3_3_Models

April 20, 2020

1 SMOTE + Standardization + ML

1. SMOTE → Oversampling technique (called Synthetic Minority Oversampling Technique)
2. No 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
# Oversampling technique: SMOTE
from imblearn.over_sampling import SMOTE
# Data is umbalance, we need Calibrated Model to ive confidence probabilities␣
 ↪result
from sklearn.calibration import CalibratedClassifierCV
# For Hyperparameter and CV Fold
from sklearn.model_selection import GridSearchCV, StratifiedKFold
# For plot AUROC graph
import matplotlib.pyplot as plt
# For heatmap
import seaborn as sns
# To ignore warninga
import warnings
warnings.filterwarnings('ignore')
# To stndardize the data
from sklearn.preprocessing import StandardScaler
```

```
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
```

```
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
D:\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
```

```
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
D:\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated;
in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

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. Split and Oversampling data

[4]:

```
# Take separate for features value
X = df_train.drop(['id','target'], axis=1)
# Take separate for class value
y = df_train['target'].values
# Take test feature value
ts_X = df_test.drop(['id'], axis=1)
# Split the data into train and cv
tr_X, cv_X, tr_y, cv_y = train_test_split(X, y, test_size=0.1, stratify=y,␣
 ↪random_state=42)
# SMOTE (Ref: https://imbalanced-learn.readthedocs.io/en/stable/generated/
 ↪imblearn.over_sampling.SMOTE.html)
smote = SMOTE()
# Oversampling using SMOTE technique
tr_X, tr_y = smote.fit_sample(tr_X, tr_y)
```

5 4. Standardization

[5]:

```
# Fit and transform on train data
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  0.339626 -0.956317 -1.237863  0.127112  0.654213  0.262920 -1.376624
1  0.307378 -1.099808  0.211797 -0.515096 -0.104340 -1.068819  1.289393
2 -0.237721  0.626275 -0.000977  0.413215  1.402811 -1.372337 -1.932263
3 -0.146178  0.558196  0.425646  1.192387 -1.528866  0.560939  0.391984
4 -1.598390  0.284829 -0.630704 -0.136670 -0.429860  0.107862  2.071481

         7         8         9    …       290       291       292       293  \
0  1.415930  1.417942 -0.831692  …  0.206119  1.233815  1.546416 -1.393713
1 -1.601171  1.130549 -0.712051  …  0.460675 -0.824870 -1.727832  0.653609
2  1.020820 -2.435939 -1.379204  … -1.474583  0.181956  0.982611  0.158128
3  0.613256 -1.477601 -1.889201  … -0.159204  0.188389 -0.273423 -0.233158
4 -0.627547 -0.244834 -0.128039  … -1.979433  1.470778 -1.945001 -0.449307
```

```
        294       295       296       297       298       299
0  2.068083  0.944863 -0.025092 -1.323833  1.439064  0.997697
1  1.179357 -0.094636  0.565596 -0.809470 -0.724716  0.959973
2  0.712383  0.082840 -0.064609 -0.755493 -0.015841  0.118191
3 -2.442397  0.740005  0.051864  1.627936 -0.315035 -1.372442
4  1.028942  1.387030 -1.180468  0.212907 -0.055929  0.929794

[5 rows x 300 columns]
```

[6]:

```
# Transform on cv data based on mean and std on train data
cv_X = stand_vec.transform(cv_X)
pd.DataFrame(cv_X).head(5)
```

[6]:

```
          0         1         2         3         4         5         6    \
0 -0.856680 -1.968087 -0.373870 -0.748442  0.031045 -0.898366  0.582789
1  1.781435  0.871362 -1.788068 -0.904682  0.575570 -0.246242  0.814479
2 -1.404900 -0.127839 -0.616734  0.357415  0.924982 -1.658260  1.727614
3 -2.272482 -2.583947  0.389109 -0.681482 -1.139635  1.096494  0.268276
4 -0.042151  0.335103  1.989216 -0.712933  1.151951 -2.750263 -0.428893

          7         8         9    …       290       291       292       293  \
0 -0.244891  0.247840 -0.675550  …  1.084814 -0.260876  0.348852 -0.351762
1 -0.130546 -1.512174  0.708437  …  2.063626 -0.713358 -0.438388  1.292080
2  0.118520  2.798727  0.852412  … -1.037899  1.528679  0.346763 -0.492537
3 -0.347914 -0.149757  0.704382  …  0.985761 -0.216915  1.905581 -0.543526
4 -0.633208 -0.212421 -1.645862  … -0.618256  2.779974 -0.904049  0.235720

        294       295       296       297       298       299
0 -1.566452 -0.476968 -1.071274  0.953760 -0.568275 -0.154499
1  0.222797 -0.091593  1.440187 -1.420144 -0.374678 -0.016537
2 -1.562519  1.814999  0.399205 -0.289814 -1.232173  0.535310
3  0.890324  0.328263 -0.612659 -0.939648  0.272598  1.909537
4  1.031891  0.095009  0.186017 -0.245363  0.704767 -1.436033

[5 rows x 300 columns]
```

[7]:

```
# Transform on test data based on mean and std value from train data
ts_X = stand_vec.transform(ts_X)
pd.DataFrame(ts_X).head(5)
```

[7]:

```
          0         1         2         3         4         5         6    \
0  0.511270 -1.131230 -1.912723  0.302629 -0.732485  0.543344  0.514644
1  0.798384  0.908021 -0.729569  1.355730 -0.884793  0.519150  0.559724
2  1.811603  0.483832 -0.259960  0.836281 -0.495562  1.561668 -0.781147
3 -0.587251 -1.992177 -0.931597  0.575543  1.563082  0.554341 -1.533880
4  0.775498 -0.305894  1.061820 -1.657477 -0.012756  0.398183  0.772544
```

```
          7         8         9  …       290       291       292       293  \
0 -0.282251 -0.657557  1.217420  … -0.124057 -2.869616 -0.980268  2.283041
1 -0.490561  0.304022  1.283324  … -0.757782 -0.122558 -0.071914  0.651392
2 -2.410636 -1.404132  0.077785  … -0.130447  0.324562 -0.731776 -0.837267
3  0.681184  0.636794  0.483349  … -0.388197 -0.895637  0.168225 -0.054696
4 -2.132135 -1.462475 -1.034475  …  2.295814 -1.220524  0.127506  1.294297

        294       295       296       297       298       299
0 -0.160613  2.195303  0.563516 -0.117301  0.265753  1.174460
1 -0.235329 -1.115879 -3.333149  0.290168 -0.650406 -0.700956
2  0.081230  0.744062  0.945175  0.135647 -1.336793  2.773953
3 -0.710168  0.962103  2.437492  1.107223 -1.604699 -1.528726
4 -0.028878  0.359701 -0.140525 -0.172336 -1.166663  1.008475

[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
    '''
    # Perform KCrossValidation with stratified target
    str_cv = StratifiedKFold(n_splits=10, random_state=42)
    # Perform Hyperparamter using GridSearchCV
    grid_clf = GridSearchCV(models, params, cv=str_cv, return_train_score=True,␣
 ↪scoring='roc_auc')
    # Fit the train model to evaluate score
    grid_clf.fit(tr_X, tr_y)
    return grid_clf

# Ref: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.
 ↪html
def plot_roc(try_true, try_pred, cvy_true, cvy_pred, n_classes):
    '''
    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
    '''
    # For train
    tr_fpr = dict()
    tr_tpr = dict()
    tr_roc_auc = dict()
    for i in range(n_classes):
        tr_fpr[i], tr_tpr[i], _ = roc_curve(try_true, try_pred[:, i])
        tr_roc_auc[i] = auc(tr_fpr[i], tr_tpr[i])

    # For cv
    cv_fpr = dict()
    cv_tpr = dict()
    cv_roc_auc = dict()
    for i in range(n_classes):
        cv_fpr[i], cv_tpr[i], _ = roc_curve(cvy_true, cvy_pred[:, i])
        cv_roc_auc[i] = auc(cv_fpr[i], cv_tpr[i])

    # Line thickness
    lw = 2
    # Plot roc for train
    plt.plot(tr_fpr[1], tr_tpr[1], color='red',
            lw=lw, label='ROC curve for Train (area = %0.2f)' % tr_roc_auc[1])
    # Plot roc for cv
    plt.plot(cv_fpr[1], cv_tpr[1], color='green',
            lw=lw, label='ROC curve for CV (area = %0.2f)' % cv_roc_auc[1])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic: train vs cv')
    plt.legend(loc="lower right")
    plt.show()

def plot_feature_importance(model, model_name, top_n = 10):
    '''
    Plot the feature importance on the basis of model.
    Parameters:
    model: Instance of model
    model_name: Name of the model
    top_n: Number of feature you want to print top features

    Return:
    df: DataFrame that return feature names with coefficient in descending order
    Plot the feature importance
    '''


    # Numpy Column Stack (See Docs: https://docs.scipy.org/doc/numpy-1.10.1/
 ↪reference/generated/numpy.column_stack.html)

    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
```

