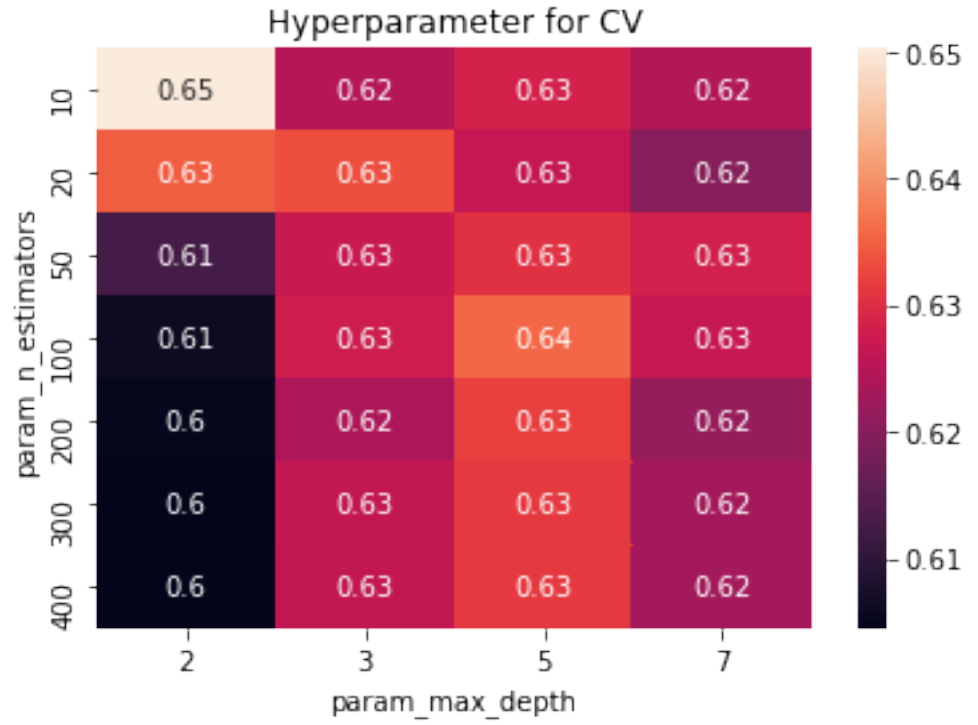
16 7.5 XGBoost

[42]:
```
# Import Xgboost
from xgboost import XGBClassifier
```

[43]:
```
# 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)
```

[44]:
```
# Ref: https://stackoverflow.com/questions/48791709/
 ↪how-to-plot-a-heat-map-on-pivot-table-after-grid-search

# Plotting of hyperpameter 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
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
1
1
1
param_max_depth
2
3
5
7
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986


Hyperparameter for CV
param_n_estimators
10
20
50
100
200
300
400
0.65
0.62
0.63
0.62
0.63
0.63
0.63
0.62
0.61
0.63
0.63
0.63
0.61
0.63
0.64
0.63
0.6
0.62
0.63
0.62
0.6
0.63
0.63
0.62
0.6
0.63
0.63
0.62
param_max_depth
2
3
5
7
0.65
0.64
0.63
0.62
0.61

[45]:
```
print(xgb_clf.best_params_)
```

```
{'max_depth': 2, 'n_estimators': 10}
```

[46]:
```
# Calibrate the model
clf = CalibratedClassifierCV(xgb_clf, cv=3)
clf.fit(tr_X, tr_y)
```

[46]:
```
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,
```

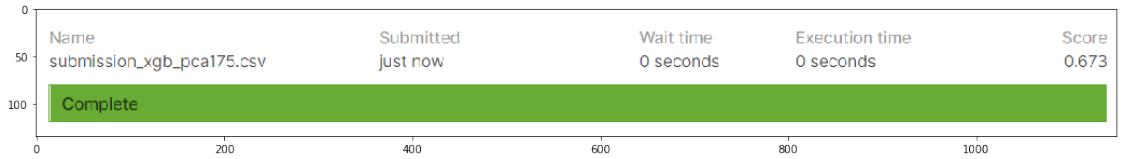
```
                                                    scoring='roc_auc',
                                                    verbose=0),
                          cv=3, method='sigmoid')
```

17 7.5.1 Kaggle Score

[47]:
```
# 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_pca175.csv', index=False)
```

[48]:
```
image = plt.imread(data_dir+'/submission_xgb_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[48]:
```
<matplotlib.image.AxesImage at 0x218604cf908>
```



18 7.6 Stacking Classifier

[49]:
```
# Import Stacking Classifier
from mlxtend.classifier import StackingClassifier
```

