3_2_Models

April 20, 2020

1 SMOTE + FE + Normalization + ML Classification Model

SMOTE → Oversampling technique (called Synthetic Minority Oversampling Technique)

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 MinMaxScaler
```

```
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]:
```python
# 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]:
```python
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 Engineering

[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)
```

```python
    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]:
```python
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. Split and Oversampling data

[7]:
```
# 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)
```

6 5. Normalization

[8]:
```
# Fit and transform on train data
stand_vec = MinMaxScaler()
tr_X = stand_vec.fit_transform(tr_X)
pd.DataFrame(tr_X).head(5)
```

[8]:
```
          0         1         2         3         4         5         6    \
0  0.543185  0.385981  0.307849  0.486355  0.593561  0.548421  0.309127
1  0.536840  0.360374  0.582875  0.362963  0.454180  0.333629  0.769649
2  0.429595  0.668411  0.542508  0.541326  0.731114  0.284675  0.213147
3  0.447605  0.656262  0.623445  0.691033  0.192427  0.596488  0.614632
4  0.161891  0.607477  0.423038  0.435673  0.394366  0.523413  0.904745

          7         8         9    …       304       305       306       307  \
0  0.730249  0.761705  0.344966  …  0.460236  0.596477  0.145704  0.460236
1  0.242868  0.711077  0.369924  …  0.326472  0.512572  0.407877  0.326472
2  0.666423  0.082794  0.230753  …  0.361306  0.051273  0.713982  0.361306
3  0.600585  0.251618  0.124365  …  0.351714  0.165939  0.542006  0.351714
4  0.400146  0.468786  0.491751  …  0.335899  0.548545  0.470274  0.335899

        308       309       310       311       312       313
0  0.758810  0.758810  0.808910  0.755952  0.758091  0.488427
1  0.415875  0.415875  0.454953  0.553698  0.330533  0.472571
2  0.092408  0.092408  0.039308  0.260016  0.274790  0.212496
3  0.169851  0.169851  0.150722  0.347510  0.324467  0.158788
4  0.332553  0.332553  0.382469  0.473672  0.267125  0.345071

[5 rows x 314 columns]
```

[9]:
```
# Transform on cv data on the basis of mean and std generated from train data
cv_X = stand_vec.transform(cv_X)
pd.DataFrame(cv_X).head(5)
```

[9]:
```
          0         1         2         3         4         5         6    \
0  0.307818  0.205421  0.471764  0.318129  0.479056  0.361121  0.647591
1  0.826852  0.712150  0.203466  0.288109  0.579111  0.466300  0.687613
2  0.199959  0.533832  0.425688  0.530604  0.643314  0.238560  0.845346
3  0.029267  0.095514  0.616514  0.330994  0.263947  0.682866  0.593263
4  0.468072  0.616449  0.920082  0.324951  0.685019  0.062433  0.472836

          7         8         9    …       304       305       306       307  \
0  0.461960  0.555577  0.377538  …  0.364418  0.367169  0.413726  0.364418
1  0.480432  0.245527  0.666244  …  0.036243  0.691210  0.344602  0.036243
2  0.520666  1.004949  0.696277  … -0.026075  0.453845  0.639559 -0.026075
3  0.445318  0.485535  0.665398  …  0.337094  0.309429  0.414117  0.337094
4  0.399232  0.474496  0.175127  …  0.252625  0.674429  0.657089  0.252625
```

```
        308       309       310       311       312       313
0  0.313638  0.313638  0.314852  0.522585  0.285660  0.361049
1  0.602803  0.602803  0.665598  0.606822  0.517266  0.476554
2  0.261661  0.261661  0.296326  0.290178  0.354881  0.184931
3  0.392901  0.392901  0.365331  0.565013  0.369783  0.471386
4  0.375732  0.375732  0.443103  0.334018  0.364260  0.308094

[5 rows x 314 columns]
```

[10]:
```
# Transform on test data on the basis of mean and std generated from train data
ts_X = stand_vec.transform(ts_X)
pd.DataFrame(ts_X).head(5)
```

[10]:
```
          0         1         2         3         4         5         6    \
0  0.576955  0.354766  0.179817  0.520078  0.338760  0.593650  0.635820
1  0.633442  0.718692  0.404281  0.722417  0.310774  0.589748  0.643607
2  0.832788  0.642991  0.493374  0.622612  0.382294  0.757893  0.411988
3  0.360827  0.201121  0.365953  0.572515  0.760563  0.595424  0.281963
4  0.628940  0.502056  0.744139  0.143470  0.471008  0.570238  0.680369

          7         8         9    …       304       305       306       307  \
0  0.455925  0.396079  0.772420  …  0.355177  0.897632  0.535569  0.355177
1  0.422275  0.565474  0.786168  …  0.310915  0.376359  0.277277  0.310915
2  0.112107  0.264560  0.534687  …  0.333485  0.403085  0.546558  0.333485
3  0.611558  0.624096  0.619289  …  0.327472  0.473930  0.543680  0.327472
4  0.157096  0.254282  0.302665  …  0.312815  0.772634  0.552227  0.312815

        308       309       310       311       312       313
0  0.777307  0.777307  0.870016  0.418998  0.854945  0.336299
1  0.414707  0.414707  0.422914  0.645653  0.356126  0.446974
2  0.242601  0.242601  0.258222  0.436530  0.189445  0.381293
3  0.310356  0.310356  0.351761  0.356717  0.398786  0.176759
4  0.700724  0.700724  0.775491  0.424850  0.767844  0.386215

[5 rows x 314 columns]
```

7 6. Apply ML Models (with hyperparameter)

[11]:
```
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
```

```python
    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

def position_featengg(df):
    '''
    Print the position of feature engg after model fitted

    Parameter:
    df: Pass Dataframe that contain Feaeture name and their coefficient

    Return:
    Print the rank of the feature engg only!
    '''
    list_feat_engg =␣
 ↪['mean','std','mean_sin','mean_cos','mean_tan','mean_sinh','mean_cosh','mean_tanh','mean_ex
                    'mean_expm1','mean_exp2','mean_x2','mean_x3','mean_x4']

    for i in list_feat_engg:
        print('Position rank of',i,':',df[df['col_name']==i].index[0])
```

7.1 6.1 kNN

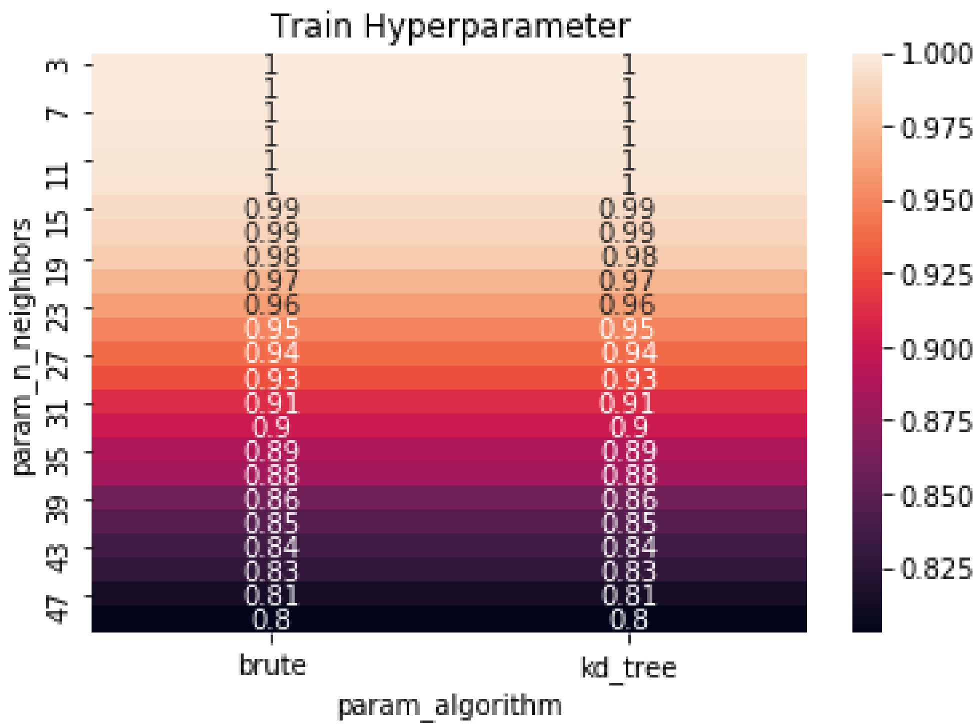
[14]:
```
# Import KNN
from sklearn.neighbors import KNeighborsClassifier
```

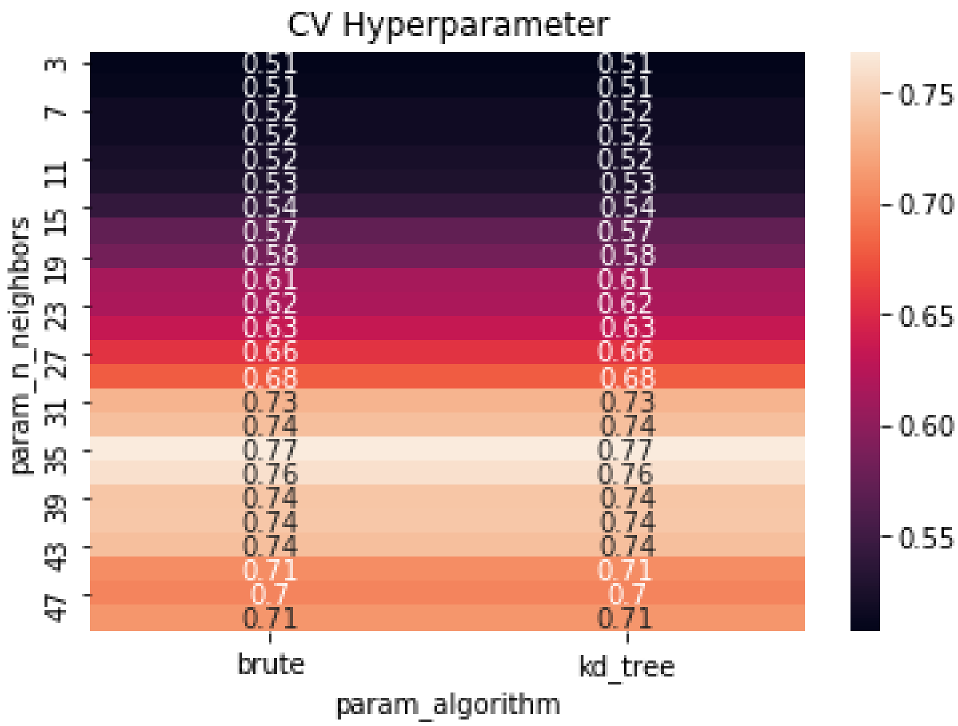
[34]:
```
# kNN (See Docs: https://scikit-learn.org/stable/modules/generated/sklearn.
 ↪neighbors.KNeighborsClassifier.html)