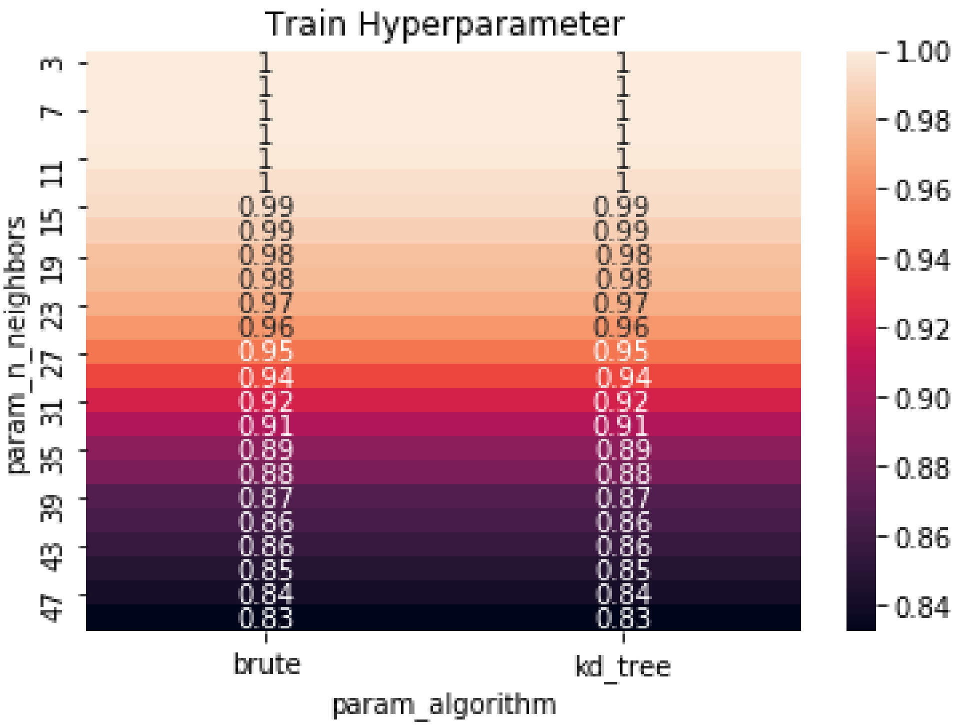
7 5.1 kNN

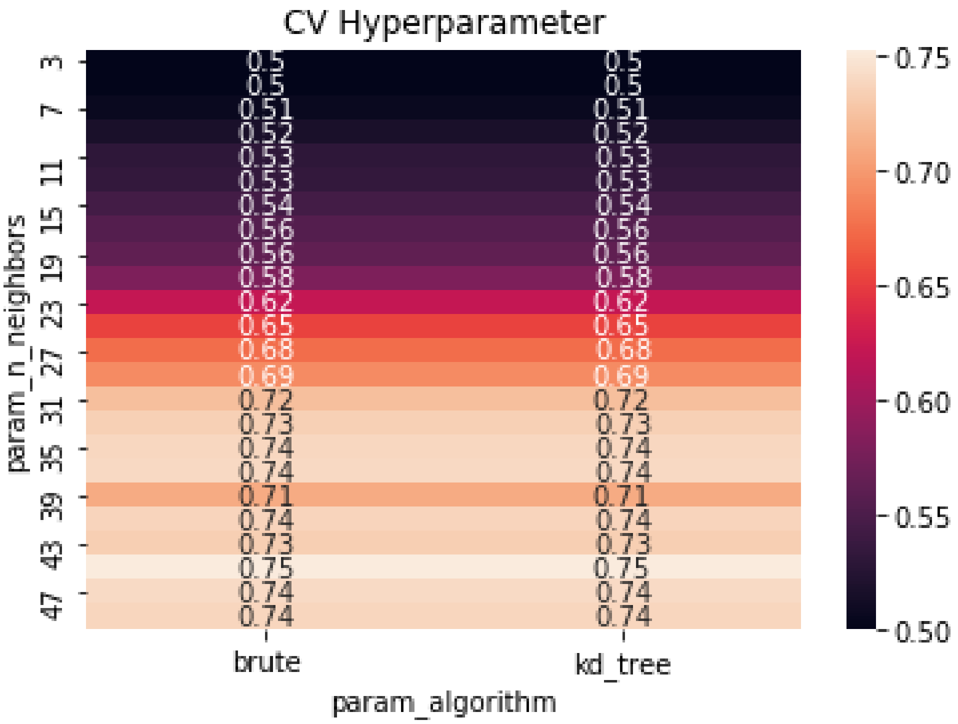
[9]:
```
# Import KNN
from sklearn.neighbors import KNeighborsClassifier
```

[14]:
```
# 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)
```

[15]:

```
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()
```





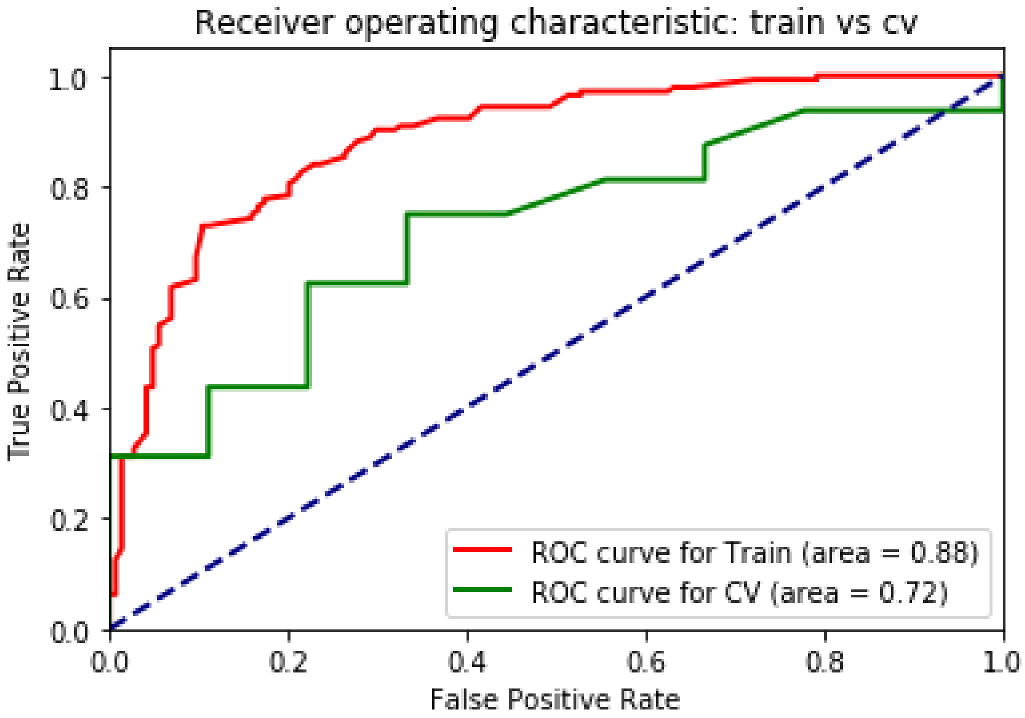
[16]:

```
print(knn_clf.best_params_)
print('CV Score',knn_clf.score(cv_X,cv_y))
```

```
{'algorithm': 'kd_tree', 'n_neighbors': 45}
CV Score 0.65625
```

[17]:

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

tr_pred = clf.predict_proba(tr_X)
cv_pred = clf.predict_proba(cv_X)

# Plot ROC cureve of train and cv data
plot_roc(tr_y, tr_pred, cv_y, cv_pred, 2)
```

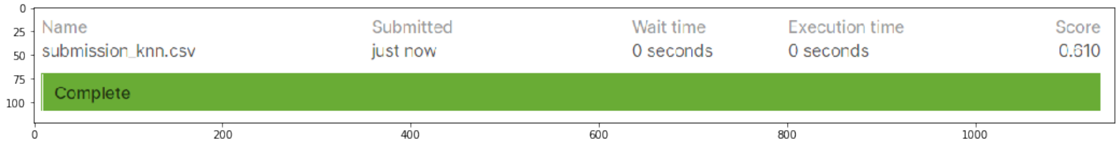


8 5.1.1 Kaggle Score