[50]:
```
# StackClassifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/StackingClassifier/#methods)

# Classifier 1: Logistic Regression with best params
clf1 = LogisticRegression(C = 100, 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=0.1, kernel='linear', 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=10, 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=5, n_estimators=400)
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)
```

[50]:
```
StackingClassifier(average_probas=False,
classifiers=[CalibratedClassifierCV(base_estimator=LogisticRegression(C=100,
          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=100,
             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',
```

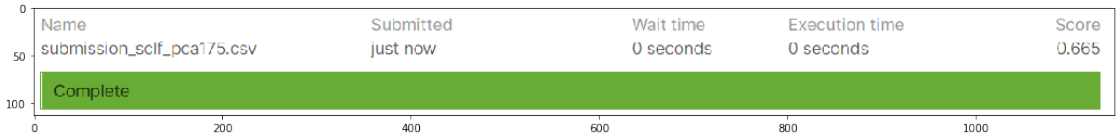
```
                random_state=42,
                solver='liblinear',
                tol=0.0001,
                verbose=0,
                warm_start=False),
                                                        cv=3,
                                                        method='sigmoid'),
                      store_train_meta_features=False, use_clones=True,
                      use_features_in_secondary=False, use_probas=True, verbose=0)
```

19 7.6.1 Kaggle score

[51]:
```
# 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_pca175.csv', index=False)
```

[52]:
```
image = plt.imread(data_dir+'/submission_sclf_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[52]:
```
<matplotlib.image.AxesImage at 0x218546b66c8>
```



20 7.7 Voting Classifier (Without Stack Classifier + no weights)

[53]:
```
# Import Voting Classifier
from mlxtend.classifier import EnsembleVoteClassifier
```

[54]:
```
# 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)
```

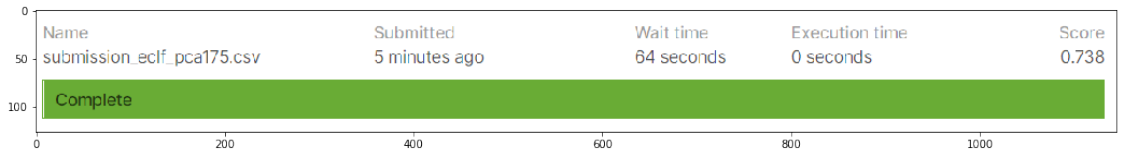
[54]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=100,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      random_state=42,
      solver='liblinear',
      tol=0.0001,
      verbose=0,
      warm_start=False),
                                                    cv=3, method='sigmoid'),
                          CalibratedClassi…
    max_depth=5,
    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=400,
    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)
```

21 7.7.1 Kaggle Score

[55]:
```
# 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_pca175.csv', index=False)
```

[56]:
```
image = plt.imread(data_dir+'/submission_eclf_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[56]:
```
<matplotlib.image.AxesImage at 0x218547e5248>
```



22 7.8 Voting Classifier (With Stack Classifier + no weights)

[57]:
```
# 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)
```

[57]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=100,
      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,
```

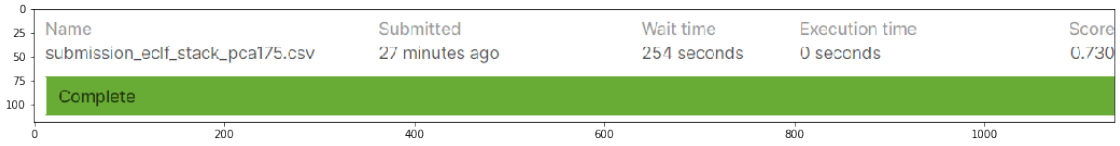
```
                                        penalty='l1',
                                        random_state=42,
                                        solver='liblinear',
                                        tol=0.0001,
                                        verbose=0,
                                        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)
```

23 7.8.1 Kaggle Score

[58]:
```
# 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_pca175.csv', index=False)
```

[59]:
```
image = plt.imread(data_dir+'/submission_eclf_stack_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[59]:
```
<matplotlib.image.AxesImage at 0x21860b4d448>
```



24 7.9 Voting Classifier (without Stack Classifier + weights)

[60]:
```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4], weights=[0.3,0.3,0.
 ↪15,0.25])
# Fit the train data
```