# 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)
```

[35]:
```
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')
```

[36]:
```
plt.title('Train Hyperparameter')
sns.heatmap(tr_pvt, annot=True)
plt.show()
```

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

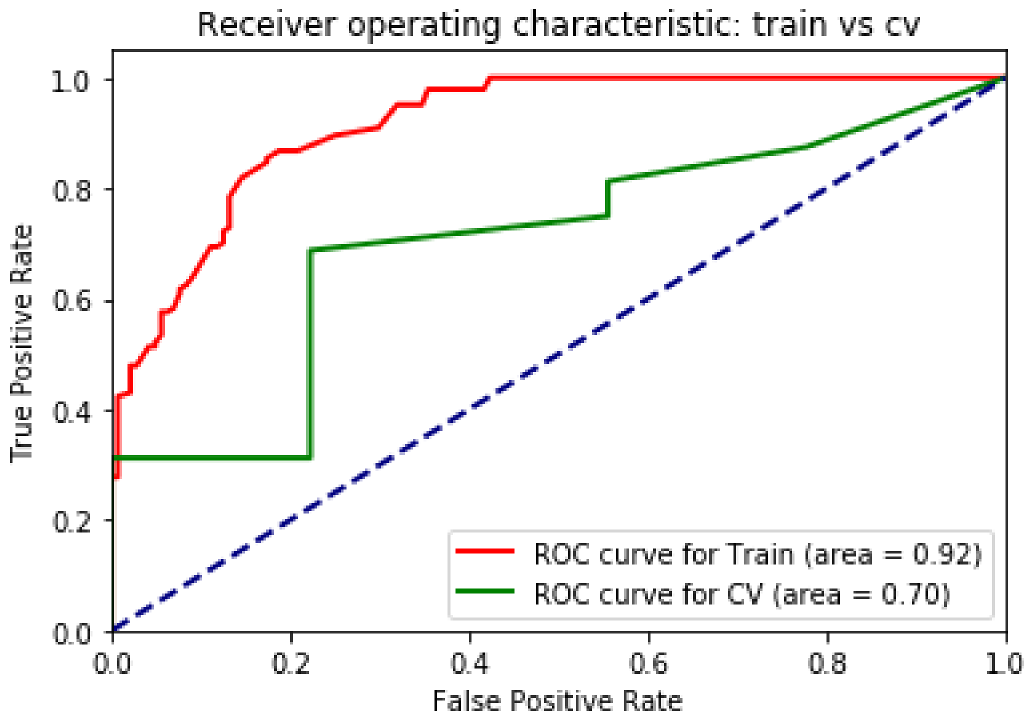




[37]:
```
print(knn_clf.best_params_)
print('CV Score',knn_clf.score(cv_X,cv_y))
```

```
{'algorithm': 'kd_tree', 'n_neighbors': 35}
CV Score 0.7291666666666667
```

[38]:
```
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)
```

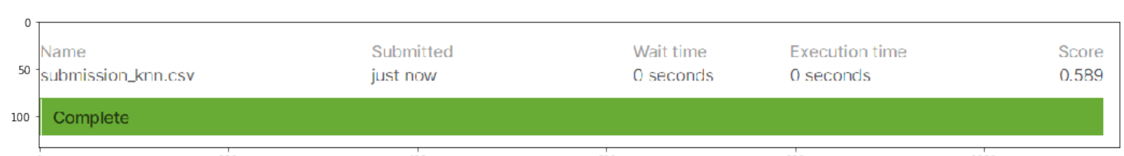


8 6.1.1 Kaggle Score

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

[40]:
```python
image = plt.imread(data_dir+'/submission_knn.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[40]: <matplotlib.image.AxesImage at 0x1fd6188b6c8>



Observation: Knn perform kind of poorly. So, Knn will not work got this kind of problem

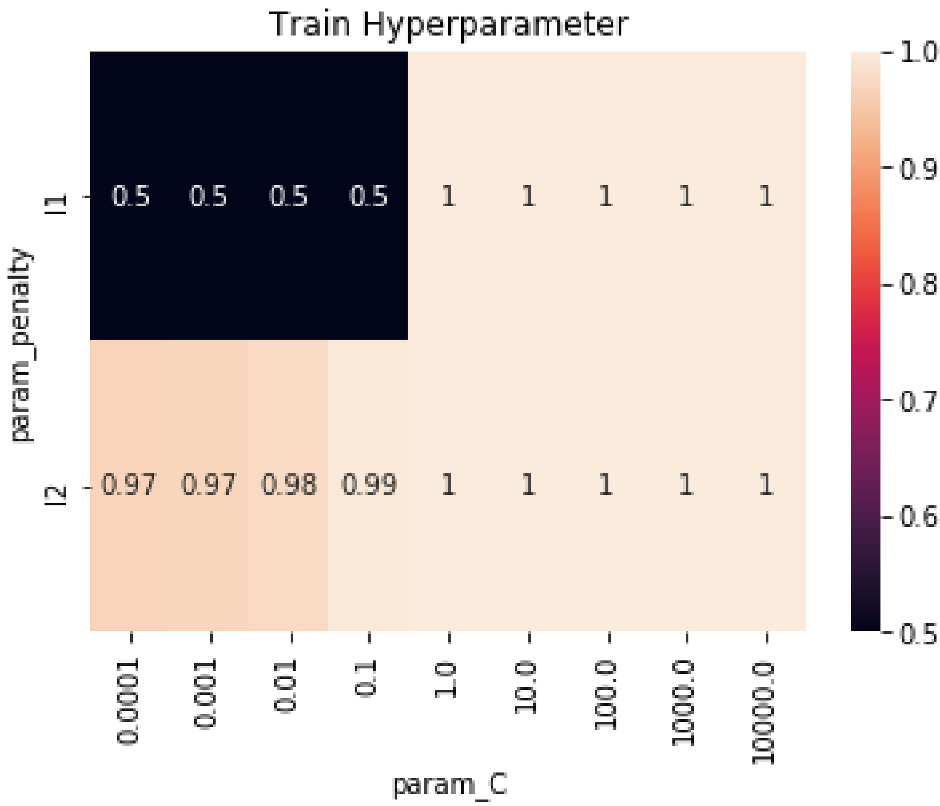
8.1 6.2 Logistic Regression

[12]:
```
# Import Logistic Regression
from sklearn.linear_model import LogisticRegression
```

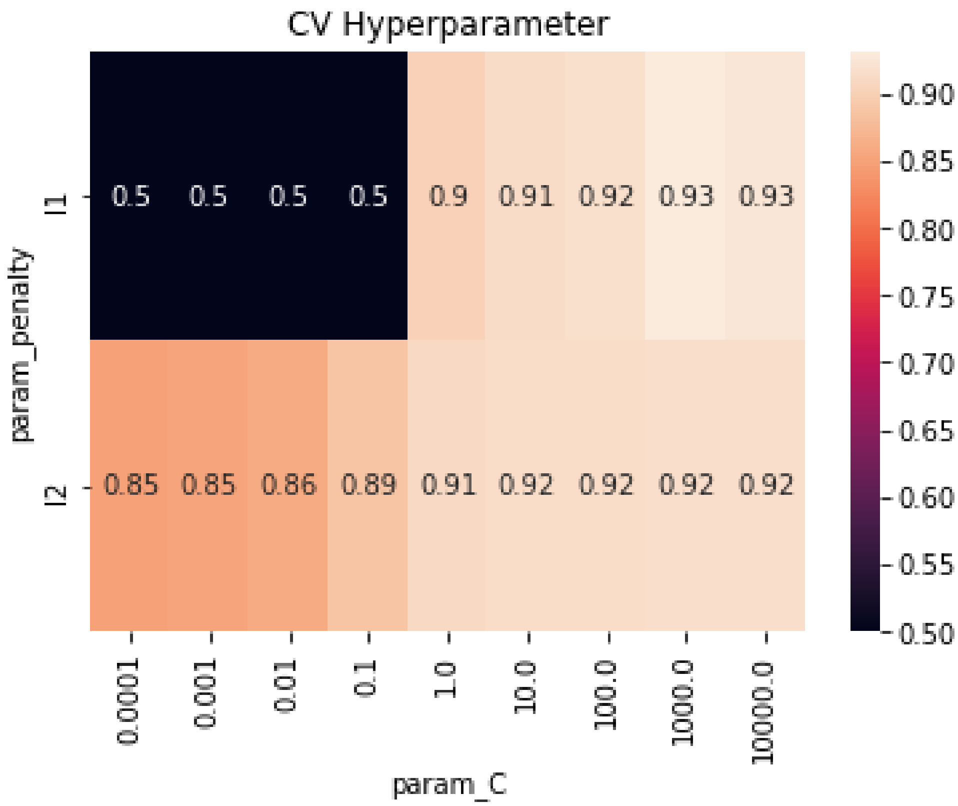
[42]:
```
# LogisticRegression (See Docs: https://scikit-learn.org/stable/modules/
↪generated/sklearn.linear_model.LogisticRegression.html)

# 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)
```

[43]:
```
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')
```

[44]:
```
plt.title('Train Hyperparameter')
sns.heatmap(tr_pvt, annot=True)
plt.show()

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



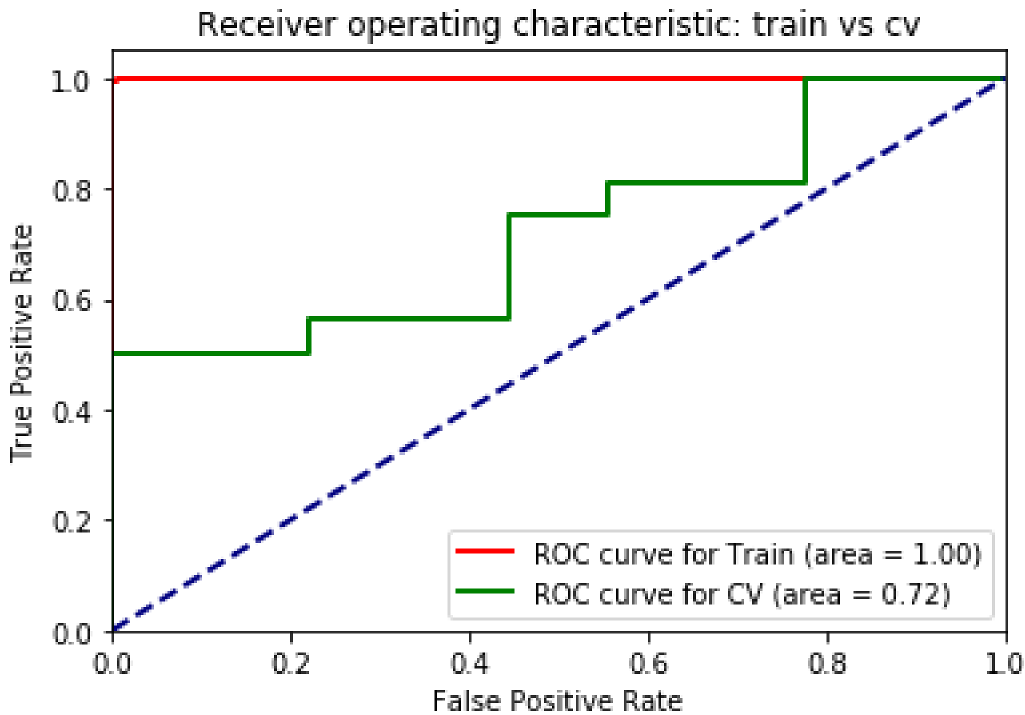
Train Hyperparameter
param_penalty
l1
l2
0.5
0.5
0.5
0.5
1
1
1
1
1
0.97
0.97
0.98
0.99
1
1
1
1
1
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
param_C
1.0
0.9
0.8
0.7
0.6
0.5



[45]:
```
print(log_clf.best_params_)
print('cv Score',log_clf.score(cv_X,cv_y))
```

```
{'C': 1000, 'penalty': 'l1', 'solver': 'liblinear'}
cv Score 0.625
```

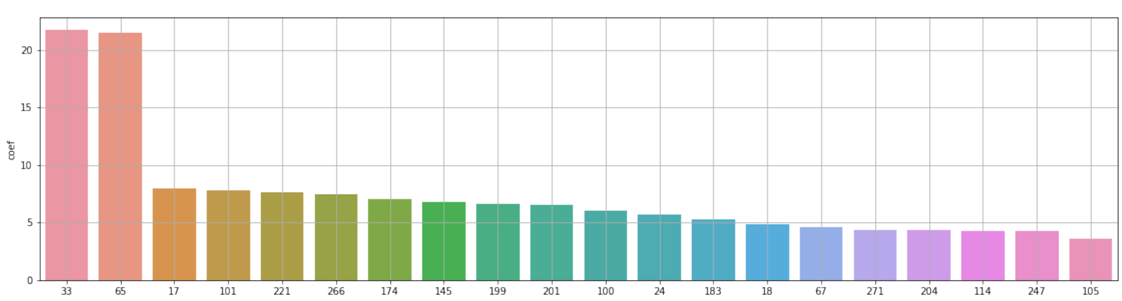
[46]:
```
clf = CalibratedClassifierCV(log_clf, cv=3)
clf.fit(tr_X,tr_y)

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)
```



[47]:
```
# Instance the model passing the best params we got
log_model = LogisticRegression(**log_clf.best_params_, random_state=42, class_weight='balanced')
log_model.fit(tr_X, tr_y)
```

[47]:
```
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)
```

[48]:
```
# Plot the feature importance on the basis of logistic Regression
df = plot_feature_importance(log_model, 'log_model', 20)
```



[49]:
```
print('After applying Logistic regression\n')
position_featengg(df)
```

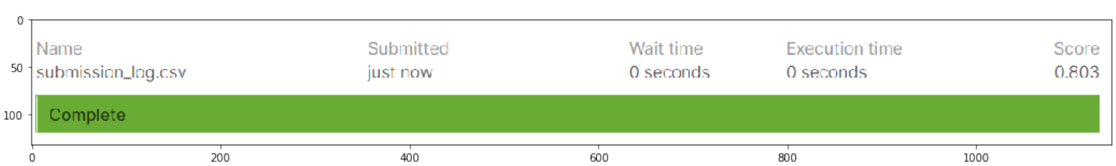
```
After applying Logistic regression

Position rank of mean : 284
Position rank of std : 120
Position rank of mean_sin : 250
Position rank of mean_cos : 121
Position rank of mean_tan : 122
Position rank of mean_sinh : 259
Position rank of mean_cosh : 123
Position rank of mean_tanh : 124
Position rank of mean_exp : 125
Position rank of mean_expm1 : 126
Position rank of mean_exp2 : 127
Position rank of mean_x2 : 128
Position rank of mean_x3 : 245
Position rank of mean_x4 : 156
```

8.2 6.2.1 Kaggle Score

[50]:
```
# 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)
```

[51]:
```
image = plt.imread(data_dir+'/submission_log.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[51]: <matplotlib.image.AxesImage at 0x1fd554e7a08>

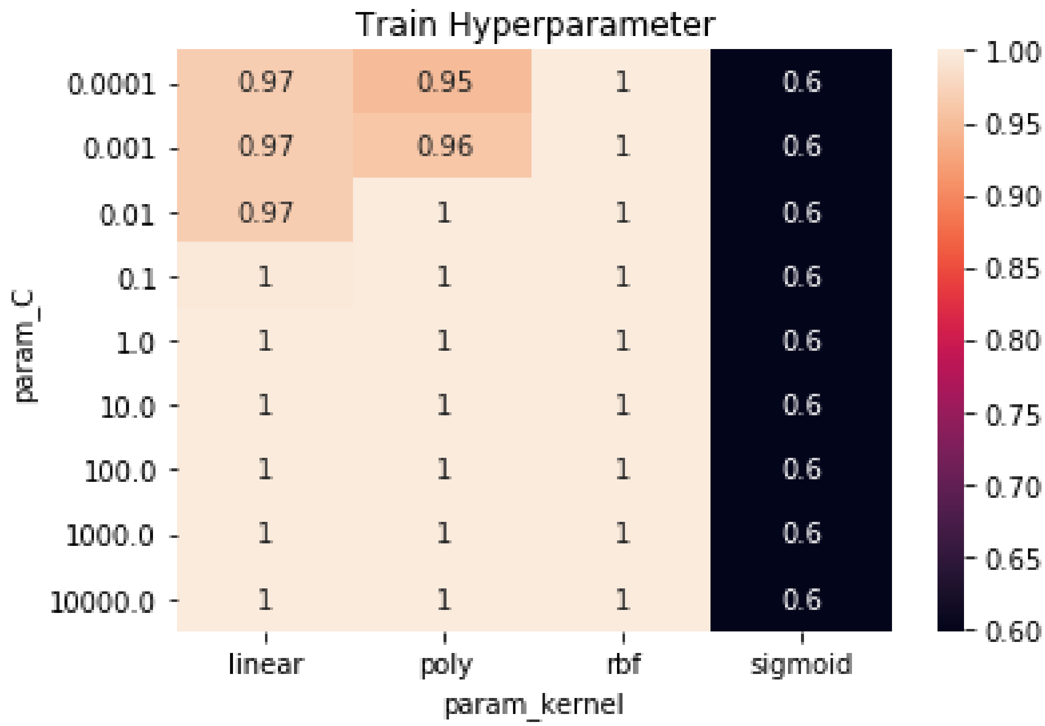


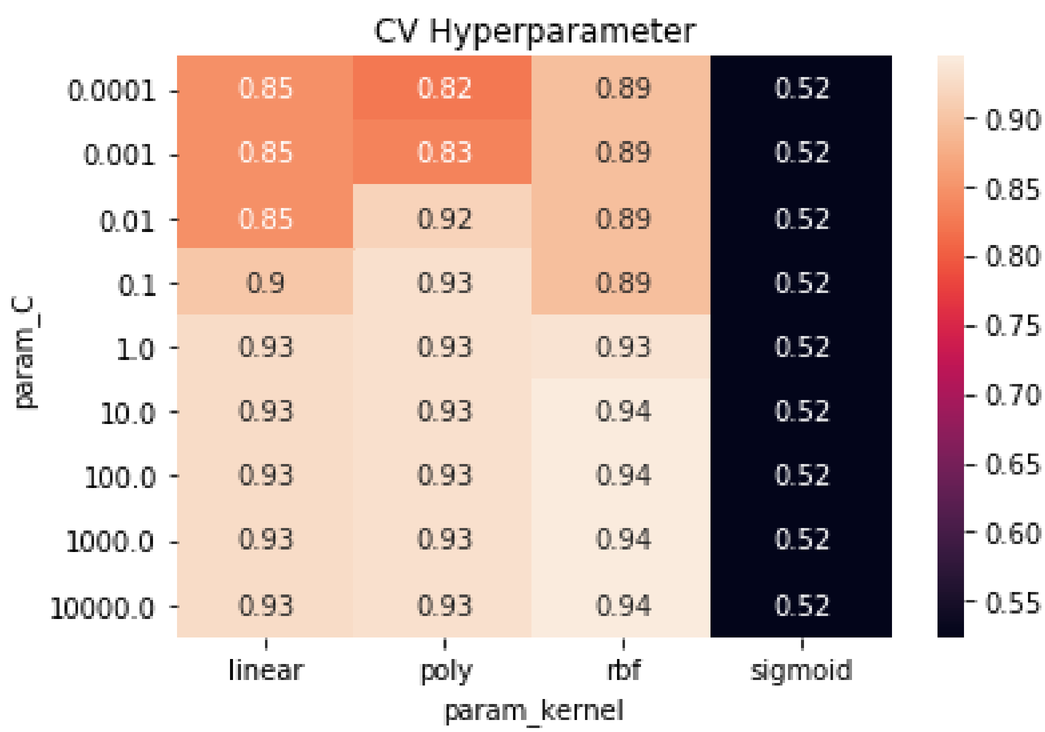
8.3 6.3 SVC

[13]:
```
# Import SVC
from sklearn.svm import SVC
```

[53]:
```
# 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)
```

[54]:
```
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()
```

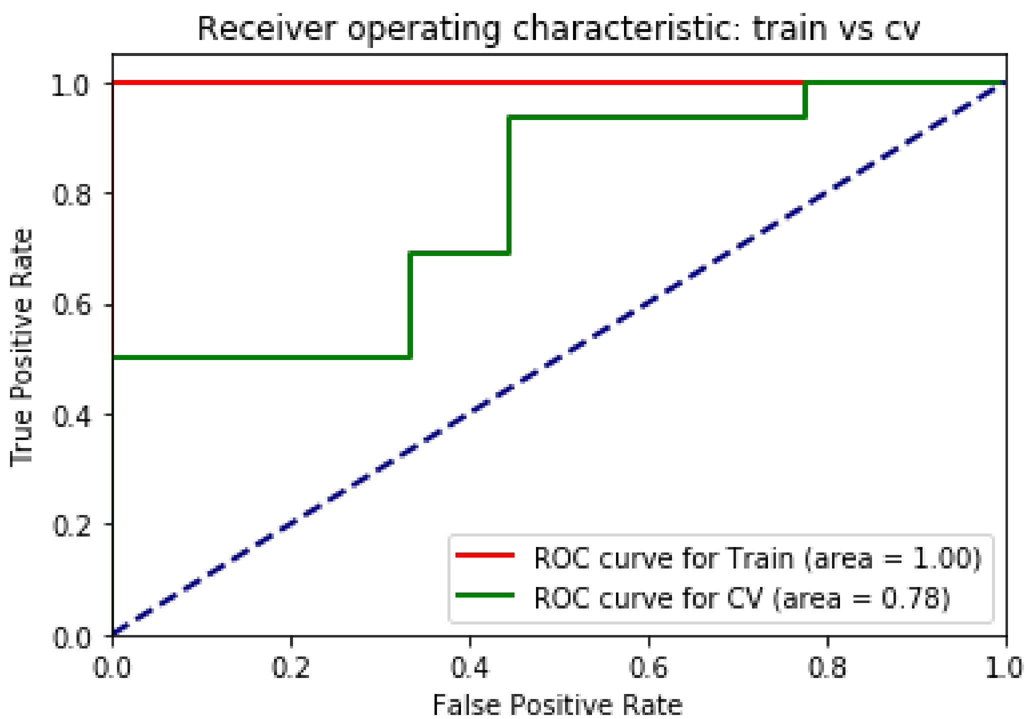

Train Hyperparameter
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
param_kernel
linear
poly
rbf
sigmoid
0.97 0.95 1 0.6
0.97 0.96 1 0.6
0.97 1 1 0.6
1 1 1 0.6
1 1 1 0.6
1 1 1 0.6
1 1 1 0.6
1 1 1 0.6
1 1 1 0.6
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60


CV Hyperparameter
param_C
0.0001
0.001
0.01
0.1
1.0
10.0
100.0
1000.0
10000.0
param_kernel
linear
poly
rbf
sigmoid
0.85 0.82 0.89 0.52
0.85 0.83 0.89 0.52
0.85 0.92 0.89 0.52
0.9 0.93 0.89 0.52
0.93 0.93 0.93 0.52
0.93 0.93 0.94 0.52
0.93 0.93 0.94 0.52
0.93 0.93 0.94 0.52
0.93 0.93 0.94 0.52
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55

[56]:
```
print(svc_clf.best_params_)
print('cv Score',svc_clf.score(cv_X,cv_y))
```

```
{'C': 10, 'kernel': 'rbf'}
cv Score 0.6597222222222222
```

[57]:
```
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)
```



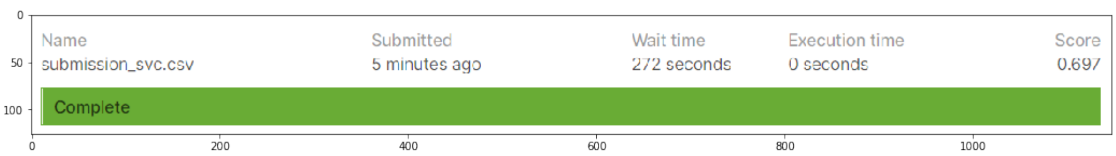
8.4 6.3.1 Kaggle Score

[58]:
```
# 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)
```

[76]:
```
image = plt.imread(data_dir+'/submission_svc.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[76]: <matplotlib.image.AxesImage at 0x1fd55683bc8>



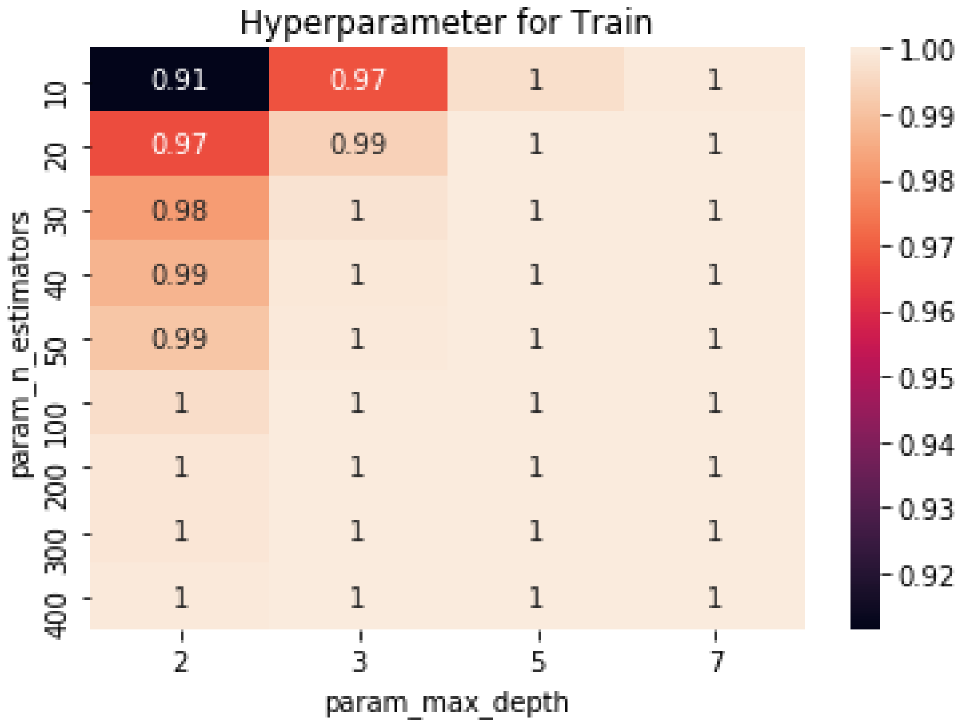
8.5 6.4 Random Forest

[14]:
```
# Impoer Random Forest
from sklearn.ensemble import RandomForestClassifier
```

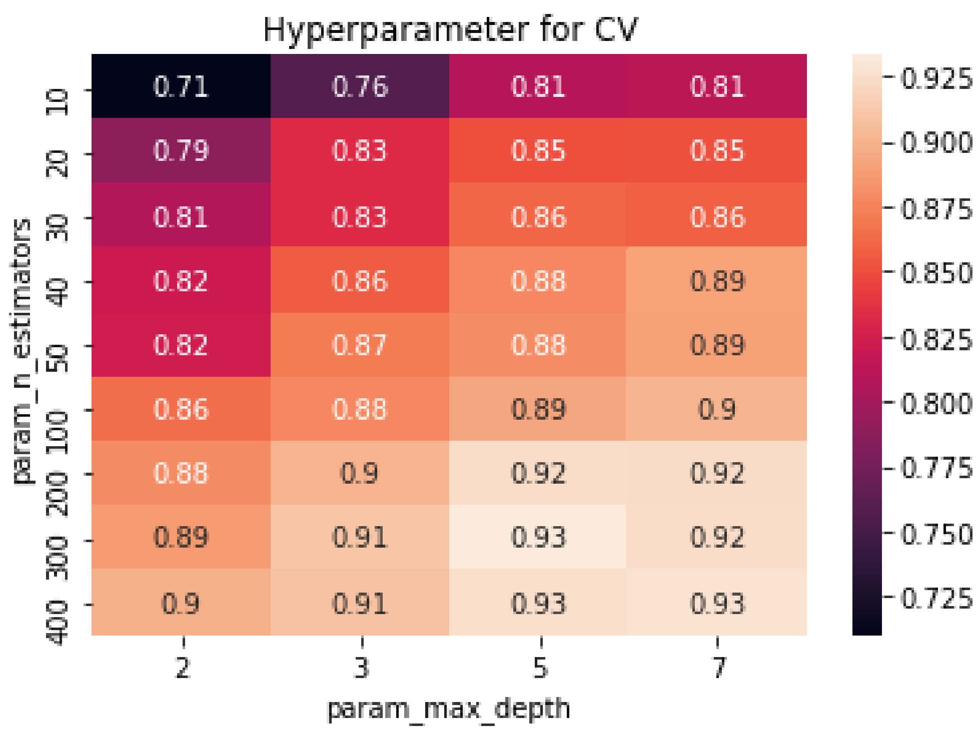
[60]:
```
# RandomForest (See Docs: https://scikit-learn.org/stable/modules/generated/
↪sklearn.ensemble.RandomForestClassifier.html)

# 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)
```

[61]:
```
# 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()
```



Hyperparameter for Train
param_n_estimators
10
20
30
40
50
100
200
300
400
param_max_depth
2
3
5
7
0.91 0.97 1 1
0.97 0.99 1 1
0.98 1 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
1.00
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
param_max_depth
2
3
5
7
0.71 0.76 0.81 0.81
0.79 0.83 0.85 0.85
0.81 0.83 0.86 0.86
0.82 0.86 0.88 0.89
0.82 0.87 0.88 0.89
0.86 0.88 0.89 0.9
0.88 0.9 0.92 0.92
0.89 0.91 0.93 0.92
0.9 0.91 0.93 0.93
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0.725

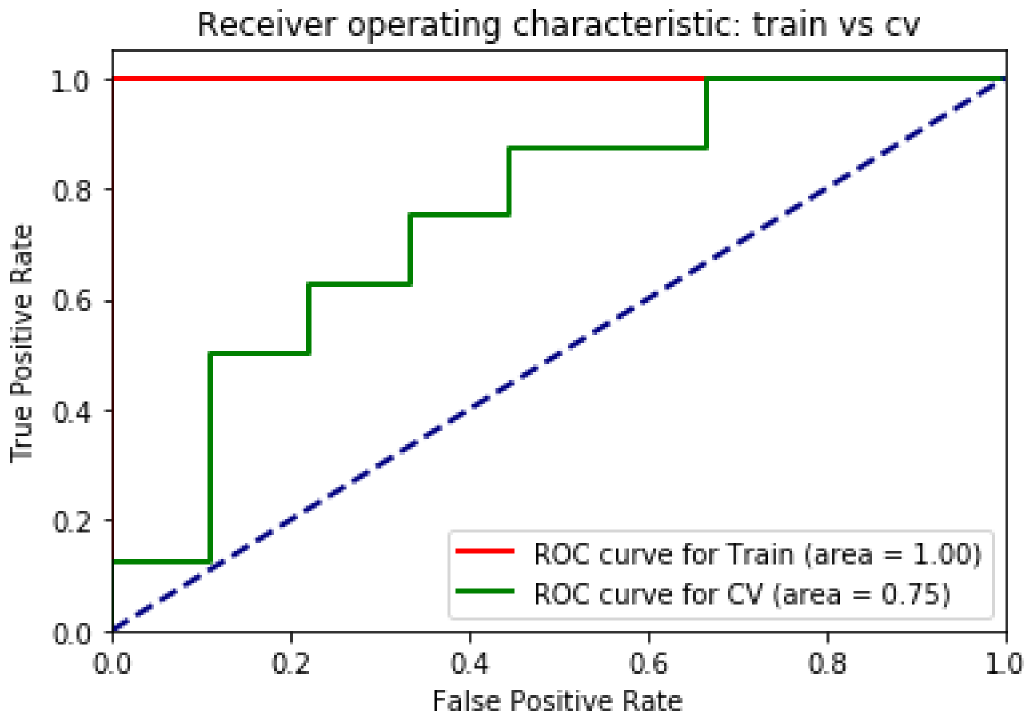
[62]:
```
print(rf_clf.best_params_)
```

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

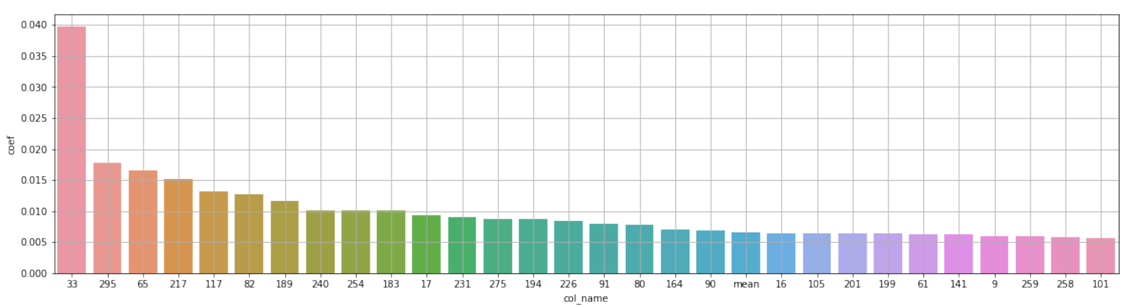
[63]:
```
rf_clf = RandomForestClassifier(**rf_clf.best_params_, random_state=42)
rf_clf.fit(tr_X,tr_y)

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

[63]:
```
CalibratedClassifierCV(base_estimator=RandomForestClassifier(bootstrap=True,
                                                             ccp_alpha=0.0,
                                                             class_weight=None,
                                                             criterion='gini',
                                                             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=300,
                                                             n_jobs=None,
                                                             oob_score=False,
                                                             random_state=42,
                                                             verbose=0,
                                                             warm_start=False),
                       cv=3, method='sigmoid')
```

[64]:
```
# Plot ROC Curve of train and cv
plot_roc(tr_y, clf.predict_proba(tr_X), cv_y, clf.predict_proba(cv_X), 2)
```




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




```
[66]: print('After applying Random Forest\n')
position_featengg(df)
```

```
After applying Random Forest

Position rank of mean : 19
Position rank of std : 81
Position rank of mean_sin : 40
Position rank of mean_cos : 56
```

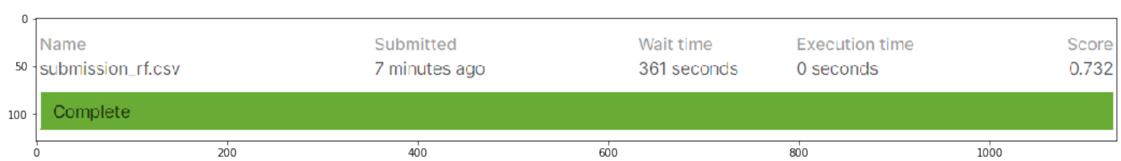
```
Position rank of mean_tan : 257
Position rank of mean_sinh : 52
Position rank of mean_cosh : 57
Position rank of mean_tanh : 181
Position rank of mean_exp : 208
Position rank of mean_expm1 : 217
Position rank of mean_exp2 : 73
Position rank of mean_x2 : 172
Position rank of mean_x3 : 108
Position rank of mean_x4 : 313
```

8.6 6.4.1 Kaggle Score

[67]:
```python
# Create a submission file format to submit in kaggle
temp_id = df_test['id']
rf_csv = clf.predict_proba(ts_X)[:,1]
rf_df = pd.DataFrame(np.column_stack((temp_id,rf_csv)), columns=['id','target'])
rf_df['id'] = rf_df['id'].astype('int32')
rf_df.to_csv(data_dir+'/submission_rf.csv', index=False)
```

[82]:
```python
image = plt.imread(data_dir+'/submission_rf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[82]:
```
<matplotlib.image.AxesImage at 0x1fd56b90a88>
```



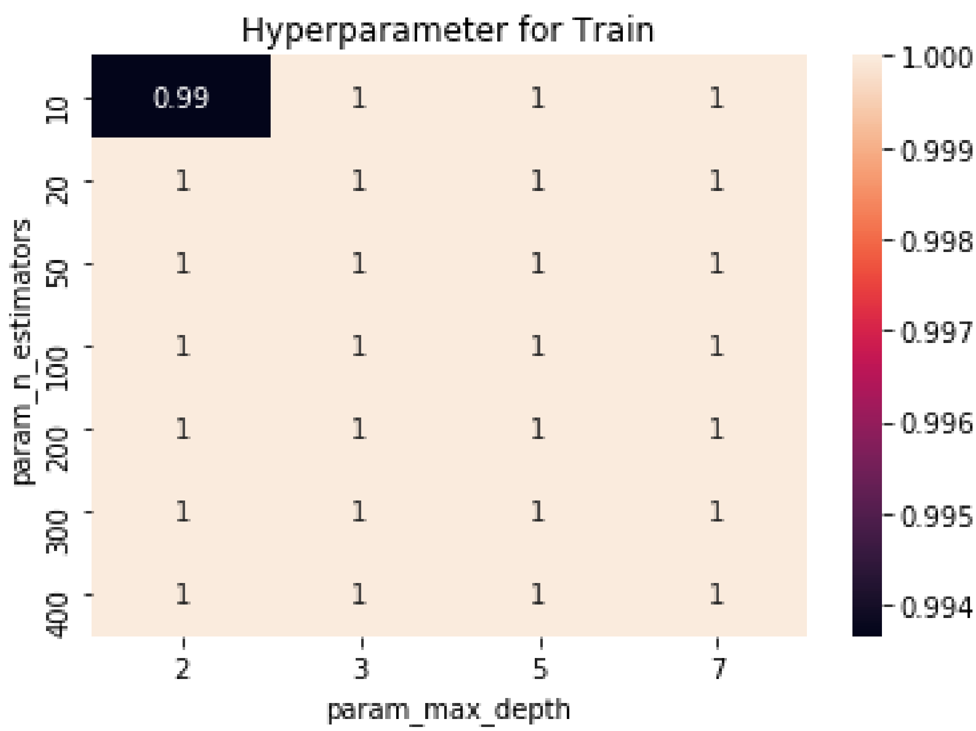
8.7 6.5 Xgboost

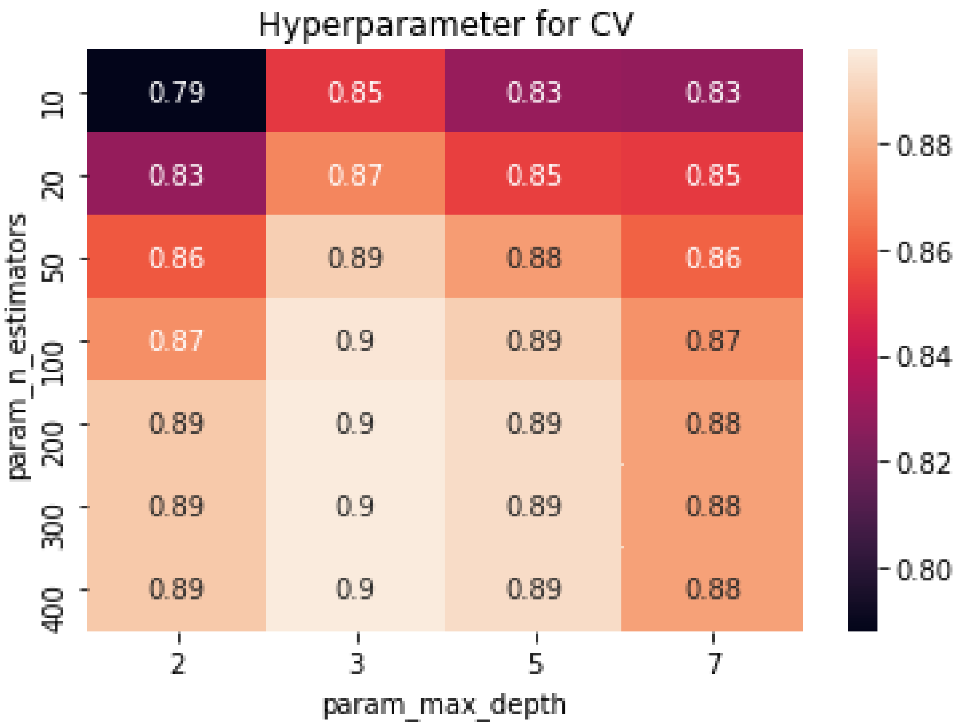
[15]:
```python
# Import Xgboost
from xgboost import XGBClassifier
```

[78]:
```python
# 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)
```

[79]:
```
# Ref: https://stackoverflow.com/questions/48791709/
↪how-to-plot-a-heat-map-on-pivot-table-after-grid-search

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

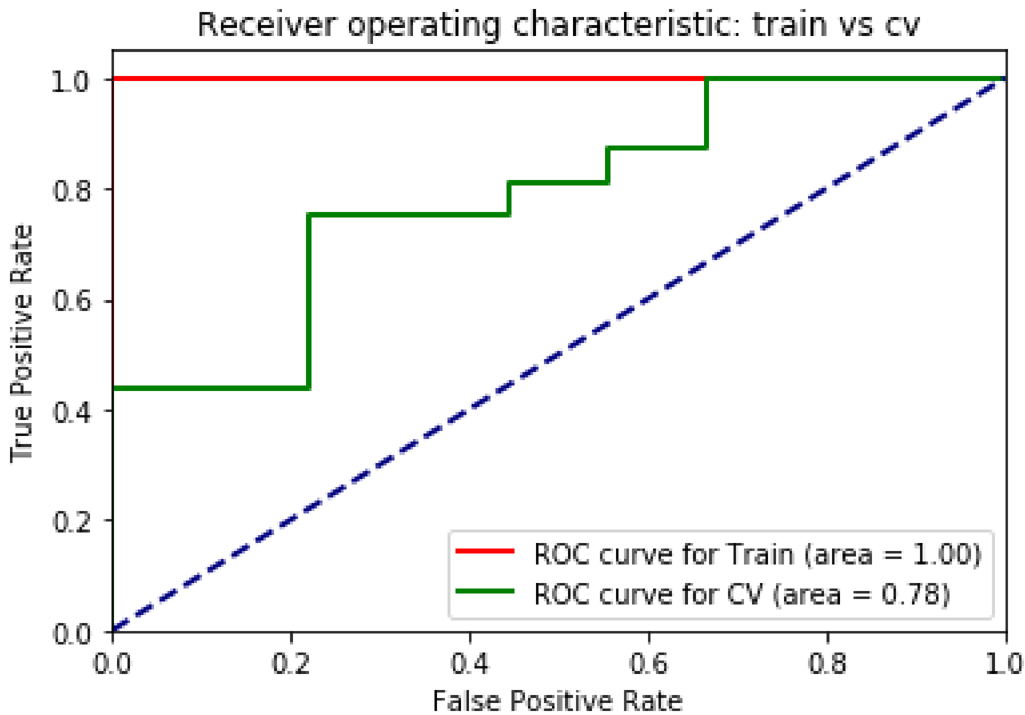




[80]:
```
print(xgb_clf.best_params_)
print('cv Score',xgb_clf.score(cv_X,cv_y))
```

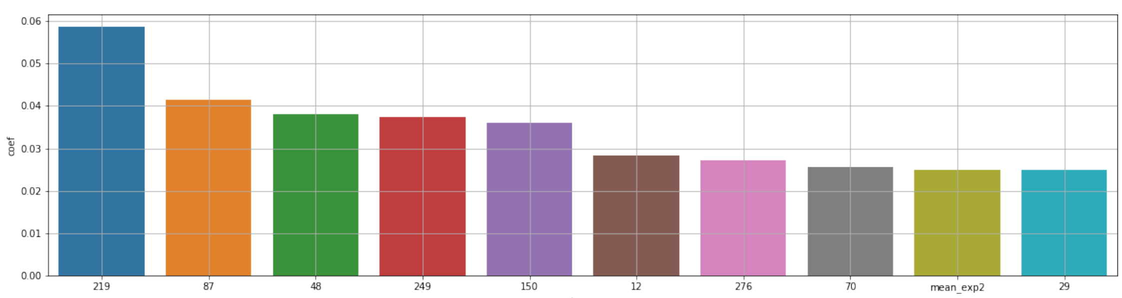
```
{'max_depth': 3, 'n_estimators': 200}
cv Score 0.8125
```

[84]:
```
# 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)
```



[85]:

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



[86]:

```python
print('After applying Gradient Boosting Random Forest\n')
position_featengg(df)
```

```
After applying Gradient Boosting Random Forest

Position rank of mean : 67
Position rank of std : 160
Position rank of mean_sin : 161
Position rank of mean_cos : 162
```

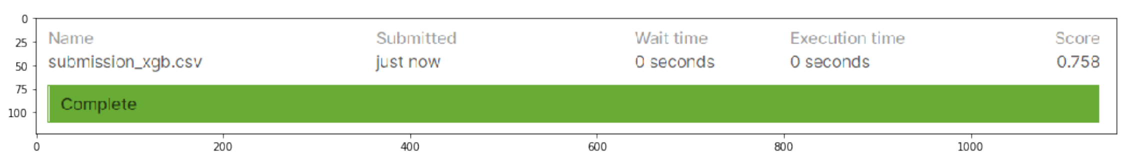
```
Position rank of mean_tan : 29
Position rank of mean_sinh : 163
Position rank of mean_cosh : 164
Position rank of mean_tanh : 165
Position rank of mean_exp : 166
Position rank of mean_expm1 : 167
Position rank of mean_exp2 : 8
Position rank of mean_x2 : 168
Position rank of mean_x3 : 84
Position rank of mean_x4 : 313
```

8.8 6.5.1 Kaggle Score

[87]:
```python
# Create submission file format to submit in Kaggle
temp_id = df_test['id']
xgb_csv = clf.predict_proba(ts_X)[:,1]
xgb_df = pd.DataFrame(np.column_stack((temp_id,xgb_csv)),␣
 ↪columns=['id','target'])
xgb_df['id'] = xgb_df['id'].astype('int32')
xgb_df.to_csv(data_dir+'/submission_xgb.csv', index=False)
```

[88]:
```python
image = plt.imread(data_dir+'/submission_xgb.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[88]:
```
<matplotlib.image.AxesImage at 0x1fd571ee148>
```



8.9 6.6 Stacking Model

[16]:
```python
# Import Stacking Classifier
from mlxtend.classifier import StackingClassifier
```

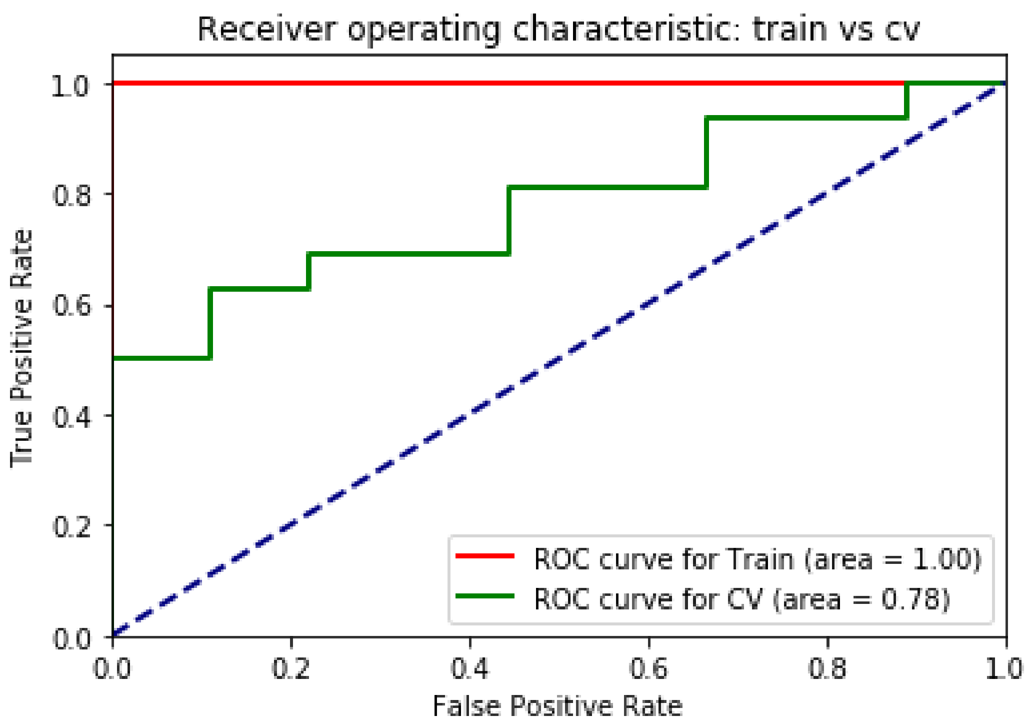
[17]:
```python
# StackClassifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
 ↪classifier/StackingClassifier/#methods)

# 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=10, kernel='rbf', random_state=42, class_weight='balanced',␣
↪probability=True)
clf2.fit(tr_X,tr_y)
clf2 = CalibratedClassifierCV(clf2, cv=3)

# Classifier 3: XGBoost with best params
clf3 = XGBClassifier(max_depth=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)
```

[91]:
```
# Score after stacking classifier
sclf.score(cv_X, cv_y)
```

[91]: 0.68

[92]:
```
# Plot ROC Curve for train and cv
plot_roc(tr_y, tr_pred, cv_y, cv_pred,2)
```

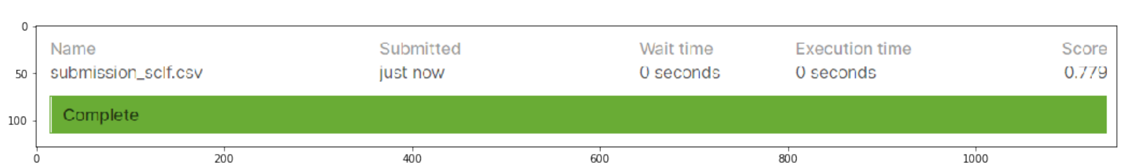


8.10 6.6.1 Kaggle Score

[93]:
```
# Create a submission file format to submit in Kaggle
temp_id = df_test['id']
sclf_csv = sclf.predict_proba(ts_X)[:,1]
sclf_df = pd.DataFrame(np.column_stack((temp_id,sclf_csv)),␣
↪columns=['id','target'])
sclf_df['id'] = sclf_df['id'].astype('int32')
sclf_df.to_csv(data_dir+'/submission_sclf.csv', index=False)
```

[94]:
```
image = plt.imread(data_dir+'/submission_sclf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[94]: <matplotlib.image.AxesImage at 0x1fd57201988>



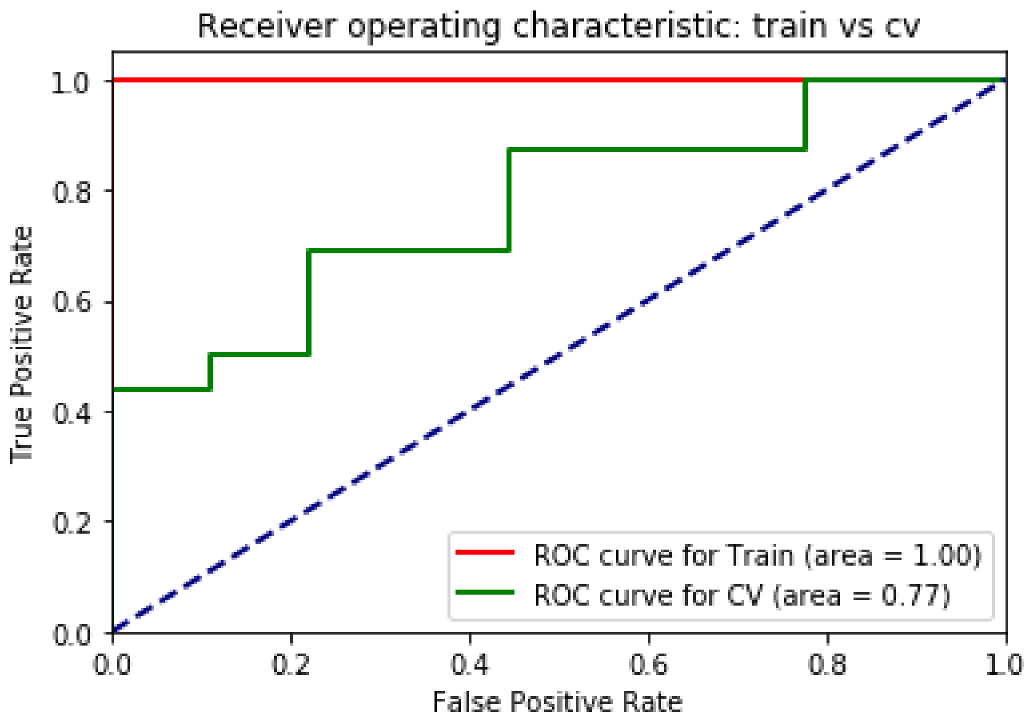
9 6.7 Voting Classifier (Without Stack Classifier + no weights)

[18]:
```
# Import Voting Classifier
from mlxtend.classifier import EnsembleVoteClassifier
```

[19]:
```
# 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)
```

[19]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(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),
                                                     cv=3, method='sigmoid'),
                           CalibratedClass…
          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=300,
          n_jobs=None,
          oob_score=False,
          random_state=None,
          verbose=0,
          warm_start=False),
                                                     cv=3, method='sigmoid')],
                       refit=True, verbose=0, voting='hard', weights=None)
```

[20]:
```
# 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)
```

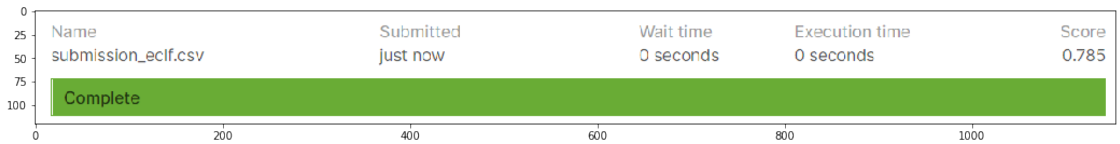


10 6.7.1 Kaggle Score

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

[22]:
```
image = plt.imread(data_dir+'/submission_eclf.png')
plt.figure(figsize=(18,5))
plt.imshow(image)
```

[22]:
```
<matplotlib.image.AxesImage at 0x20b16277d48>
```



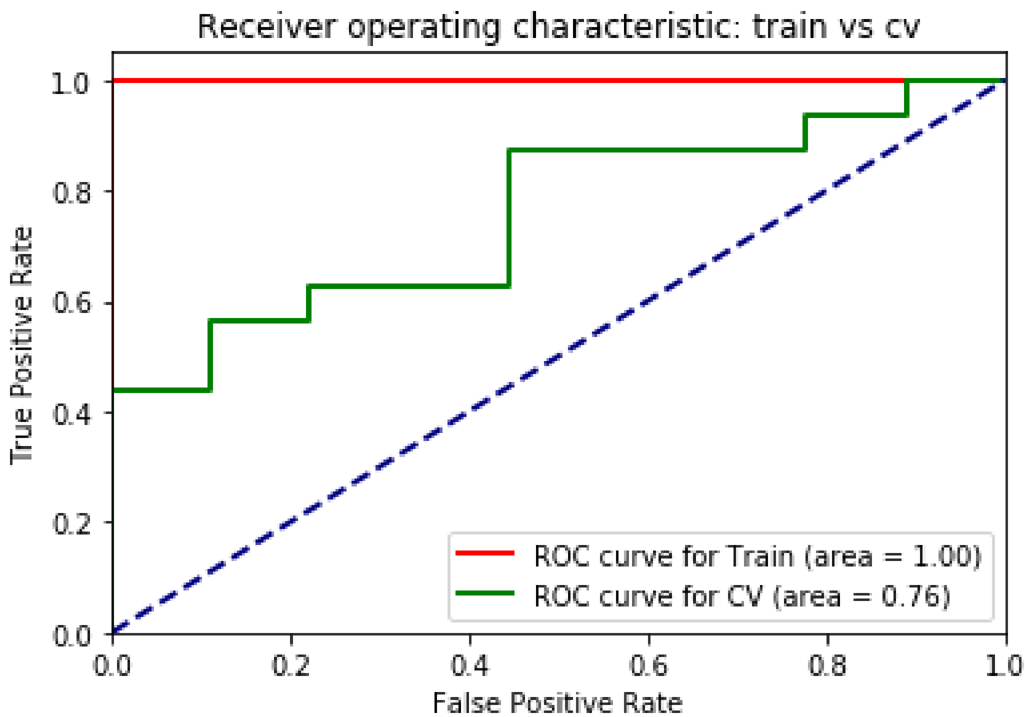
11 6.8 Voting Classifier (With Stack Classifier + no weights)

[23]:
```
# 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)
```

[23]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(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),
                                              cv=3, method='sigmoid'),
                               CalibratedClass…
                                          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)
```

[24]:
```
# 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)
```

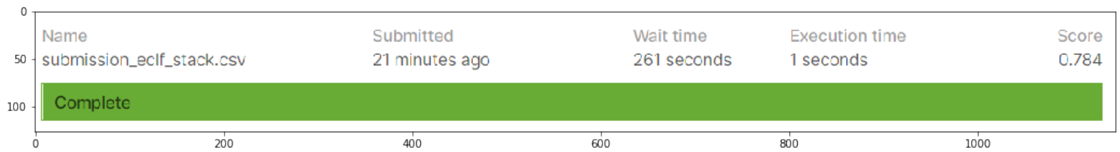


12 6.8.1 Kaggle Score

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

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

[26]:
```
<matplotlib.image.AxesImage at 0x20b29c1be88>
```



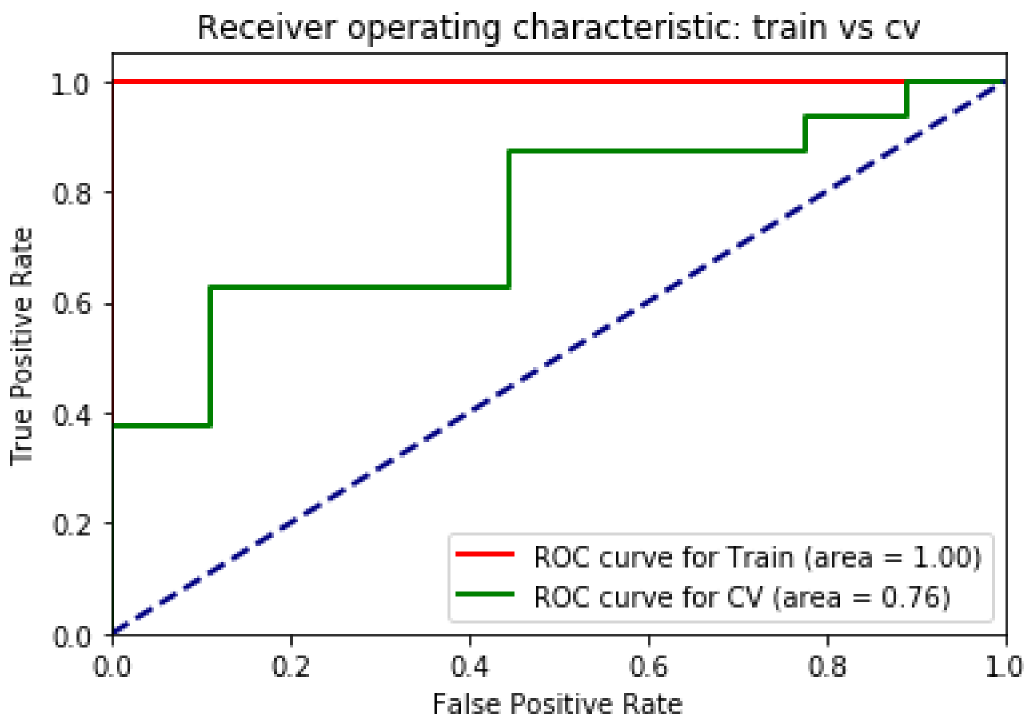
13 6.9 Voting Classifier (without Stack Classifier + weights)

[27]:
```
# Voting Classifier (See Docs: http://rasbt.github.io/mlxtend/user_guide/
↪classifier/EnsembleVoteClassifier/)
eclf = EnsembleVoteClassifier(clfs=[clf1,clf2,clf3,clf4], weights=[0.3,0.2,0.
↪2,0.3])
# Fit the train data
eclf.fit(tr_X,tr_y)
```

[27]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(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),
                                                cv=3, method='sigmoid'),
                             CalibratedClass…
          max_features='auto',
          max_leaf_nodes=None,
          max_samples=None,
          min_impurity_decrease=0.0,
          min_impurity_split=None,
```

```
                min_samples_leaf=1,
                min_samples_split=2,
                min_weight_fraction_leaf=0.0,
                n_estimators=300,
                n_jobs=None,
                oob_score=False,
                random_state=None,
                verbose=0,
                warm_start=False),
                                                  cv=3, method='sigmoid')],
                         refit=True, verbose=0, voting='hard',
                         weights=[0.3, 0.2, 0.2, 0.3])
```

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

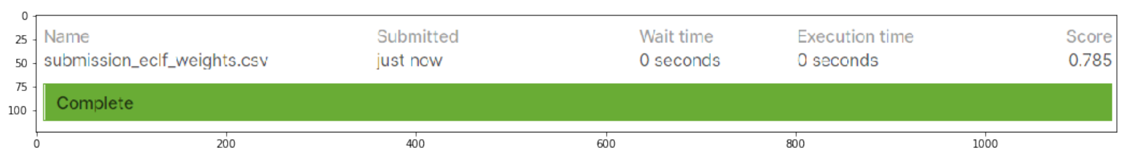


14 6.9.1 Kaggle Score

[29]:
```
# 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)
```

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

[30]:
```
<matplotlib.image.AxesImage at 0x20b29ebb348>
```



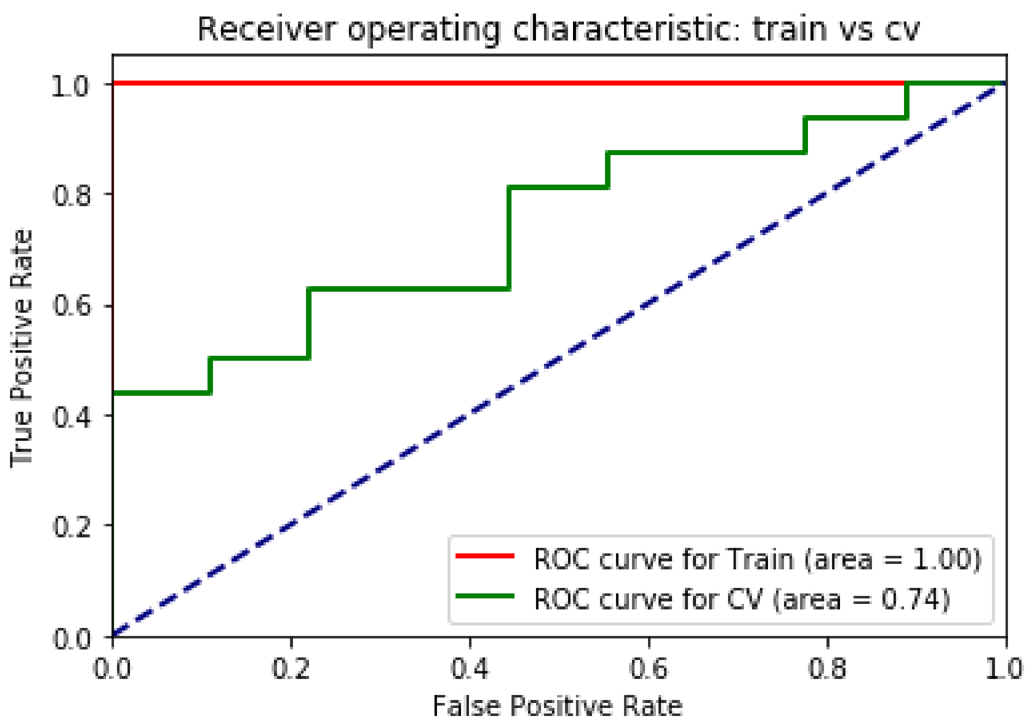
15 6.10 Voting Classifier (with Stack Classifier + weights)

[31]:
```
# 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.
↪1,0.15,0.15,0.3])
# Fit the train data
eclf.fit(tr_X,tr_y)
```

[31]:
```
EnsembleVoteClassifier(clfs=[CalibratedClassifierCV(base_estimator=LogisticRegre
ssion(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),
                                                       cv=3, method='sigmoid'),
                             CalibratedClass…
                                         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, 0.1, 0.15, 0.15, 0.3])
```

[32]:
```
# 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)
```



16 6.10.1 Kaggle Score

[33]:
```
# 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)
```

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

[34]: <matplotlib.image.AxesImage at 0x20b2a0ab348>

17 7. Summary of all Models

[35]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ['Model','Hyerparameter','cv','test']
x.add_row(['kNN',r"{'algorithm': 'kd_tree', 'n_neighbors': 35}",0.70,0.589])
x.add_row(['Logistic Regression',r"{'C': 1000, 'penalty': 'l1', 'solver':␣
↪'liblinear'}",0.72,0.803])
x.add_row(['SVC',r"{'C': 10, 'kernel': 'rbf'}",0.78,0.697])
x.add_row(['RandomForest',r"{'max_depth': 5, 'n_estimators': 300}",0.75,0.732])
x.add_row(['XGBoost',r"{'max_depth': 3, 'n_estimators': 200}",0.78,0.758])
x.add_row(['StackClassifier','-',0.78,0.779])
x.add_row(['Voting Classifier(No stacking + no weights)','-',0.77,0.785])
x.add_row(['Voting Classifier(stacking + no weights)','-',0.76,0.784])
x.add_row(['Voting Classifier(no stacking + weights)','-',0.76,0.785])
x.add_row(['Voting Classifier(stacking + weights)','-',0.74,0.785])
print(x)
```

```
+---------------------------------------------+---------------------------------
--------------------+------+-------+
|                    Model                    |                    Hyerparameter
|  cv  |  test |
+---------------------------------------------+---------------------------------
--------------------+------+-------+
|                     kNN                     |     {'algorithm': 'kd_tree',
'n_neighbors': 35}      | 0.7  | 0.589 |
|             Logistic Regression             | {'C': 1000, 'penalty': 'l1',
'solver': 'liblinear'} | 0.72 | 0.803 |
|                     SVC                     |             {'C': 10, 'kernel':
'rbf'}              | 0.78 | 0.697 |
|                 RandomForest                |       {'max_depth': 5,
'n_estimators': 300}        | 0.75 | 0.732 |
|                   XGBoost                   |       {'max_depth': 3,
'n_estimators': 200}        | 0.78 | 0.758 |
|               StackClassifier               |                           -
| 0.78 | 0.779 |
| Voting Classifier(No stacking + no weights) |                           -
| 0.77 | 0.785 |
|   Voting Classifier(stacking + no weights)  |                           -
| 0.76 | 0.784 |
|   Voting Classifier(no stacking + weights)  |                           -
| 0.76 | 0.785 |
|    Voting Classifier(stacking + weights)    |                           -
| 0.74 | 0.785 |
+---------------------------------------------+---------------------------------
--------------------+------+-------+
```

```
[ ]:
```