```
[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.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 0x1ff4bcd1588>
```



Name	Submitted	Wait time	Execution time	Score
submission_knn.csv	just now	0 seconds	0 seconds	0.610

Complete

8.1 5.2 Logistic Regression

[9]:

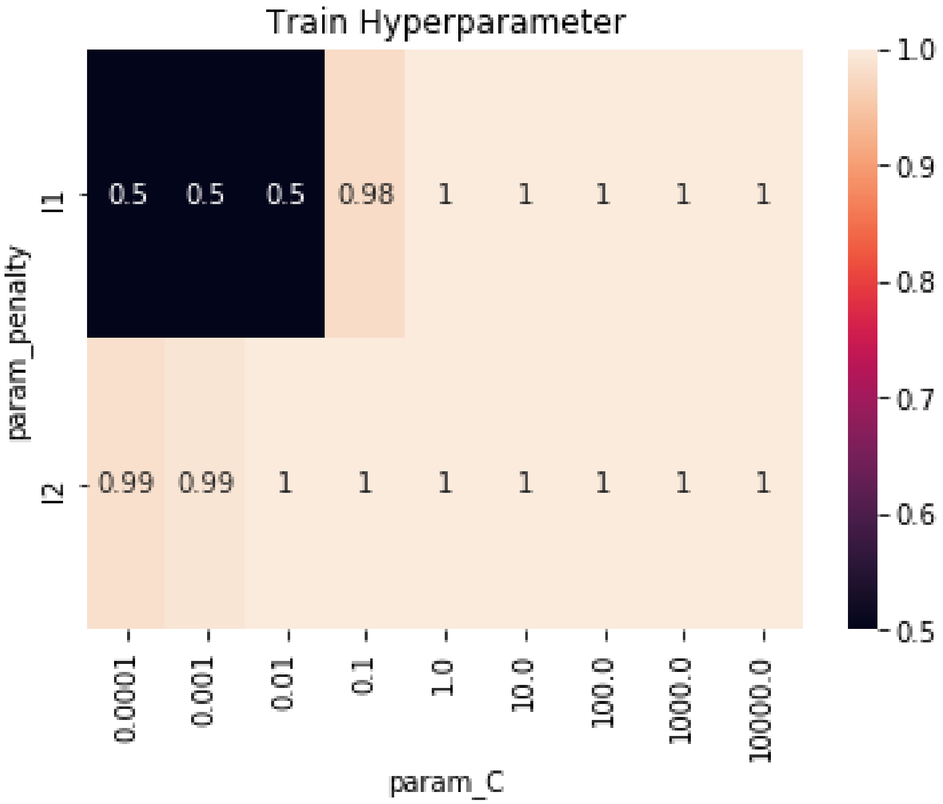
```
# Import Logistic Regression
from sklearn.linear_model import LogisticRegression
```

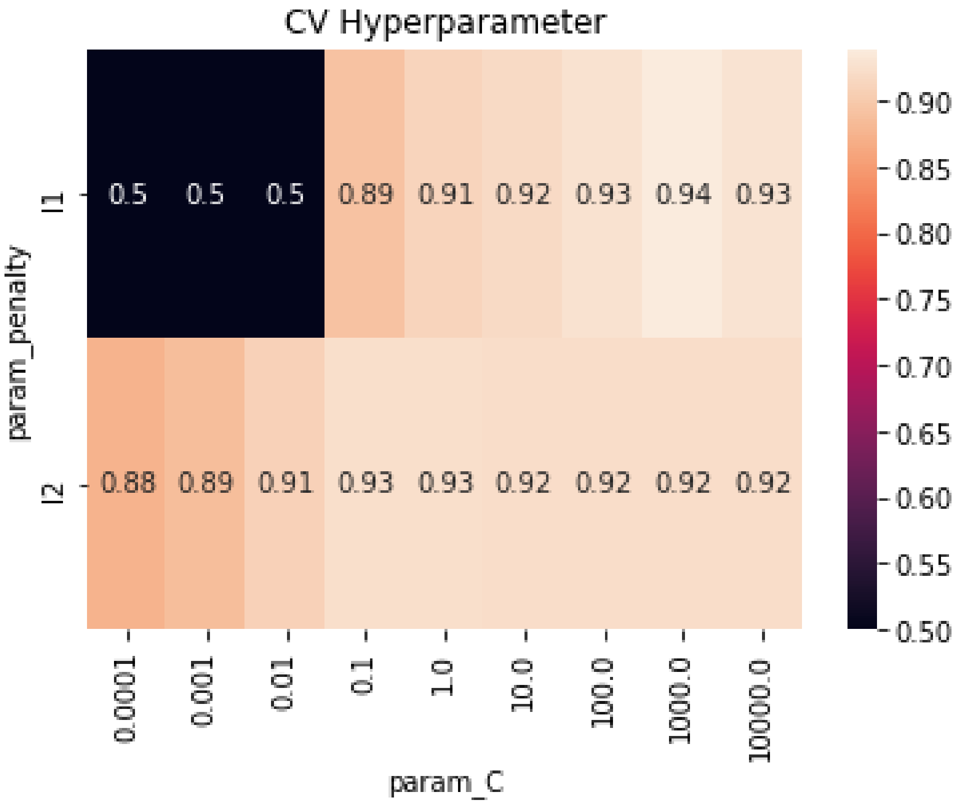
[30]:

```
# 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 logistic regression
log_model = LogisticRegression(random_state=42, class_weight='balanced')
# Call hyperparemeter to find the best params
log_clf = hyperparameter_model(log_model, params)
```

[31]:

```
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()
```

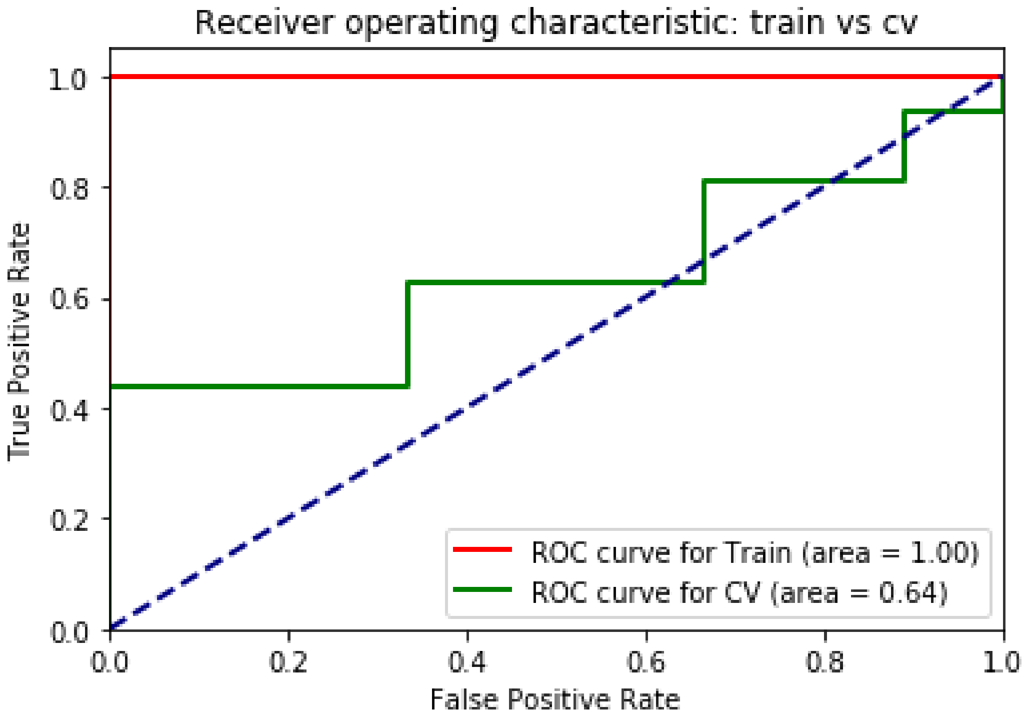

Train Hyperparameter
param_penalty
l1
l2
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
0.5
0.5
0.5
0.98
1
1
1
1
1
0.99
0.99
1
1
1
1
1
1
1
1.0
0.9
0.8
0.7
0.6
0.5



```
[32]: print(log_clf.best_params_)
      print('cv score',log_clf.score(cv_X,cv_y))
```

```
{'C': 1000, 'penalty': 'l1', 'solver': 'liblinear'}
cv score 0.6944444444444444
```

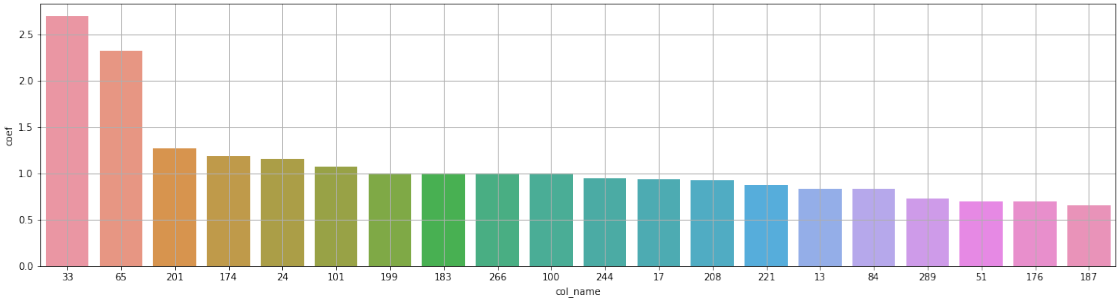
```
[33]: clf = CalibratedClassifierCV(log_clf, cv=3)
      clf.fit(tr_X,tr_y)

      tr_pred = clf.predict_proba(tr_X)
      cv_pred = clf.predict_proba(cv_X)

      # Plot ROC cureve of train and cv data
      plot_roc(tr_y, tr_pred, cv_y, cv_pred, 2)
```



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

```
[34]: LogisticRegression(C=1000, 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)
```

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

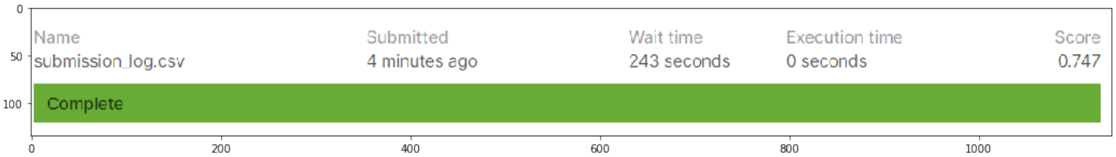


8.2 5.2.1 Kaggle Score

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

[37]:
```
image = plt.imread(data_dir+'/submission_log.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[37]:
```
<matplotlib.image.AxesImage at 0x1ff55f16ec8>
```



Name	Submitted	Wait time	Execution time	Score
submission_log.csv	4 minutes ago	243 seconds	0 seconds	0.747

Complete

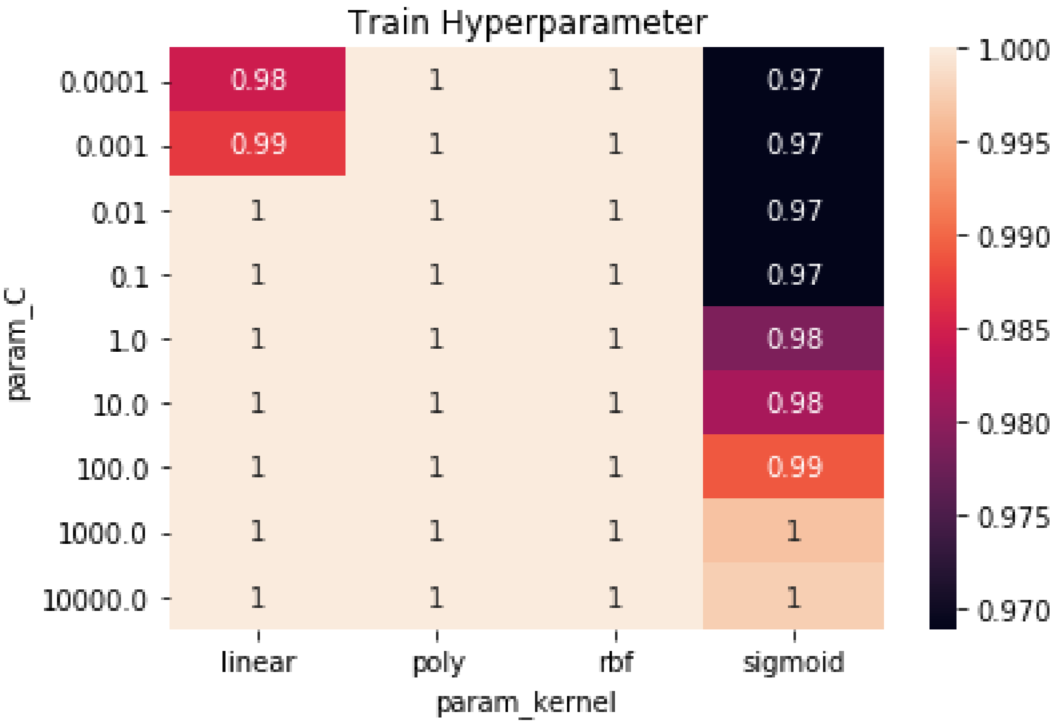
8.3 5.3 SVC

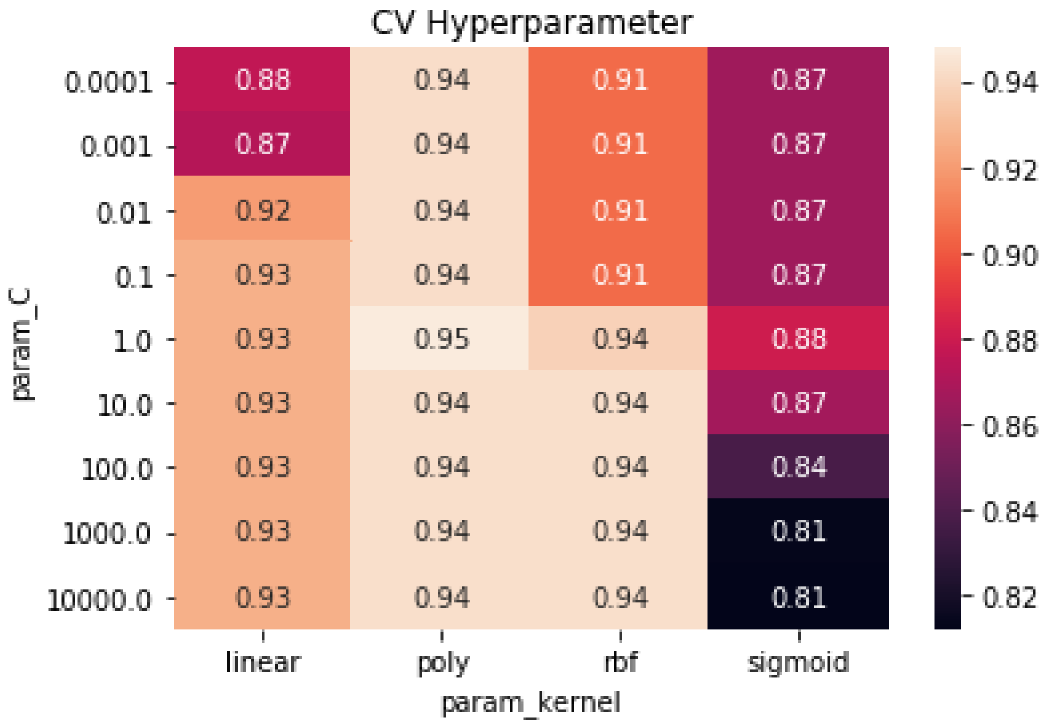
[10]:
```
# Import SVC
from sklearn.svm import SVC
```

[39]:
```
# 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)
```

[40]:
```
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()
```





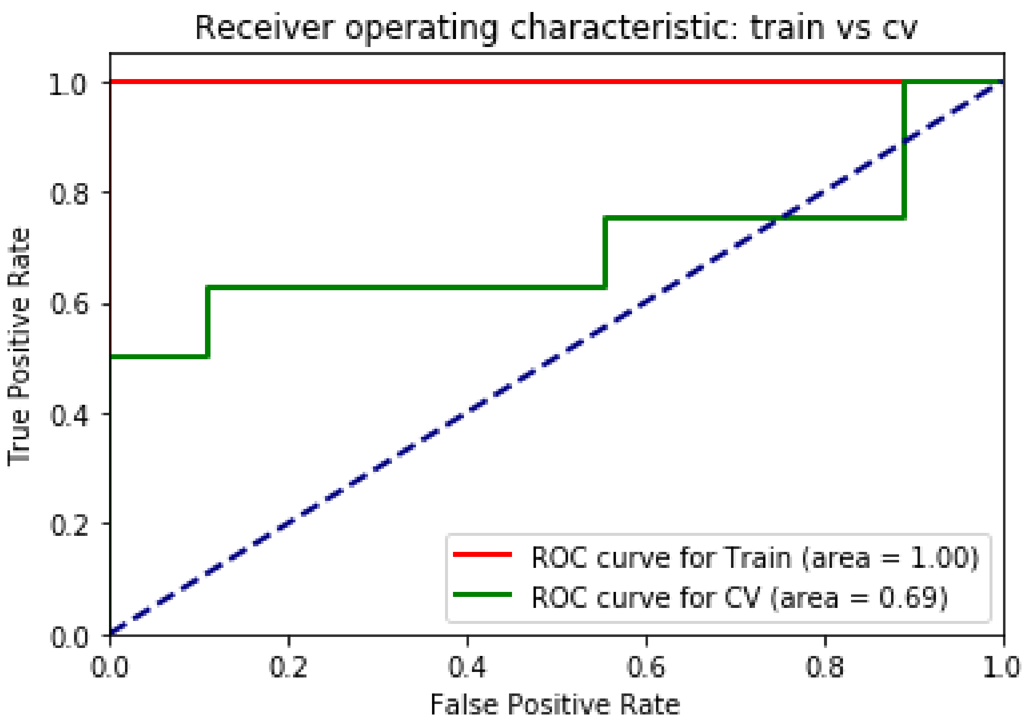
[41]:

```
print(svc_clf.best_params_)
print('cv Score',svc_clf.score(cv_X,cv_y))
```

```
{'C': 1, 'kernel': 'poly'}
cv Score 0.7083333333333333
```

[42]:

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

tr_pred = clf.predict_proba(tr_X)
cv_pred = clf.predict_proba(cv_X)

# Plot ROC curve of this model
plot_roc(tr_y, tr_pred, cv_y, cv_pred, 2)
```

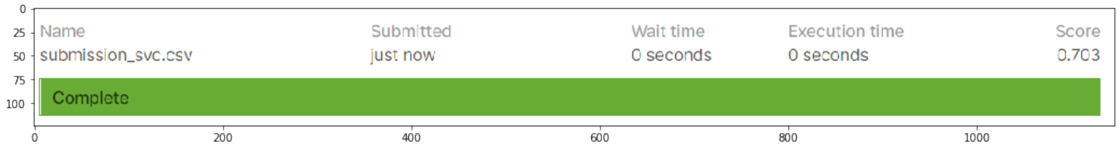


8.4 5.3.1 Kaggle Score

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

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

```
[53]: <matplotlib.image.AxesImage at 0x1ff552c4708>
```



Name	Submitted	Wait time	Execution time	Score
submission_svc.csv	just now	0 seconds	0 seconds	0.703

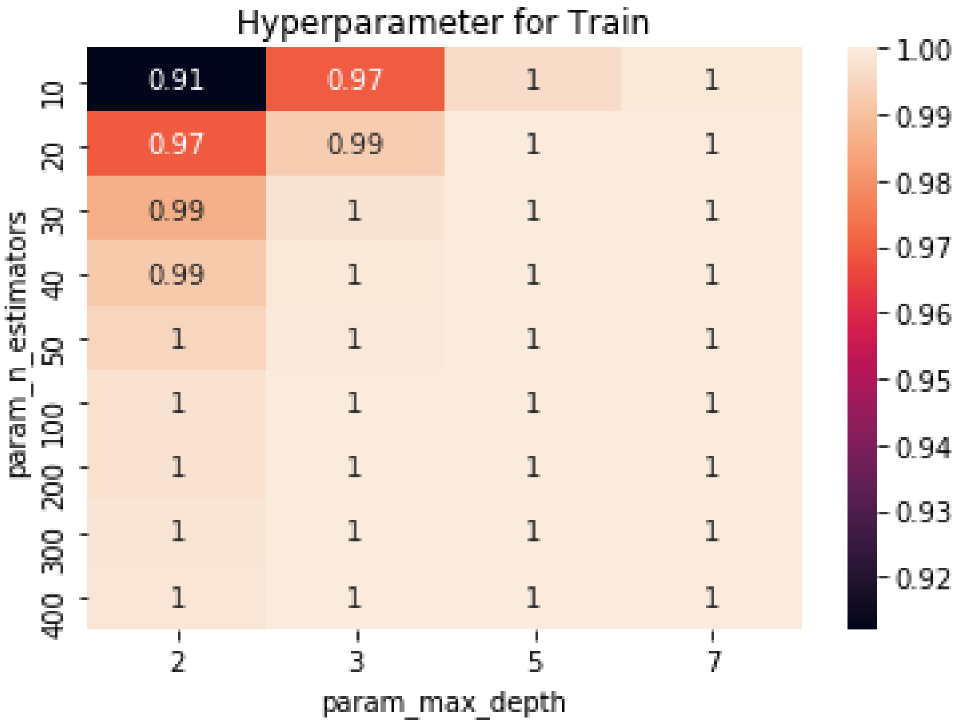
Complete

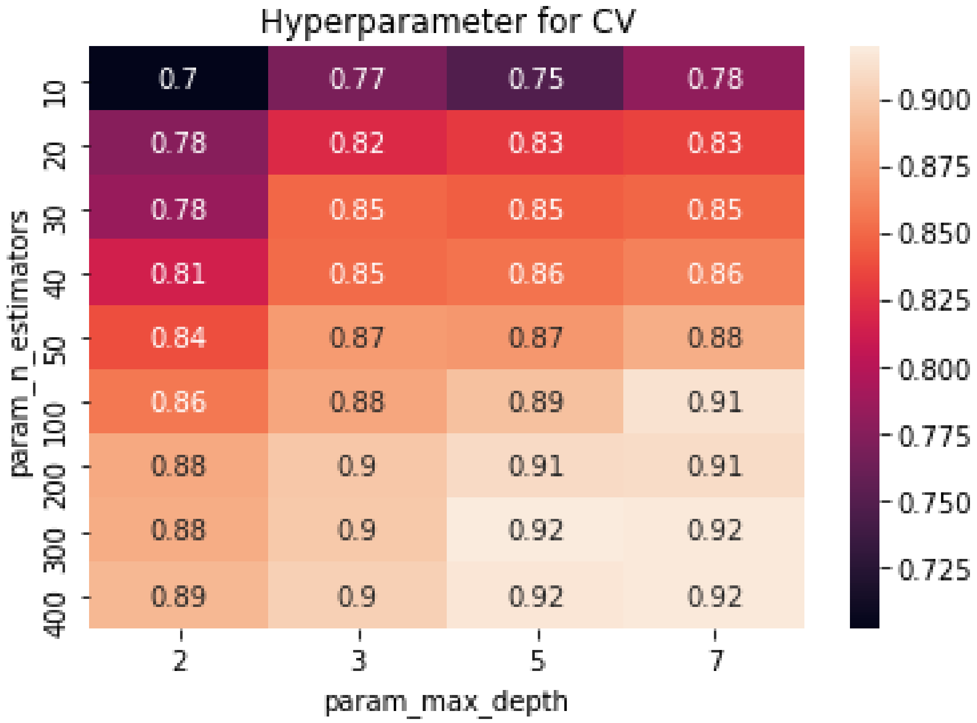
9 5.4 RandomForest

[11]:
```
# Impoer Random Forest
from sklearn.ensemble import RandomForestClassifier
```

[45]:
```
# 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)
```

[46]:
```
# 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
param_n_estimators
10
20
30
40
50
100
200
300
400
0.91
0.97
1
1
0.97
0.99
1
1
0.99
1
1
1
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
2
3
5
7
param_max_depth
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92