```
eclf.fit(tr_X,tr_y)
```

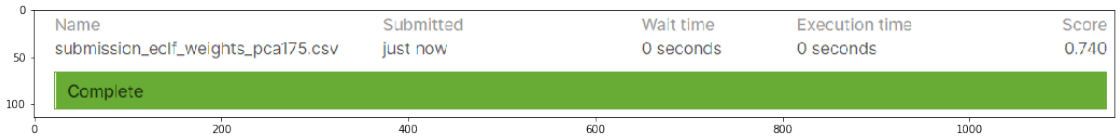
[60]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=100,
      class_weight='balanced',
      dual=False,
      fit_intercept=True,
      intercept_scaling=1,
      l1_ratio=None,
      max_iter=100,
      multi_class='auto',
      n_jobs=None,
      penalty='l1',
      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=400,
          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=[0.3, 0.3, 0.15, 0.25])
```

7.9.1 Kaggle Score

[61]:
```
# 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_pca175.csv', index=False)
```

[62]:
```
image = plt.imread(data_dir+'/submission_eclf_weights_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[62]:
```
<matplotlib.image.AxesImage at 0x21860b51f88>
```



25 7.10 Voting Classifier (with Stack Classifier + weights)

[64]:
```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4,sclf], weights=[0.3,0.
 ↪4,0.1,0.1,0.1])
# Fit the train data
eclf.fit(tr_X,tr_y)
```

[64]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(C=100,
      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',
```

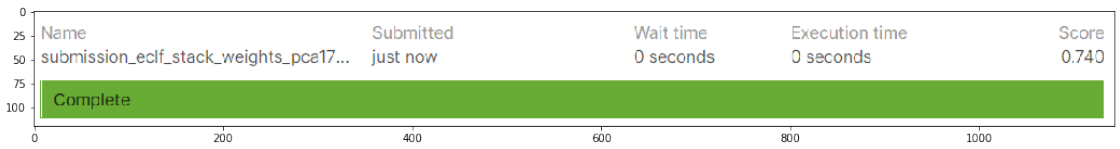
```
                                        random_state=42,
                                        solver='liblinear',
                                        tol=0.0001,
                                        verbose=0,
                                        warm_start=False),
            cv=3,
            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=[0.3, 0.4, 0.1, 0.1, 0.1])
```

26 7.10.1 Kaggle Score

[65]:
```
# 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_pca175.csv',␣
 ↪index=False)
```

[66]:
```
image = plt.imread(data_dir+'/submission_eclf_stack_weights_pca175.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[66]:
```
<matplotlib.image.AxesImage at 0x21861680348>
```



27 6. Summary of All Models

[1]:
```
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': 19}",0.642])
```

```
x.add_row(['Logistic Regression','AF',r"{'C': 100, 'penalty': 'l1', 'solver':␣
↪'liblinear'}",0.717])
x.add_row(['SVC','AF',r"{'C': 0.1, 'kernel': 'linear'}",0.725])
x.add_row(['RandomForest','AF',r"{'max_depth': 5, 'n_estimators': 400}",0.672])
x.add_row(['XGBoost','AF',r"{'max_depth': 2, 'n_estimators': 10}",0.673])
x.add_row(['Stacking Classifier','AF','-',0.665])
x.add_row(['Voting Classifier(No stacking + no weights)','AF',"-",0.738])
x.add_row(['Voting Classifier(stacking + no weights)','AF',"-",0.730])
x.add_row(['Voting Classifier(no stacking + weights)','AF',"-",0.740])
x.add_row(['Voting Classifier(stacking + weights)','AF',"-",0.740])
print(x)
```

```
+-----------------------------------------------+----------+----------------------
------------------------------+------------+
|                     Model                     | Features |
Hyperparameter                    | Test Score |
+-----------------------------------------------+----------+----------------------
------------------------------+------------+
|                      knn                      |    AF    |    {'algorithm':
'kd_tree', 'n_neighbors': 19}     |   0.642    |
|             Logistic Regression               |    AF    | {'C': 100, 'penalty':
'l1', 'solver': 'liblinear'} |   0.717    |
|                      SVC                      |    AF    |           {'C': 0.1,
'kernel': 'linear'}           |   0.725    |
|                 RandomForest                  |    AF    |       {'max_depth':
5, 'n_estimators': 400}        |   0.672    |
|                   XGBoost                     |    AF    |        {'max_depth':
2, 'n_estimators': 10}        |   0.673    |
|             Stacking Classifier               |    AF    |
-                             |   0.665    |
| Voting Classifier(No stacking + no weights) |    AF    |
-                             |   0.738    |
|   Voting Classifier(stacking + no weights)  |    AF    |
-                             |    0.73    |
|   Voting Classifier(no stacking + weights)  |    AF    |
-                             |    0.74    |
|    Voting Classifier(stacking + weights)    |    AF    |
-                             |    0.74    |
+-----------------------------------------------+----------+----------------------
------------------------------+------------+
```

Notation: 1. AF: All features

[ ]: