Step 1: Import Required Libraries and Functions

If the following code doesn't run, then do 'pip install ipynb' in the command line. This code lets us import functions from notebooks in the lib folder. Lib has all of the feature extraction and model training/predicting functions.

In [1]:
```
import ipynb
import sys
sys.path.append('../lib/')
```

If the following code doesn't run, then do 'pip install imblearn' in the command line. This code lets us do SMOTE (synthetic minority oversampling technique) and random undersampling to help deal with the imbalanced data.

In [2]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
```

Here we import the remaining libraries that we'll need.

In [3]:
```
import pandas as pd
import numpy as np
import math
import os
import scipy.io
import pickle
import bz2
import time
import _pickle as cPickle
from sklearn.metrics import pairwise_distances, classification_report, confusion_matrix, roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from xgboost.sklearn import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import balanced_accuracy_score
from sklearn.model_selection import cross_val_score
```

Finally, we'll import the training and test functions from the lib folder in this cell.

In [4]:
```
from ipynb.fs.full.train_gbm import train_gbm
from ipynb.fs.full.train_xgb import train_xgb
from ipynb.fs.full.train_knn import train_knn
from ipynb.fs.full.train_lda import train_lda
from ipynb.fs.full.train_random_forest import train_random_forest
from ipynb.fs.full.train_logistic import train_logistic
from ipynb.fs.full.train_svm import train_svm
from ipynb.fs.full.train_naive_bayes import train_naive_bayes
from ipynb.fs.full.train_lasso import train_lasso
from ipynb.fs.full.train_bagging import train_bagging

from ipynb.fs.full.test_model import test_model
from ipynb.fs.full.compute_metrics import compute_metrics
```

Step 2: Set Work Directories

In [5]:
```
np.random.seed(2020)
```

Here we set the directories for the training set points and labels.

In [6]:
```
train_dir = '../data/train_set/'
train_image_dir = train_dir+"images/"
train_pt_dir = train_dir+"points/"
train_label_path = train_dir+"label.csv"
```

Step 3: Set Up Controls

In this cell, we have a set of controls for the feature extraction. If true, then we process the features from scratch, and if false, then we load existing features from files in the output folder.

- (T/F) initial feature extraction on training set
- (T/F) initial feature extraction on test set
- (T/F) improved feature extraction on training set
- (T/F) improved feature extraction on test set
- (T/F) SMOTE using improved features on train set
- (T/F) PCA using improved features on training set and test set (doesn't make sense to only do PCA from scratch on train but not test and vice versa so only the option to do it from scratch on both or neither is given)

In [7]:
```
run_feature_train_initial = True
run_feature_test_initial = True

run_feature_train = True
run_feature_test = True

run_feature_train_SMOTE = True

run_feature_PCA = True
```

In this cell, we have a set of controls for model training/testing. If true, then we train the model and generate predictions on the test set, and if false, then we skip that model. By default all the models are set to run.

In [8]:
```
run_baseline = True
run_advanced = True

run_baseline_improved = True
run_baseline_pca = True
run_knn = True
run_knn_smote = True
run_xgboost=True
feature_initial=True
run_random_forest=True
run_LDA=True
run_logistic=True
run_weighted_logistic=True
run_svm = True
run_svm_pca = True
run_weighted_svm = True
run_lasso = True
run_weighted_lasso = True
run_bagging_smote = True
run_naivebayes = True
```

The overwrite_saved_model_results option lets you decide if you want to save the model statistics from your run to a saved model results file. It is recommended that you set it to False if you plan to not run all of the models.

The run_10_cv option runs through the 10-fold cross validation with AUC scoring that we used on weighted logistic, weighted SVM, weighted lasso, and XGBoost with SMOTE to determine which to pick as the advanced model. By default it is set to False as it takes about 3 hours to run.

In [9]:
```
overwrite_saved_model_results = True
run_10_cv = False
```

Step 4: Import Data and Train-Test Split

Here we import the data, and we can see that the dataset is imbalanced and that there are more records with basic emotions than records with complex emotions.

In [10]:
```
info = pd.read_csv(train_label_path)
n = info.shape[0]

#Data is imbalanced
print('Number of records with label 0 (basic emotion):   {:4d} '.format(info.loc[info['label']==0].shape[0]))
print('Number of records with label 1 (complex emotion): {:2d} '.format(info.loc[info['label']==1].shape[0]))
```

```
Number of records with label 0 (basic emotion):   2402
Number of records with label 1 (complex emotion): 598
```

We do an 80-20 train-test split.

In [11]:
```
n_train = int(round(n*(4/5),0))
train_idx = np.random.choice(list(info.index),size=n_train,replace=False)
test_idx = list(set(list(info.index))-set(train_idx)) #set difference
```

Fiducial points are stored in matlab format. In this step, we read them and store them in a list.

In [12]:
```
#function to read fiducial points
#input: index
#output: matrix of fiducial points corresponding to the index

n_files = len(os.listdir(train_pt_dir))

def readMat_matrix(index):
    try:
        mat_data = scipy.io.loadmat(train_pt_dir+'{:04d}'.format(index)+'.mat')['faceCoordinatesUnwarped']
    except KeyError:
        mat_data = scipy.io.loadmat(train_pt_dir+'{:04d}'.format(index)+'.mat')['faceCoordinates2']
    return np.matrix.round(mat_data,0)

#load fiducial points into list and store them in output
fiducial_pt_list = list(map(readMat_matrix,list(range(1,n_files+1))))
pickle.dump(fiducial_pt_list, open( "../output/fiducial_pt_list.p", "wb" ) )
```

Step 5: Construct Features and Responses

Starter Code Features

Use feature.ipynb's feature_initial function to generate pairwise distance features for the baseline model. This is the same feature extraction method as that of the starter code. Note that this method counts distances from x-axis and from y-axis separately between points.

Feature extraction times exclude the time it takes to write to an output file.

In [13]:
```
from ipynb.fs.full.feature import feature_initial

tm_feature_train_intitial = np.nan
if run_feature_train_initial == True:
    start = time.time()
    dat_train_initial = feature_initial(fiducial_pt_list, train_idx, info)
    end = time.time()
    tm_feature_train_initial = end-start
    with bz2.BZ2File('../output/train_data_initial' + '.pbz2', 'w') as f:
        cPickle.dump(dat_train_initial, f)
    print('Initial feature extraction time for train: {:4f}'.format(tm_feature_train_initial))
else:
    dat_train_initial = cPickle.load(bz2.BZ2File('../output/train_data_initial.pbz2', 'rb'))


tm_feature_test_initial = np.nan
if run_feature_test_initial == True:
    start = time.time()
    dat_test_initial = feature_initial(fiducial_pt_list, test_idx, info)
    end = time.time()
    tm_feature_test_initial = end-start
    with bz2.BZ2File('../output/test_data_initial' + '.pbz2', 'w') as f:
        cPickle.dump(dat_test_initial, f)
    print('Initial feature extraction time for test:  {:4f}'.format(tm_feature_test_initial))
else:
    dat_test_initial = cPickle.load(bz2.BZ2File('../output/test_data_initial.pbz2', 'rb'))
```

```
Initial feature extraction time for train: 5.326151
Initial feature extraction time for test:  1.202793
```

In [14]:
```
feature_train_initial = dat_train_initial.loc[:, dat_train_initial.columns != 'labels']
label_train_initial = dat_train_initial['labels']

feature_test_initial = dat_test_initial.loc[:, dat_test_initial.columns != 'labels']
label_test_initial = dat_test_initial['labels']
```

Improved Features

Use feature.ipynb's feature_improved function to generate pairwise euclidean distance features to be used by all of the models other than the baseline. Since feature_improved just uses a single euclidean distance value rather than separate x-distance and y-distance values, feature_improved produces exactly half as many features as feature_initial while keeping the same information.

In [15]:
```
from ipynb.fs.full.feature import feature_improved

tm_feature_train_improved = np.nan
if run_feature_train == True:
    start = time.time()
    dat_train = feature_improved(fiducial_pt_list, train_idx, info)
    end = time.time()
    tm_feature_train_improved = end-start
    with bz2.BZ2File('../output/train_data' + '.pbz2', 'w') as f:
        cPickle.dump(dat_train, f)
    print('Improved feature extraction time for train: {:4f}'.format(tm_feature_train_improved))
else:
    dat_train = cPickle.load(bz2.BZ2File('../output/train_data.pbz2', 'rb'))


tm_feature_test_improved = np.nan
if run_feature_test == True:
    start = time.time()
    dat_test = feature_improved(fiducial_pt_list, test_idx, info)
    end = time.time()
    tm_feature_test_improved = end-start
    with bz2.BZ2File('../output/test_data' + '.pbz2', 'w') as f:
        cPickle.dump(dat_test, f)
    print('Improved feature extraction time for test:  {:4f}'.format(tm_feature_test_improved))
else:
    dat_test = cPickle.load(bz2.BZ2File('../output/test_data.pbz2', 'rb'))
```

```
Improved feature extraction time for train: 0.181775
Improved feature extraction time for test:  0.036049
```

In [16]:
```
feature_train = dat_train.loc[:, dat_train.columns != 'labels']
label_train = dat_train['labels']

feature_test = dat_test.loc[:, dat_test.columns != 'labels']
label_test = dat_test['labels']
```

SMOTE Features

Here we do the feature extraction for SMOTE which will be discussed more in the advanced model section. SMOTE is only done on the training data and not on the test data. SMOTE is a modification of the improved features.

If the improved features are obtained from scratch, then we include the time it takes to get the improved features with the time it takes to use SMOTE. Otherwise, in the case where the improved features are loaded from the disk, we just use the time it takes to use SMOTE on the features.

In [17]:
```
from ipynb.fs.full.feature import feature_SMOTE

tm_feature_train_SMOTE = np.nan
if run_feature_train_SMOTE == True:
    start = time.time()
    dat_train_SMOTE = feature_SMOTE(dat_train)
    end = time.time()
    if pd.isnull(tm_feature_train_improved):
        tm_feature_train_SMOTE = end-start
    else:
        tm_feature_train_SMOTE = (end-start)+tm_feature_train_improved
    with bz2.BZ2File('../output/train_data_SMOTE' + '.pbz2', 'w') as f:
        cPickle.dump(dat_train_SMOTE, f)
    print('SMOTE feature extraction time for train: {:4f}'.format(tm_feature_train_SMOTE))
else:
    dat_train_SMOTE = cPickle.load(bz2.BZ2File('../output/train_data_SMOTE.pbz2', 'rb'))
```

```
SMOTE feature extraction time for train: 2.594856
```

In [18]:
```
feature_train_sm = dat_train_SMOTE.loc[:,dat_train_SMOTE.columns!='labels']
label_train_sm = dat_train_SMOTE['labels']
```

PCA Features

Finally, here we do PCA which is only done for a couple of the model candidates that were not chosen for the advanced model. PCA is done as a modification of the improved features. Also, it doesn't make sense to only do the PCA transformation on one of either the training data or test data, so both are inputs here.

In [19]:
```
from ipynb.fs.full.feature import feature_PCA

tm_feature_train_PCA = np.nan
tm_feature_test_PCA = np.nan

if run_feature_PCA == True:
    [dat_train_PCA, dat_test_PCA, tm_feature_train_PCA, tm_feature_test_PCA] = feature_PCA(dat_train,dat_test)

    if pd.isnull(tm_feature_train_improved)==False:
        tm_feature_train_PCA = tm_feature_train_PCA+tm_feature_train_improved
    if pd.isnull(tm_feature_test_improved)==False:
        tm_feature_test_PCA = tm_feature_test_PCA+tm_feature_test_improved

    with bz2.BZ2File('../output/train_data_PCA' + '.pbz2', 'w') as f:
        cPickle.dump(dat_train_PCA, f)
    with bz2.BZ2File('../output/test_data_PCA' + '.pbz2', 'w') as f:
        cPickle.dump(dat_test_PCA, f)

    print('PCA feature extraction time for train: {:4f}'.format(tm_feature_train_PCA))
    print('PCA feature extraction time for test:  {:4f}'.format(tm_feature_test_PCA))

else:
    dat_train_PCA = cPickle.load(bz2.BZ2File('../output/train_data_PCA.pbz2', 'rb'))
    dat_test_PCA = cPickle.load(bz2.BZ2File('../output/test_data_PCA.pbz2', 'rb'))
```

```
PCA feature extraction time for train: 6.715786
PCA feature extraction time for test:  0.114706
```

In [20]:
```
feature_train_PCA = dat_train_PCA.loc[:,dat_train_PCA.columns!='labels']
label_train_PCA = dat_train_PCA['labels'] #labels are same as label_train

feature_test_PCA = dat_test_PCA.loc[:,dat_test_PCA.columns!='labels']
label_test_PCA = dat_test_PCA['labels'] #labels are same as label_test
```

Step 6: Baseline Model ~58% Balanced Accuracy

Before discussing the models, we will go over the two main metrics we used for finding a better model than the baseline. Since the data is imbalanced, we examined AUC and balanced accuracy rather than the regular accuracy metric.

AUC is the area under the ROC curve which measures the TP vs FP rate as the classification decision threshold changes over time. For imbalanced data, it is a better performance metric than accuracy.

Balanced accuracy is given by the formula

$$balanced\_accuracy = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right).$$

Balanced accuracy is used for imbalanced data as an estimate for the accuracy if the data was balanced, so the true performance of our models on balanced data will be close to the balanced accuracy.

The baseline model is a GBM fitted on the initial pairwise fiducial features. The parameters were chosen from a grid search with AUC scoring. Note that feature extraction times for all models are already known from the previous step, so we do not need to re-calculate them.

We claim that the accuracy of the baseline model on balanced test data is about 58% based on the fact that we get 58% balanced accuracy as shown below.

In [21]:
```
#grid search for optimal parameters
#params = {'learning_rate':[0.01,0.05,0.1,0.5], 'max_depth': [1,2,3], 'n_estimators':[50,100,150]}
#gscv = GridSearchCV(GradientBoostingClassifier(),params,cv=3,scoring='roc_auc').fit(feature_train_initial,label_train_initial)
#gscv.best_params_
# output: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
```

In [22]:
```
#data frame used to store all of the model results
model_results_df = pd.DataFrame(columns=['Feature Extraction Train Time','Feature Extraction Test Time',
                                          'Train Time','Prediction Time','Accuracy','Balanced Accuracy','AUC'])
```

In [23]:
```
if run_baseline == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_initial))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_initial))

    [train_time,baseline] = train_gbm(feature_train_initial,label_train_initial,
                                      learning_rate=0.1,max_depth=3,n_estimators=150)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(baseline,feature_test_initial)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test_initial,label_test_initial,test_preds,baseline)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    #save baseline model
    pickle.dump(baseline,open("../output/baseline.p", "wb"))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_initial,
                    'Feature Extraction Test Time':tm_feature_test_initial,
                     'Train Time':train_time,
                     'Prediction Time':prediction_time,
                    'Accuracy':accuracy,
                    'AUC':auc,
                    'Balanced Accuracy':balanced_accuracy},name='Baseline')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 5.326151 seconds
Feature extraction time for test: 1.202793  seconds

Training time: 186.644306 seconds
Prediction time: 0.076027 seconds

Accuracy: 0.808333
Balanced Accuracy: 0.587563
AUC: 0.785287
```

Step 7: Advanced Model (SMOTEBoost) ~70% Balanced Accuracy

For the advanced model, we decided to use SMOTEBoost, which is a modified version of XGBoost that uses SMOTE (Synethic Minority Oversampling Technique) (add reference). Our model also uses the improved features which do not double count distances, and the parameters were chosen from grid search with AUC scoring.

The idea of SMOTE is to modify the imbalanced training data by randomly undersampling from the majority class and then creating new synthetic minority data that is close to the existing feature space. The modified SMOTE features then have an equal number of data in each class. To see the details of how we implemented SMOTE, check the feature_SMOTE function in feature.ipynb.

We went with this model for a couple of reasons. First of all, it addresses the fact that the training data is imbalanced. It also has a higher AUC, accuracy, and balanced accuracy than the baseline GBM model. Finally, compared to the other candidates for the advanced model, it has the highest AUC from 10 fold cross validation.

Hence, our claimed accuracy with the advanced model on balanced test data is about 70% based on the fact that we get 70% balanced accuracy as shown below.

In [24]:
```
print('Number of records with label 0 after SMOTE (basic emotion):   {:4d} '.format(len(label_train_sm)-sum(label_train_sm)))
print('Number of records with label 1 after SMOTE (complex emotion): {:2d} '.format(sum(label_train_sm)))
```

```
Number of records with label 0 after SMOTE (basic emotion):   1929
Number of records with label 1 after SMOTE (complex emotion): 1929
```

In [25]:
```
if run_advanced == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_SMOTE))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, advanced] = train_xgb(feature_train_sm, label_train_sm, learning_rate=0.25, n_estimators=300,
                                       max_depth=3,min_child_weight=1,objective='binary:logistic',scale_pos_weight=4)
    print('\nTraining time: {:4f} seconds'.format(train_time))


    [prediction_time,test_preds] = test_model(advanced,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,advanced)

    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    pickle.dump(advanced,open("../output/advanced.p", "wb"))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_SMOTE,
                    'Feature Extraction Test Time':tm_feature_test_improved,
                     'Train Time':train_time,
                     'Prediction Time':prediction_time,
                    'Accuracy':accuracy,
                    'AUC':auc,
                    'Balanced Accuracy':balanced_accuracy},name='Advanced (SMOTEBoost)')
    model_results_df.loc['Advanced (SMOTEBoost)'] = row
```

```
Feature extraction time for train: 2.594856 seconds
Feature extraction time for test:  0.036049  seconds

Training time: 206.306090 seconds
Prediction time: 0.068411 seconds

Accuracy: 0.810000
Balanced Accuracy: 0.706697
AUC: 0.827371
```

Optional Step: Remaining Models

This is a pre-made data frame of all of the model run times and metrics in case you want a comparison without running through all of the models.

In [26]:
```
pickle.load(open('../output/model_results.p','rb'))
```

Out[26]:

	Feature Extraction Train Time	Feature Extraction Test Time	Train Time	Prediction Time	Accuracy	Balanced Accuracy	AUC
Baseline	5.043 s	1.285 s	184.954 s	0.046 s	0.808	0.588	0.785
Advanced (SMOTEBoost)	2.505 s	0.037 s	197.78 s	0.07 s	0.810	0.707	0.827
Baseline with Improved Features	0.186 s	0.037 s	176.08 s	0.024 s	0.817	0.610	0.798
Baseline with PCA	6.614 s	0.115 s	1.179 s	0.003 s	0.790	0.536	0.693
KNN	0.186 s	0.037 s	0.371 s	5.421 s	0.792	0.517	0.675
SMOTE KNN	2.505 s	0.037 s	0.723 s	7.67 s	0.607	0.638	0.684
XGBoost	0.186 s	0.037 s	85.464 s	0.075 s	0.813	0.689	0.809
Random Forest	0.186 s	0.037 s	7.452 s	0.032 s	0.810	0.563	0.766
LDA	0.186 s	0.037 s	6.5 s	0.024 s	0.822	0.636	0.786
Logistic Regression	0.186 s	0.037 s	109.518 s	0.023 s	0.822	0.700	0.822
Weighted Logistic	0.186 s	0.037 s	135.264 s	0.025 s	0.830	0.679	0.823
SVM	0.186 s	0.037 s	86.185 s	1.746 s	0.855	0.683	0.828
SVM with PCA	6.614 s	0.115 s	0.812 s	0.018 s	0.787	0.585	0.745
Weighted SVM	0.186 s	0.037 s	396.483 s	1.844 s	0.837	0.718	0.837
Naive Bayes	0.186 s	0.037 s	0.141 s	0.039 s	0.663	0.628	0.660
Lasso	0.186 s	0.037 s	208.918 s	0.019 s	0.825	0.693	0.826
Weighted Lasso	0.186 s	0.037 s	61.742 s	0.018 s	0.832	0.623	0.819
SMOTE Bagging	2.505 s	0.037 s	656.995 s	0.677 s	0.807	0.633	0.795

In thie rest of this step, we run through the other models that were candidates for the advanced model.

Baseline Model with Improved Features

In [27]:
```
#grid search for optimal parameters
#params = {'learning_rate':[0.01,0.05,0.1,0.5], 'max_depth': [1,2,3], 'n_estimators':[50,100,150]}
#gscv = GridSearchCV(GradientBoostingClassifier(),params,cv=3,scoring='roc_auc').fit(feature_train,label_train)
#gscv.best_params_
# output: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 150}
```

In [28]:
```
if run_baseline_improved == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, baseline_improved] = train_gbm(feature_train,label_train,
                                                learning_rate=0.1,max_depth=3,n_estimators=150)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(baseline_improved,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,baseline_improved)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
                'Feature Extraction Test Time':tm_feature_test_improved,
                 'Train Time':train_time,
                 'Prediction Time':prediction_time,
                'Accuracy':accuracy,
                'AUC':auc,
                'Balanced Accuracy':balanced_accuracy},name='Baseline with Improved Features')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test:  0.036049  seconds

Training time: 183.706270 seconds
Prediction time: 0.023944 seconds

Accuracy: 0.816667
Balanced Accuracy: 0.610128
AUC: 0.797856
```

Baseline Model With PCA

In [29]:
```
#params = {'learning_rate':[0.01,0.05,0.1,0.5], 'max_depth': [1,2,3], 'n_estimators':[50,100,150]}
#gscv = GridSearchCV(GradientBoostingClassifier(),params,cv=3).fit(feature_train_PCA,label_train)
#gscv.best_params_
#output: {'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 150}
```

In [30]:
```
if run_baseline_pca == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_PCA))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_PCA))

    [train_time, baseline_PCA] = train_gbm(feature_train_PCA,label_train,
                                           learning_rate=0.1,max_depth=2,n_estimators=150)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(baseline_PCA,feature_test_PCA)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test_PCA,label_test,test_preds,baseline_PCA)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_PCA,
            'Feature Extraction Test Time':tm_feature_test_PCA,
             'Train Time':train_time,
             'Prediction Time':prediction_time,
            'Accuracy':accuracy,
            'AUC':auc,
            'Balanced Accuracy':balanced_accuracy},name='Baseline with PCA')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 6.715786 seconds
Feature extraction time for test:  0.114706  seconds

Training time: 1.191914 seconds
Prediction time: 0.002096 seconds

Accuracy: 0.790000
Balanced Accuracy: 0.535616
AUC: 0.692814
```

KNN Model

In [31]:
```
#params = {'n_neighbors':list(range(5,55,5))}
#gscv = GridSearchCV(KNeighborsClassifier(),params,cv=5).fit(feature_train,label_train)
#gscv.best_params_
#output: {'n_neighbors': 25}
```

In [32]:
```
if run_knn == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, knn] = train_knn(feature_train,label_train,n_neighbors=25)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(knn,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,knn)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='KNN')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test:  0.036049  seconds

Training time: 0.385187 seconds
Prediction time: 5.693887 seconds

Accuracy: 0.791667
Balanced Accuracy: 0.516514
AUC: 0.674527
```

SMOTE KNN

In [33]:
```
#params = {'n_neighbors':list(range(5,55,5))}
#gscv = GridSearchCV(KNeighborsClassifier(),params,cv=5).fit(feature_train_sm,label_train_sm)
#gscv.best_params_
#output: {'n_neighbors': 5}
```

In [34]:
```
if run_knn_smote == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_SMOTE))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, knn] = train_knn(feature_train_sm,label_train_sm,n_neighbors=5)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(knn,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,knn)
    balanced_accuracy = balanced_accuracy_score(label_test,test_preds)

    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_SMOTE,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='SMOTE KNN')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 2.594856 seconds
Feature extraction time for test:  0.036049  seconds

Training time: 0.752246 seconds
Prediction time: 7.720591 seconds

Accuracy: 0.606667
Balanced Accuracy: 0.638211
AUC: 0.683691
```

XGBoost Model

In [35]:
```
if run_xgboost == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test:  {:4f} seconds'.format(tm_feature_test_improved))

    [train_time, xgb] = train_xgb(feature_train, label_train, learning_rate=0.1, n_estimators=200,
                                  max_depth=3,min_child_weight=1,objective='binary:logistic',scale_pos_weight=4)
    print('\nTraining time: {:4f} seconds'.format(train_time))


    [prediction_time,test_preds] = test_model(xgb,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,xgb)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='XGBoost')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test:  0.036049 seconds

Training time: 85.553323 seconds
Prediction time: 0.068330 seconds

Accuracy: 0.813333
Balanced Accuracy: 0.688652
AUC: 0.808726
```

Random Forest Model

In [36]:
```
if run_random_forest==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, rf_model] = train_random_forest(feature_train,label_train,n_estimators=100,criterion='gini',min_samples_leaf=1,max_features='sqrt')
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(rf_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,rf_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Random Forest')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 6.693948 seconds
Prediction time: 0.035192 seconds

Accuracy: 0.810000
Balanced Accuracy: 0.562701
AUC: 0.766485
```

LDA Model

In [37]:
```python
if run_LDA==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, lda_model] = train_lda(feature_train, label_train,solver='eigen', shrinkage=.1, n_comp
onents=1)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lda_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lda_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='LDA')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 5.942372 seconds
Prediction time: 0.019037 seconds

Accuracy: 0.821667
Balanced Accuracy: 0.636339
AUC: 0.786203
```

Logistic Model

In [38]:
```python
#grid={"C":[0.001,0.01,0.1,0.25,0.5,1,10]}
#cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
#gscv = GridSearchCV(LogisticRegression(dual=False, fit_intercept=True,
#                    intercept_scaling=1, max_iter=1200000,
#                    multi_class='multinomial', penalty='l2',
#                    solver='lbfgs', tol=0.0001),grid,cv=cv,return_train_score=True)
#gscv.fit(feature_train,label_train)
#gscv.best_params_
#output: {'C': 0.01}
```

In [39]:
```python
if run_logistic==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:1.0, 1:1.0}
    [train_time,lr_model] = train_logistic(feature_train,label_train,
                                           C=0.01, dual=False, fit_intercept=True,
                                           intercept_scaling=1, max_iter=1200000,
                                           multi_class='multinomial', penalty='l2',
                                           solver='lbfgs', tol=0.0001,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lr_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lr_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Logistic Regression')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 97.020723 seconds
Prediction time: 0.022679 seconds

Accuracy: 0.821667
Balanced Accuracy: 0.699697
AUC: 0.822310
```

Weighted Logistic Model

In [40]:
```python
#grid={"C":[0.001,0.01,0.1,0.25,0.5,1,10]}
#weights = {0:80.0, 1:20.0}
#cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
#gscv = GridSearchCV(LogisticRegression(dual=False, fit_intercept=True,
#                    intercept_scaling=1, max_iter=1200000,
#                    multi_class='multinomial', penalty='l2',
#                    solver='lbfgs', tol=0.0001,class_weight=weights),grid,cv=cv,return_train_score=Tru
e)
#gscv.fit(feature_train,label_train)
#gscv.best_params_
#output: {'C': 0.001}
```

In [41]:
```python
if run_weighted_logistic==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:80.0, 1:20.0}
    [train_time,lr_w] = train_logistic(feature_train,label_train,
                                           C=0.001, dual=False, fit_intercept=True,
                                           intercept_scaling=1, max_iter=120000000000,
                                           multi_class='multinomial', penalty='l2',
                                           solver='lbfgs', tol=0.0001,class_weight=weights)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lr_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lr_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Weighted Logistic')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 122.259835 seconds
Prediction time: 0.022598 seconds

Accuracy: 0.830000
Balanced Accuracy: 0.679063
AUC: 0.822843
```

```
/Users/rohan/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:764: Converg
enceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of f AND g EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

SVM

In [42]:
```python
# #grid search with cv 3 to find the best performed parameters
# param= {'C': [0.00001,0.0001,0.001,0.01,1,10],
#        'kernel':['linear', 'rbf', 'poly'],
#        'degree':[2,3,4]}

# gscv = GridSearchCV(SVC(random_state = 2020), param, cv=3, return_train_score=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# #output: {'C': 10, 'degree': 4, 'kernel': 'poly'}
```

In [43]:
```python
from ipynb.fs.full.train_svm import train_svm

#improved svm using parameters from grid search
if run_svm==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:1.0, 1:1.0}
    [train_time,svm_model] = train_svm(feature_train,label_train,
                                       C=10, kernel='poly', degree=4,
                                       probability=True,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,svm_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
         'Train Time':train_time,
         'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='SVM')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 79.830844 seconds
Prediction time: 1.624750 seconds

Accuracy: 0.855000
Balanced Accuracy: 0.683400
AUC: 0.828070
```

SVM With PCA

In [44]:
```python
# #grid search with cv 3 to find the best performed parameters
# param= {'C': [0.001,0.01,1,10,15,20],
#        'kernel':['linear', 'rbf', 'poly'],
#        'degree':[2,3,4]}

# gscv = GridSearchCV(SVC(random_state = 2020), param, cv=3, return_train_score=True)
# gscv.fit(feature_train_PCA,label_train)
# gscv.best_params_
# #output: {'C': 10, 'degree': 2, 'kernel': 'rbf'}
```

In [45]:
```python
#improved svm with PCA

if run_svm_pca==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_PCA))
    print('Feature extraction time for test:  {:4f}  seconds'.format(tm_feature_test_PCA))

    weights = {0:1.0, 1:1.0}
    [train_time, svm_PCA] = train_svm(feature_train_PCA,label_train_PCA,
                                      C=10,degree=2,kernel='rbf',probability=True,
                                     class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_PCA,feature_test_PCA)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test_PCA,label_test_PCA,test_preds,svm
_PCA)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_PCA,
        'Feature Extraction Test Time':tm_feature_test_PCA,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='SVM with PCA')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 6.715786 seconds
Feature extraction time for test:  0.114706  seconds

Training time: 0.746498 seconds
Prediction time: 0.018930 seconds

Accuracy: 0.786667
Balanced Accuracy: 0.585341
AUC: 0.745118
```

Weighted SVM

In [46]:
```python
# weights = {0:80.0, 1:20.0}
# params= {'C': [1,10,15,20],
#        'kernel':['linear', 'rbf', 'poly'],
#        'degree':[2,3,4]}
# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(SVC(class_weight=weights,random_state = 2020,probability=True), params, cv=3, sco
ring='roc_auc',verbose=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# #output: output: {'C': 10, 'degree': 4, 'kernel': 'poly'}
```

In [47]:
```python
if run_weighted_svm==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:80.0, 1:20.0}
    [train_time,svm_w] = train_svm(feature_train,label_train,
                                   C=10, kernel='poly', degree=4,
                                   probability=True,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))


    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,svm_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Weighted SVM')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 384.685321 seconds
Prediction time: 1.667472 seconds

Accuracy: 0.836667
Balanced Accuracy: 0.717851
AUC: 0.837193
```

Naive Bayes

In [48]:
```python
if run_naivebayes == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time,gnb] = train_naive_bayes(feature_train,label_train)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(gnb,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,gnb)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Naive Bayes')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 0.138798 seconds
Prediction time: 0.037708 seconds

Accuracy: 0.663333
Balanced Accuracy: 0.628073
AUC: 0.660369
```

Lasso

In [49]:
```python
if run_lasso==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:1.0, 1:1.0}
    [train_time, lasso_model] = train_lasso(feature_train,label_train,
                                            penalty='l1',solver='liblinear',
                                            class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lasso_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lasso_model
)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Lasso')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 197.398233 seconds
Prediction time: 0.017074 seconds

Accuracy: 0.825000
Balanced Accuracy: 0.693171
AUC: 0.826389
```

Weighted Lasso

In [50]:
```python
# weights = {0:80.0, 1:20.0}
# param= {'solver':['liblinear', 'saga']}
# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticRegression(penalty='l1', class_weight=weights), params, cv=3, scoring='ro
c_auc',verbose=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
```

In [51]:
```python
if run_weighted_lasso==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    weights = {0:80.0, 1:20.0}
    [train_time, lasso_w] = train_lasso(feature_train,label_train,
                                        penalty='l1',solver='liblinear',
                                        class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lasso_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lasso_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_improved,
        'Feature Extraction Test Time':tm_feature_test_improved,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='Weighted Lasso')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 56.160090 seconds
Prediction time: 0.017086 seconds

Accuracy: 0.831667
Balanced Accuracy: 0.622522
AUC: 0.819164
```

SMOTE Bagging

In [52]:
```python
#params = {'n_estimators':[25,50,75,100]}
#cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
#gscv = GridSearchCV(BaggingClassifier(),params,cv=cv,scoring='roc_auc').fit(feature_train_sm,label_tra
in_sm)
#gscv.best_params_
#output: {{'n_estimators': 100}}
```

In [53]:
```python
if run_bagging_smote == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_SMOTE))
    print('Feature extraction time for test: {:4f}  seconds'.format(tm_feature_test_improved))

    [train_time, smote_bagging] = train_bagging(feature_train_sm,label_train_sm,n_estimators=100)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(smote_bagging,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,smote_baggi
ng)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series({'Feature Extraction Train Time':tm_feature_train_SMOTE,
        'Feature Extraction Test Time':tm_feature_test_improved,
        'Train Time':train_time,
        'Prediction Time':prediction_time,
        'Accuracy':accuracy,
        'AUC':auc,
        'Balanced Accuracy':balanced_accuracy},name='SMOTE Bagging')
    model_results_df = model_results_df.append(row)
```

```
Feature extraction time for train: 2.594856 seconds
Feature extraction time for test: 0.036049  seconds

Training time: 639.069607 seconds
Prediction time: 0.614470 seconds

Accuracy: 0.806667
Balanced Accuracy: 0.632585
AUC: 0.794535
```

Model Results Table

In this part, we display the model results table for all of the models that were set to run.

In [54]:
```python
model_results_df = model_results_df.applymap(lambda x: round(x,3))
model_results_df['Train Time'] = [str(x)+' s' for x in list(model_results_df['Train Time'])]
model_results_df['Prediction Time'] = [str(x)+' s' for x in list(model_results_df['Prediction Time'])]
model_results_df['Feature Extraction Train Time'] = [str(x)+' s' for x in list(model_results_df['Featur
e Extraction Train Time'])]
model_results_df['Feature Extraction Test Time'] = [str(x)+' s' for x in list(model_results_df['Feature
 Extraction Test Time'])]
```

In [55]:
```python
model_results_df
```

Out[55]:

	Feature Extraction Train Time	Feature Extraction Test Time	Train Time	Prediction Time	Accuracy	Balanced Accuracy	AUC
Baseline	5.326 s	1.203 s	186.644 s	0.076 s	0.808	0.588	0.785
Advanced (SMOTEBoost)	2.595 s	0.036 s	206.306 s	0.068 s	0.810	0.707	0.827
Baseline with Improved Features	0.182 s	0.036 s	183.706 s	0.024 s	0.817	0.610	0.798
Baseline with PCA	6.716 s	0.115 s	1.192 s	0.002 s	0.790	0.536	0.693
KNN	0.182 s	0.036 s	0.385 s	5.694 s	0.792	0.517	0.675
SMOTE KNN	2.595 s	0.036 s	0.752 s	7.721 s	0.607	0.638	0.684
XGBoost	0.182 s	0.036 s	85.553 s	0.068 s	0.813	0.689	0.809
Random Forest	0.182 s	0.036 s	6.694 s	0.035 s	0.810	0.563	0.766
LDA	0.182 s	0.036 s	5.942 s	0.019 s	0.822	0.636	0.786
Logistic Regression	0.182 s	0.036 s	97.021 s	0.023 s	0.822	0.700	0.822
Weighted Logistic	0.182 s	0.036 s	122.26 s	0.023 s	0.830	0.679	0.823
SVM	0.182 s	0.036 s	79.831 s	1.625 s	0.855	0.683	0.828
SVM with PCA	6.716 s	0.115 s	0.746 s	0.019 s	0.787	0.585	0.745
Weighted SVM	0.182 s	0.036 s	384.685 s	1.667 s	0.837	0.718	0.837
Naive Bayes	0.182 s	0.036 s	0.139 s	0.038 s	0.663	0.628	0.660
Lasso	0.182 s	0.036 s	197.398 s	0.017 s	0.825	0.693	0.826
Weighted Lasso	0.182 s	0.036 s	56.16 s	0.017 s	0.832	0.623	0.819
SMOTE Bagging	2.595 s	0.036 s	639.07 s	0.614 s	0.807	0.633	0.795

In [56]:
```python
#store model_results
if overwrite_saved_model_results == True:
    pickle.dump(model_results_df, open( "../output/model_results.p", "wb" ) )
```

Optional Step: 10-fold Cross Validation

This is how we did the 10-fold cross validation with AUC scoring to choose from the best candidate models. By default the option to run this cell is set to False.

In [57]:
```python
if run_10_cv == True:
    # weighted lasso
    print("Cross Validation Score: ", np.mean(cross_val_score(lasso_w, feature_train, label_train, cv=1
0, scoring='roc_auc')))
    # output:0.8010359440647241

    # weighted svm
    print("Cross Validation Score: ", np.mean(cross_val_score(svm_w, feature_train, label_train, cv=10,
 scoring='roc_auc')))
    # output:0.8117648633580459

    # weighted Logistic
    print("Cross Validation Score: ", np.mean(cross_val_score(lr_w, feature_train, label_train, cv=10,
scoring='roc_auc')))
    # output:0.8300000000000001

    # XGBoost with SMOTE
    print("Cross Validation Score: ", np.mean(cross_val_score(advanced, feature_train, label_train, cv=
10, scoring='roc_auc')))
    # output:0.8387998261113715
```

References

1. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
2. https://arxiv.org/pdf/1106.1813.pdf
3. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3648438/