Hyperparameter for CV
param_n_estimators
10
20
30
40
50
100
200
300
400
0.7
0.77
0.75
0.78
0.78
0.82
0.83
0.83
0.78
0.85
0.85
0.85
0.81
0.85
0.86
0.86
0.84
0.87
0.87
0.88
0.86
0.88
0.89
0.91
0.88
0.9
0.91
0.91
0.88
0.9
0.92
0.92
0.89
0.9
0.92
0.92
2
3
5
7
param_max_depth
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725

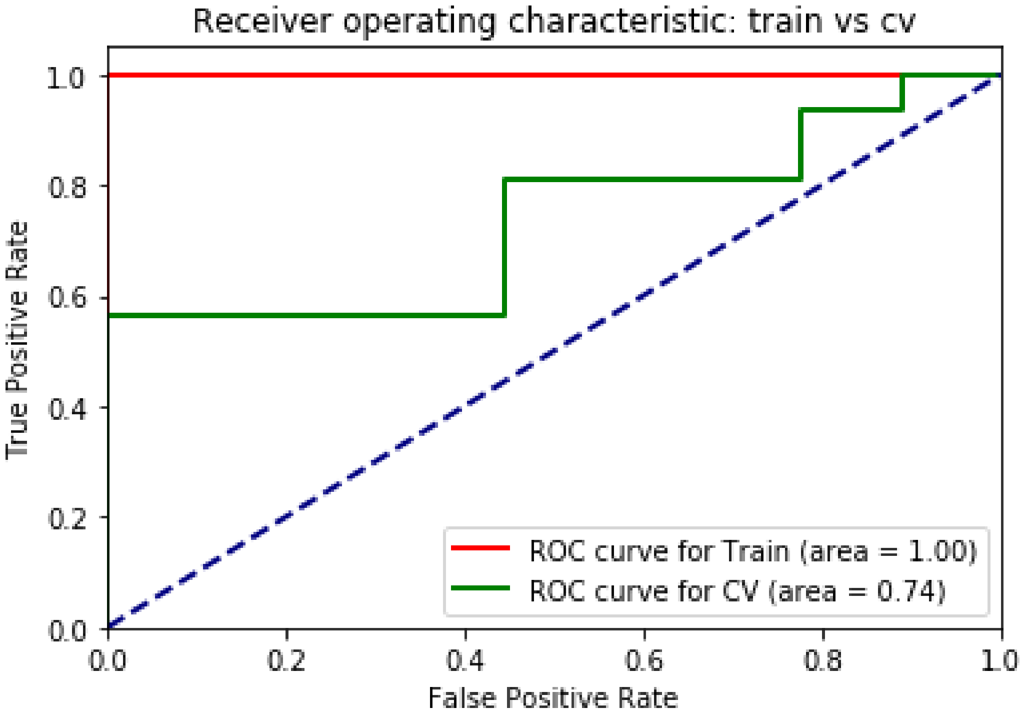
```
[47]: print(rf_clf.best_params_)
```

```
{'max_depth': 5, 'n_estimators': 300}
```

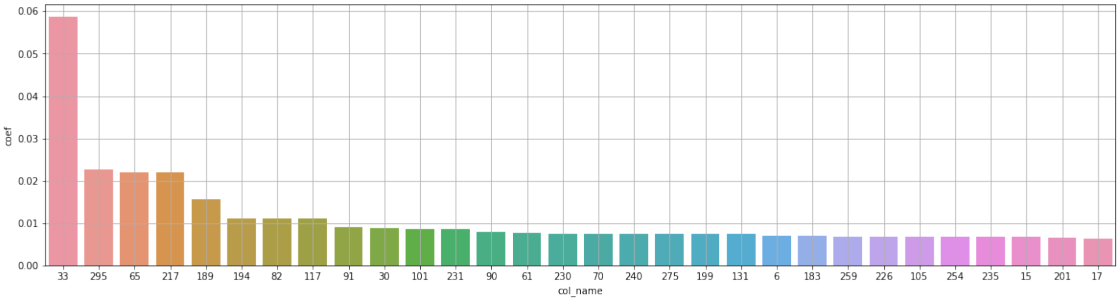
```
[48]: # Calibrate the model
      clf = CalibratedClassifierCV(rf_clf, cv=3)
      clf.fit(tr_X, tr_y)
```

```
[48]: CalibratedClassifierCV(base_estimator=GridSearchCV(cv=StratifiedKFold(n_splits=1
0, random_state=42, shuffle=False),
                                                   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=None,
    mi…
    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')
```

[49]:
```
# Plot ROC Curve of train and cv
plot_roc(tr_y, clf.predict_proba(tr_X), cv_y, clf.predict_proba(cv_X), 2)
```



[51]:
```
rf_clf = RandomForestClassifier(**rf_clf.best_params_, random_state=42)
rf_clf.fit(tr_X,tr_y)

# Plot the feature importance on the basis of this model
df = plot_feature_importance(rf_clf, 'rf',30)
```

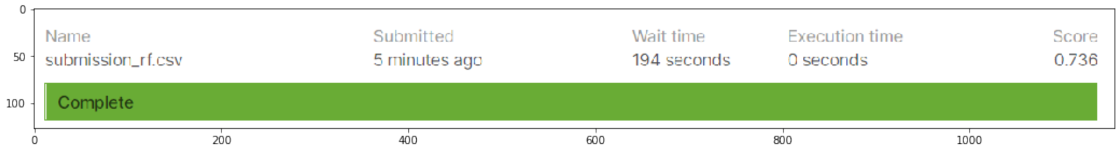


9.1 5.4.1 Kaggle Score

[52]:
```
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)
```

[54]:
```
image = plt.imread(data_dir+'/submission_rf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[54]:
```
<matplotlib.image.AxesImage at 0x1ff560ca3c8>
```



Name	Submitted	Wait time	Execution time	Score
submission_rf.csv	5 minutes ago	194 seconds	0 seconds	0.736

Complete

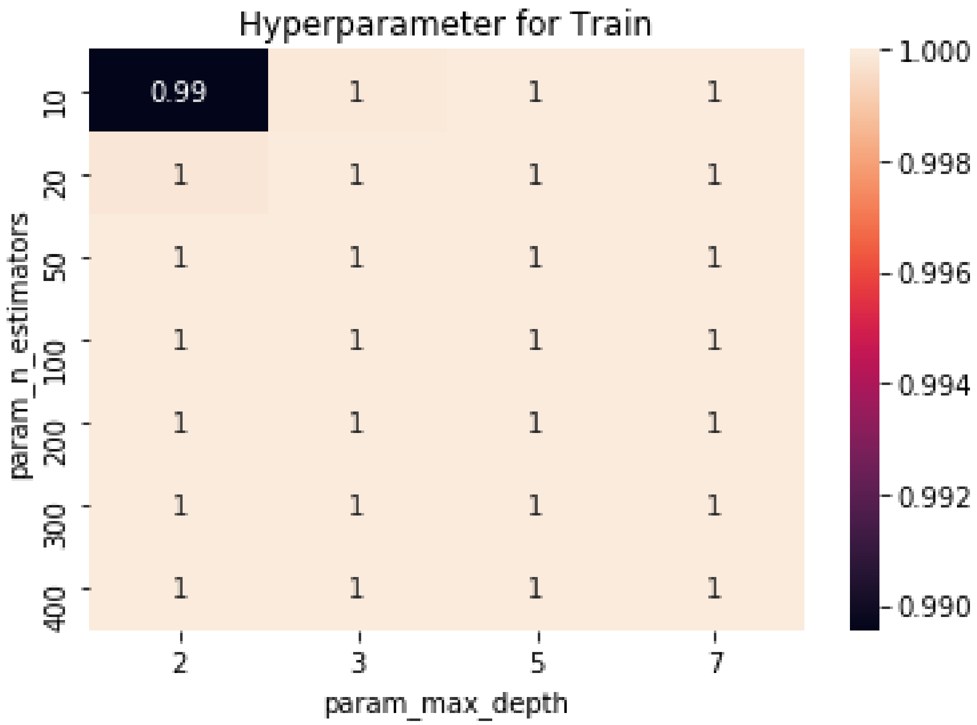
9.2 5.5 Xgboost

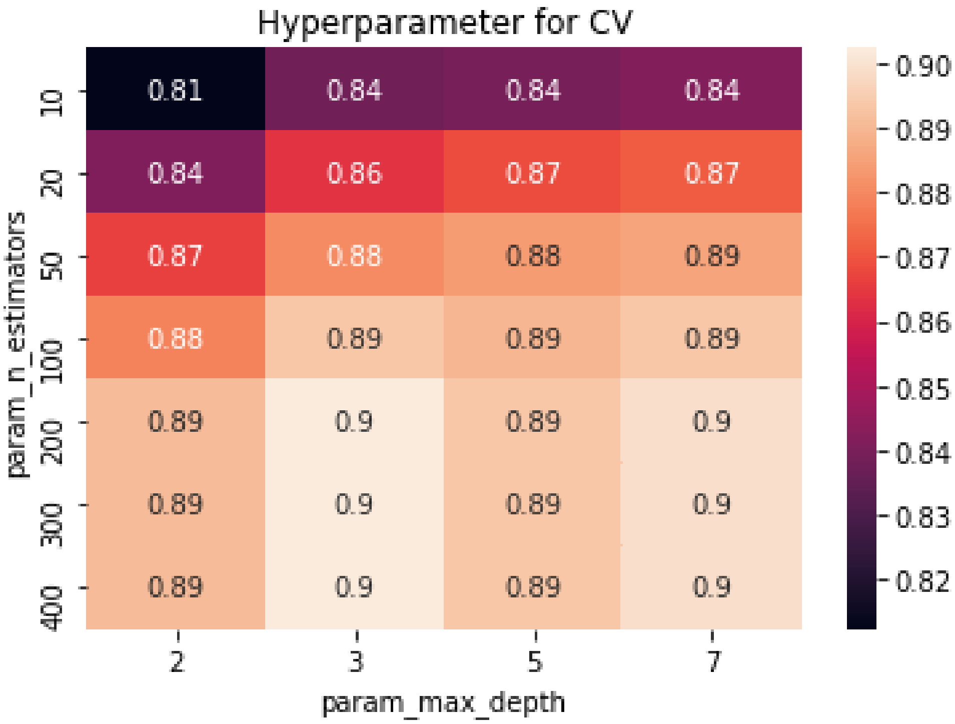
[12]:
```
# Import Xgboost
from xgboost import XGBClassifier
```

[56]:
```
# 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)
```

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





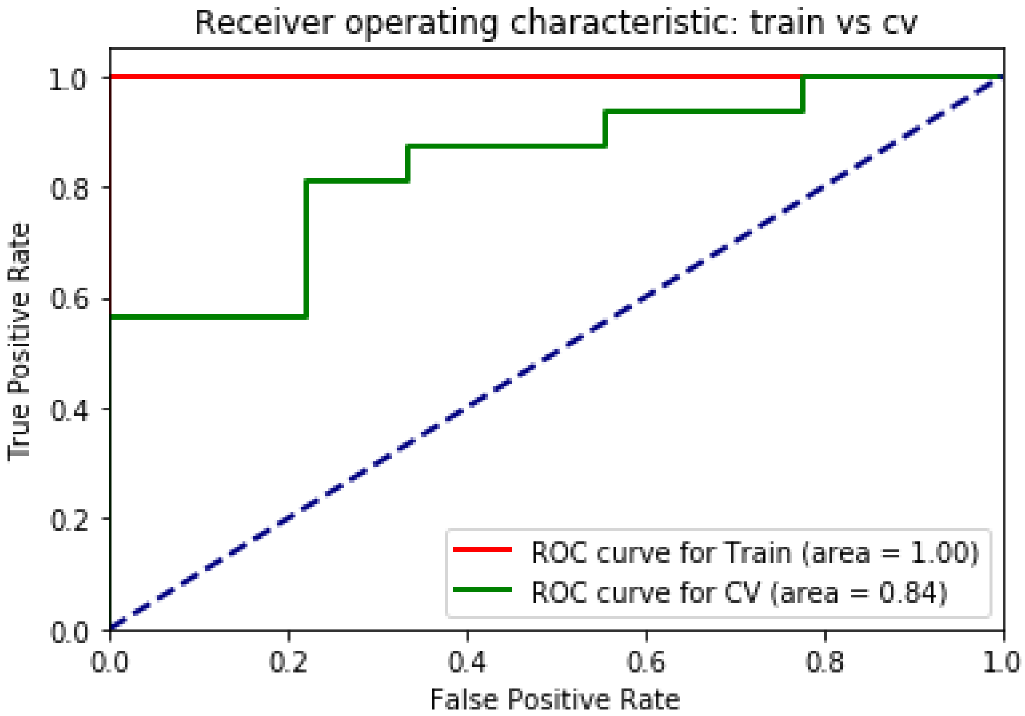
[58]:

```
print(xgb_clf.best_params_)
print('cv Score',xgb_clf.score(cv_X,cv_y))
```

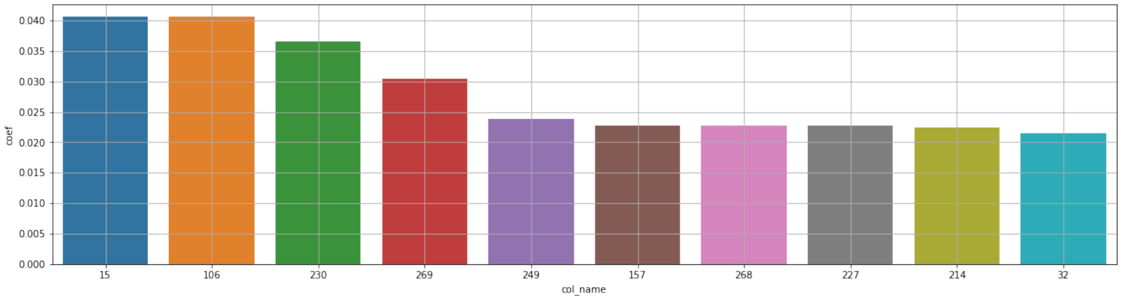
```
{'max_depth': 3, 'n_estimators': 200}
cv Score 0.7847222222222223
```

[59]:

```
# Instance of randomforest with best parameters
xgb_clf = XGBClassifier(**xgb_clf.best_params_, random_state=42,␣
 ↪scale_pos_weight=0.5)
# Fit the model
xgb_clf.fit(tr_X,tr_y)
# Calibrate the model
clf = CalibratedClassifierCV(xgb_clf, cv=3)
clf.fit(tr_X, tr_y)

tr_pred = clf.predict_proba(tr_X)
cv_pred = clf.predict_proba(cv_X)

# Plot ROC curve of train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred, 2)
```



[60]:
```
# Instance of XGBoost model with best parameters
df = plot_feature_importance(xgb_clf, 'xgb',10)
```

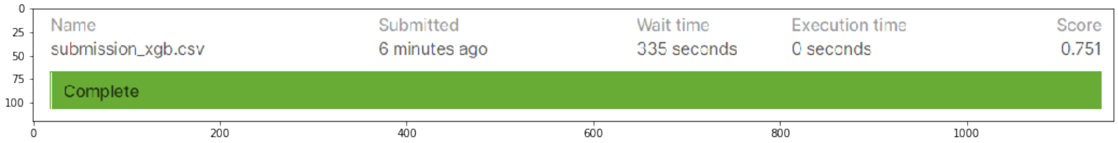


9.3 5.5.1 Kaggle Score

[63]:
```
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)
```

[69]:
```
image = plt.imread(data_dir+'/submission_xgb.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[69]:
```
<matplotlib.image.AxesImage at 0x1ff5f36ff08>
```



Name	Submitted	Wait time	Execution time	Score
submission_xgb.csv	6 minutes ago	335 seconds	0 seconds	0.751

Complete

9.4 5.6 Stacking Model

[13]:
```
# Import Stacking Classifier
from mlxtend.classifier import StackingClassifier
```

[14]:
```
# StackClassifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
↪classifier/StackingClassifier/#methods)

# Classifier 1: Logistic Regression with best params
clf1 = LogisticRegression(C = 1000, 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=1, kernel='poly', 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=3, n_estimators=200, 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=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)

# Predict in probabilities
tr_pred = sclf.predict_proba(tr_X)
cv_pred = sclf.predict_proba(cv_X)
```

[66]:

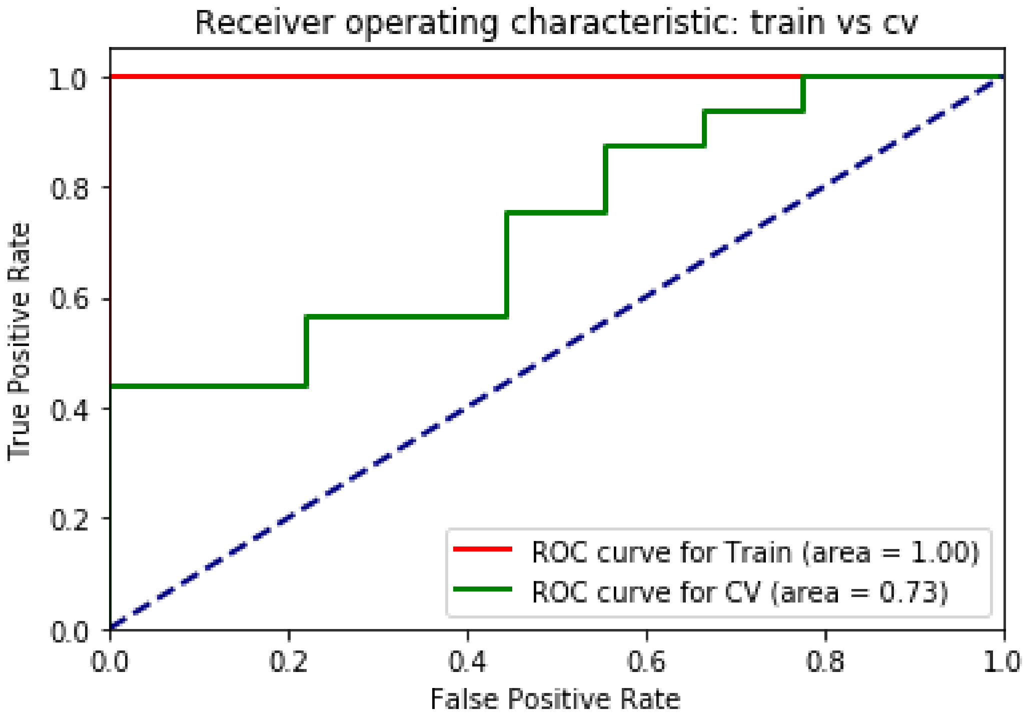
```
# Score after stacking classifier
sclf.score(cv_X, cv_y)
```

[66]:

```
0.68
```

[67]:

```
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```



9.5 5.6.1 Kaggle Score

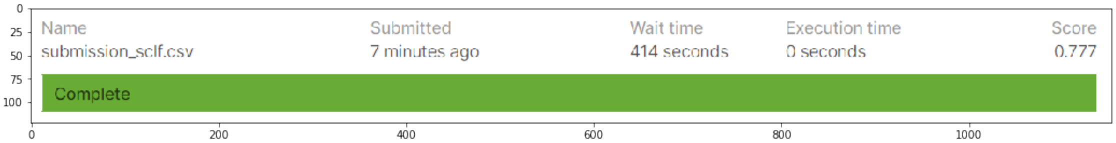
[68]:

```
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)
```

[70]:
```
image = plt.imread(data_dir+'/submission_sclf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[70]:
```
<matplotlib.image.AxesImage at 0x1ff5e0e7848>
```



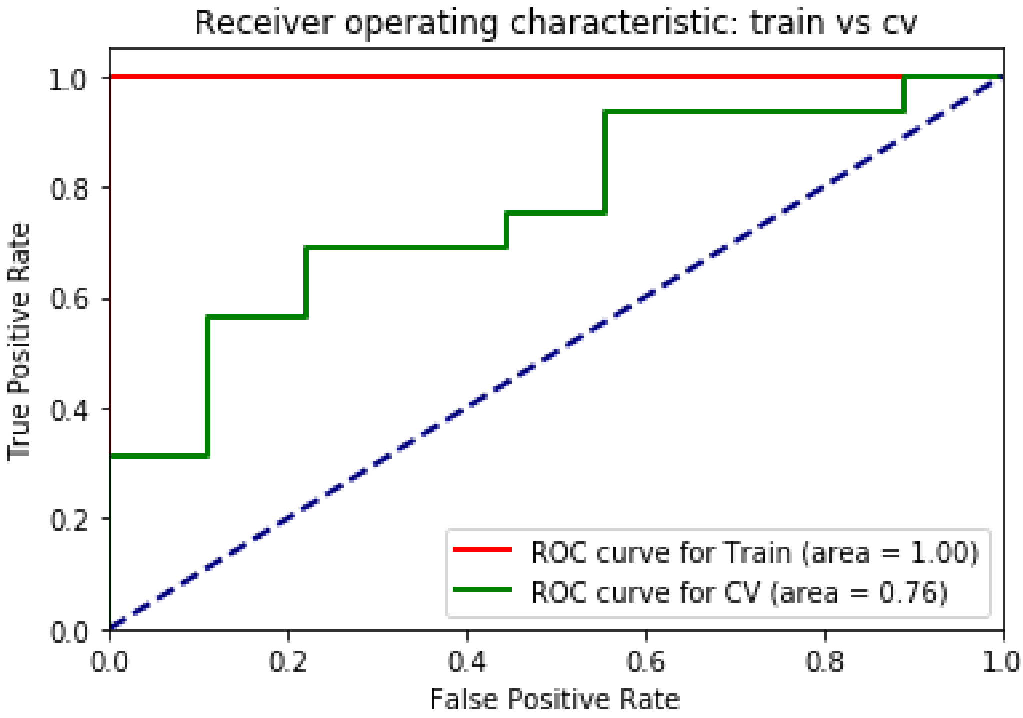
Name	Submitted	Wait time	Execution time	Score
submission_sclf.csv	7 minutes ago	414 seconds	0 seconds	0.777

Complete

10 5.7 Voting Classifier (Without Stack Classifier + no weights)

[16]:
```
# Import Voting Classifier
from mlxtend.classifier import EnsembleVoteClassifier
```

[17]:
```
# 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)

# Predict in probabilities
tr_pred = eclf.predict_proba(tr_X)
cv_pred = eclf.predict_proba(cv_X)
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```

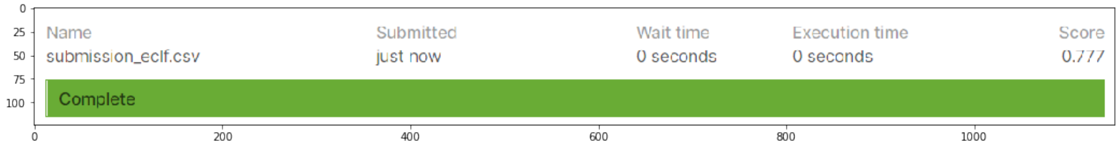


11 5.7.1 Kaggle Score

```
[18]: # 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)
```

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

```
[19]: <matplotlib.image.AxesImage at 0x1c723fc7488>
```



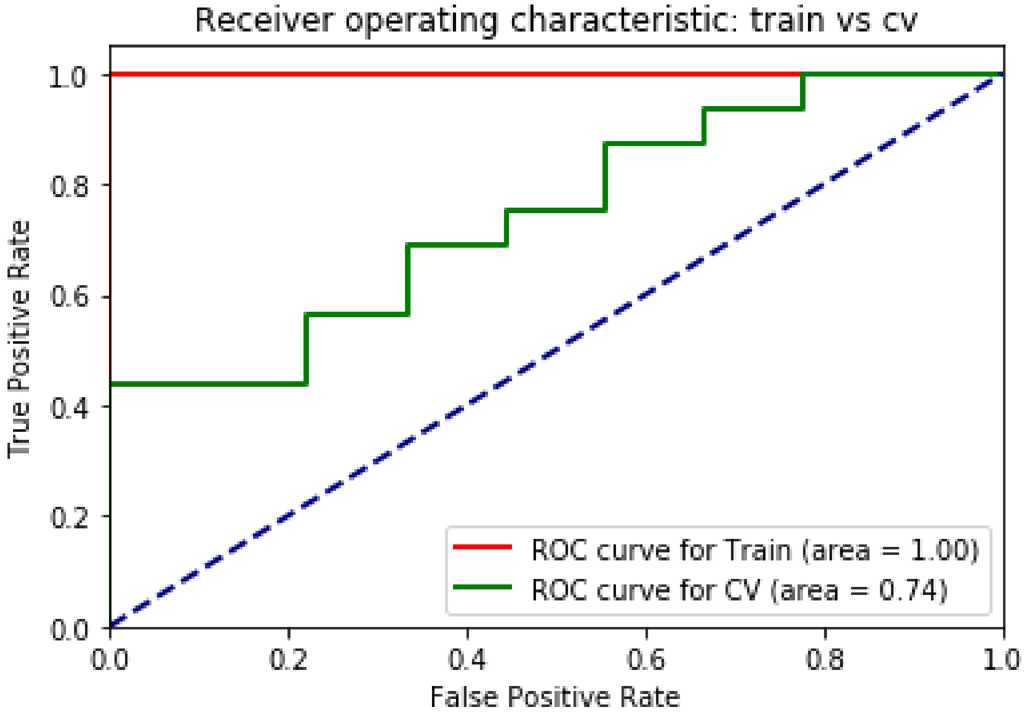
Name	Submitted	Wait time	Execution time	Score
submission_eclf.csv	just now	0 seconds	0 seconds	0.777

Complete

12 5.8 Voting Classifier (With Stack Classifier + no weights)

[20]:

```
# 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)

# Predict in probabilities
tr_pred = eclf.predict_proba(tr_X)
cv_pred = eclf.predict_proba(cv_X)
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```



13 5.8.1 Kaggle Score

[21]:

```
# 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')
```

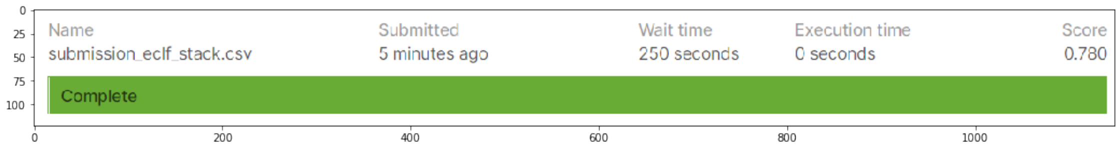
```
eclf_df.to_csv(data_dir+'/submission_eclf_stack.csv', index=False)
```

[23]:

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

[23]:

```
<matplotlib.image.AxesImage at 0x1c722df4108>
```



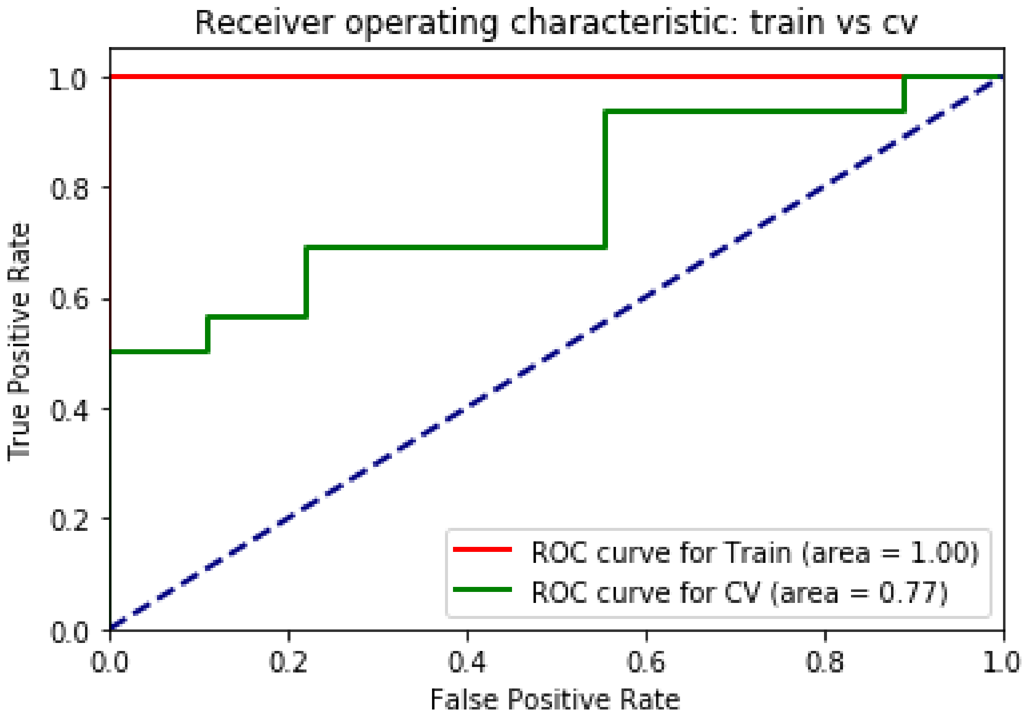
Name	Submitted	Wait time	Execution time	Score
submission_eclf_stack.csv	5 minutes ago	250 seconds	0 seconds	0.780

Complete

14 5.9 Voting Classifier (without Stack Classifier + weights)

[24]:

```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4], weights=[0.3,0.1,0.
↪3,0.3])
# Fit the train data
eclf.fit(tr_X,tr_y)

# Predict in probabilities
tr_pred = eclf.predict_proba(tr_X)
cv_pred = eclf.predict_proba(cv_X)
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```

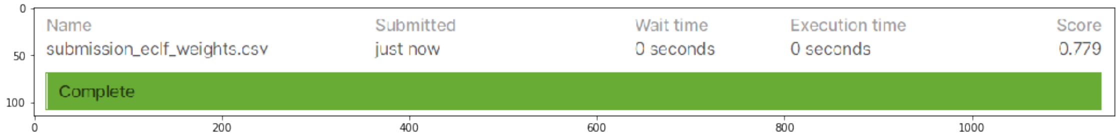


15 5.9.1 Kaggle Score

[25]:
```
# 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)
```

[26]:
```
image = plt.imread(data_dir+'/submission_eclf_weights.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[26]:
```
<matplotlib.image.AxesImage at 0x1c7232a5288>
```

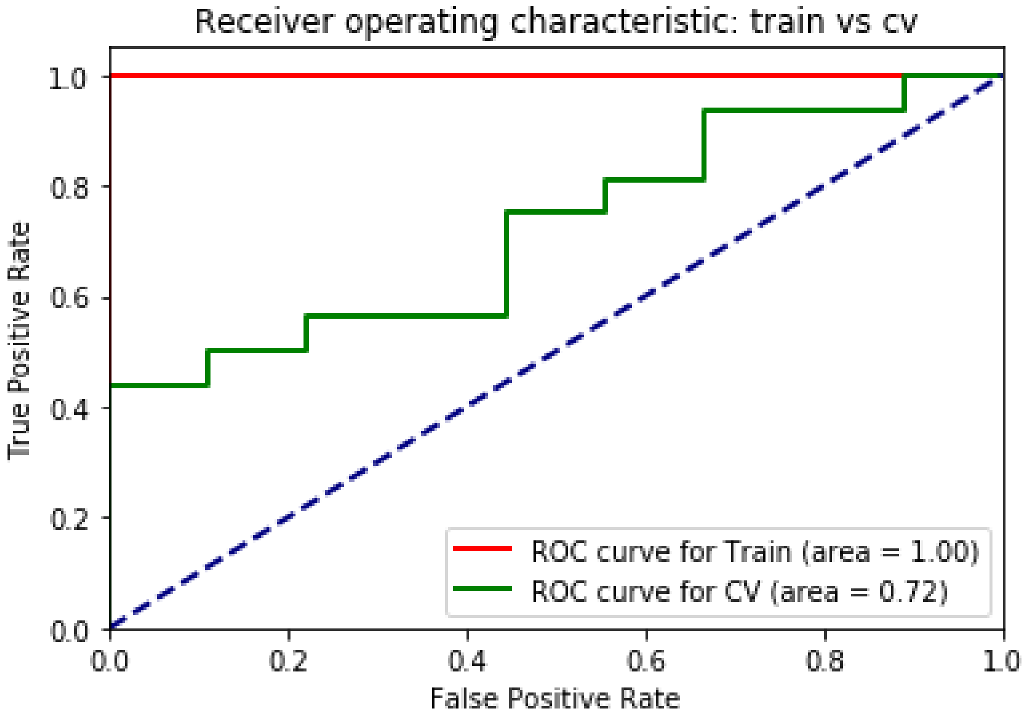


Name	Submitted	Wait time	Execution time	Score
submission_eclf_weights.csv	just now	0 seconds	0 seconds	0.779

Complete

16 5.10 Voting Classifier (with Stack Classifier + weights)

[27]:
```
# 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.
 ↪05,0.15,0.2,0.3])
# Fit the train data
eclf.fit(tr_X,tr_y)

# Predict in probabilities
tr_pred = eclf.predict_proba(tr_X)
cv_pred = eclf.predict_proba(cv_X)
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```



17 5.10.1 Kaggle Score

[28]:
```
# 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'])
```

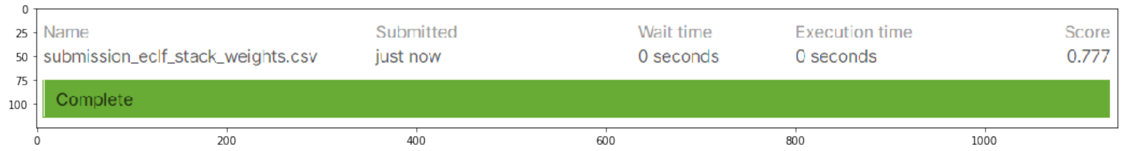
```
eclf_df['id'] = eclf_df['id'].astype('int32')
eclf_df.to_csv(data_dir+'/submission_eclf_stack_weights.csv', index=False)
```

[29]:

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

[29]:

```
<matplotlib.image.AxesImage at 0x1c72362d788>
```



Name	Submitted	Wait time	Execution time	Score
submission_eclf_stack_weights.csv	just now	0 seconds	0 seconds	0.777

Complete

18 6. Summary of all Models

[30]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = (['Model','Hyperparameter','CV score','Test score'])
x.add_row(['kNN',r"{'algorithm': 'kd_tree', 'n_neighbors': 45}",0.72,0.61])
x.add_row(['Logistic Regression',r"{'C': 1000, 'penalty': 'l1', 'solver':
↪'liblinear'}",0.64,0.747])
x.add_row(['SVC',r"{'C': 1, 'kernel': 'poly'}",0.69,0.703])
x.add_row(['RandomForest',r"{'max_depth': 5, 'n_estimators': 100}",0.74,0.736])
x.add_row(['XGBoost',r"{'max_depth': 3, 'n_estimators': 200}",0.84,0.751])
x.add_row(['Stack Classifier','-',0.73,0.777])
x.add_row(['Voting Classifier(No stacking + no weights)','-',0.76,0.777])
x.add_row(['Voting Classifier(stacking + no weights)','-',0.74,0.780])
x.add_row(['Voting Classifier(no stacking + weights)','-',0.77,0.779])
x.add_row(['Voting Classifier(stacking + weights)','-',0.72,0.777])
print(x)
```

```
+---------------------------------------------+---------------------------------
-------------------+----------+------------+
|                    Model                    |                   Hyperparameter
| CV score | Test score |
+---------------------------------------------+---------------------------------
-------------------+----------+------------+
|                     kNN                     |    {'algorithm': 'kd_tree',
'n_neighbors': 45}      |   0.72   |    0.61    |
|             Logistic Regression             | {'C': 1000, 'penalty': 'l1',
'solver':'liblinear'} |   0.64   |   0.747    |
|                     SVC                     |            {'C': 1, 'kernel':
'poly'}               |   0.69   |   0.703    |
|                 RandomForest                 |       {'max_depth': 5,
'n_estimators': 100}        |   0.74   |   0.736    |
|                   XGBoost                    |       {'max_depth': 3,
'n_estimators': 200}        |   0.84   |   0.751    |
|               Stack Classifier               |                          -
|   0.73   |   0.777    |
| Voting Classifier(No stacking + no weights) |                          -
|   0.76   |   0.777    |
|   Voting Classifier(stacking + no weights)  |                          -
|   0.74   |    0.78    |
|   Voting Classifier(no stacking + weights)  |                          -
|   0.77   |   0.779    |
|    Voting Classifier(stacking + weights)    |                          -
|   0.72   |   0.777    |
+---------------------------------------------+----------------------------------
-------------------+----------+-------------+
```

[ ]: