Final_Model_withoutcategorical

November 5, 2018

Partners for this Homework are: Pinaki Bhagat
Sydney Correa
Kaggle Team Name: 20178ml08
Ideas tried in multiple subsmissions are outlined below:

1. Combine x,y,z into a single point
2. Look at subject vs phase vs output mapping and add additional features in the dataset
3. Drop the following:
   a. Original "subject","phase","state","indicator"
   b. Drop all 0 value featues
   c. Drop all highly correlated features (>.9)
4. Perform PCA
5. Handle distribution of class weight in several ways:
   a. Use class weight parameter for providing weighted values for uneven output classes
   b. Use upsampling technique (SMOTE)
6. Use individual algorithms such as Random Forest and XGBoost to find out accuracy and AUC
   a. Hyper tuning of parameters
7. Use primary components from PCA to develop model
8. Try Stacking with 4 initial algorithms and use final as XGBoost

Though a host of ways have been explored and multiple submissions made, AUC never went up beyond 62%. Considering heavy work loads (year end approaching) we didn't get as much time as we would like to spend on this. Given the luxury of time, we would have tried several other methods like these below to better our predictions:
a. Upsampling of clusters
b. Few other ensemble methods

Import all python libraries

In [1]:
```
# special IPython command to prepare the notebook for matplotlib
%matplotlib inline

import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt

from datetime import datetime
import requests
from io import BytesIO
import seaborn

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn import metrics


# special matplotlib command for global plot configuration
from matplotlib import rcParams
import matplotlib.cm as cm
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap

import sklearn
from sklearn import neighbors, decomposition, metrics, preprocessing
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import metrics

from sklearn import decomposition, preprocessing
from scipy.spatial import distance

import math
import warnings

warnings.simplefilter("ignore")
```

Define function to combine x,y,z inputs and code categorical variables

```
In [2]: def sensor_combine(sensor_in):

            #from scipy.spatial import distance
            sensor_x = np.array
            sensor_y = np.array
            sensor_z = np.array

            sensor_x = sensor_in.iloc[:,0:221].values
            sensor_y = sensor_in.iloc[:,222:443].values
            sensor_z = sensor_in.iloc[:,444:665].values

            sensor_xyz = np.sqrt(np.square(sensor_x) + np.square(sensor_y) + np.square(sensor_y

            df_sensor_xyz = pd.DataFrame(data=(sensor_xyz))

            #adding categorical encoding
            sensor_catg = sensor_in[['SubA_Phase1','SubI_Phase1','SubM_Phase1','SubA_Phase2',
                                     'SubF_Phase2','SubI_Phase3','SubL_Phase3','SubL_Phase4','Su

            frames = [df_sensor_xyz, sensor_catg]
            df_sensor_combine = pd.concat(frames, axis=1)

            return df_sensor_combine
```

Download train dataset

```
In [3]: #url = 'C:\\Users\\corre\\Desktop\\CSCI E-82\\PS 4\\all\\train_data.csv'
        url = 'C:\\Users\\pinakibhagat\\Downloads\\Personal\\Harvard\\CSCIE-82\\Homework 4\\al
        sensor_train = pd.read_csv(url, sep=',')

        print(sensor_train.shape)

(4584, 670)
```

Download test dataset

```
In [4]: #url1 = 'C:\\Users\\corre\\Desktop\\CSCI E-82\\PS 4\\all\\test_data.csv'
        url1 = 'C:\\Users\\pinakibhagat\\Downloads\\Personal\\Harvard\\CSCIE-82\\Homework 4\\a
        sensor_test = pd.read_csv(url1, sep=',')

        print(sensor_test.shape)

(1732, 669)
```

Add indicator to combine and split later

In [5]:
```
sensor_train['indicator']='train'
sensor_test['indicator']='test'

Y = sensor_train.output
sensor_train.drop('output',inplace=True, axis=1)
```

Combine test and train datasets

In [6]:
```
df_sensor_all = pd.concat([sensor_train,sensor_test])
df_sensor_all.reset_index(inplace=True, drop=True) #drop index
df_sensor_all.shape
```

Out[6]: (6316, 670)

Create additional features based on outcome of feature engineering

In [7]:
```
df_sensor_all['SubjectK'] = 0
df_sensor_all.loc[df_sensor_all[df_sensor_all.subject=='K'].index,'SubjectK']=1
```

In [8]:
```
#Step by step
i1 = df_sensor_all[((df_sensor_all.phase==1) & (df_sensor_all.subject=='A'))].index
arr = np.zeros(df_sensor_all.shape[0],dtype=int)
arr[i1]=1
df_sensor_all['SubA_Phase1']=arr

#concise
df_sensor_all['SubI_Phase1'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==1) & (df_sensor_all.subject=='I

df_sensor_all['SubM_Phase1'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==1) & (df_sensor_all.subject=='M

df_sensor_all['SubA_Phase2'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==2) & (df_sensor_all.subject=='A

df_sensor_all['SubF_Phase2'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==2) & (df_sensor_all.subject=='F

df_sensor_all['SubI_Phase3'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==3) & (df_sensor_all.subject=='I

df_sensor_all['SubL_Phase3'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==3) & (df_sensor_all.subject=='L

df_sensor_all['SubL_Phase4'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==4) & (df_sensor_all.subject=='L

df_sensor_all['SubI_Phase4'] = 0
df_sensor_all.loc[df_sensor_all[((df_sensor_all.phase==4) & (df_sensor_all.subject=='I
```

Split back into train and test datasets

In [9]:
```
sensor_train_1 = df_sensor_all[df_sensor_all.indicator=='train']
sensor_test_1 = df_sensor_all[df_sensor_all.indicator=='test']
sensor_train_1.reset_index(inplace=True,drop=True)
sensor_test_1.reset_index(inplace=True,drop=True)
```

Drop the categorical old features

In [10]:
```
sensor_train_1 = sensor_train_1.drop(['state','subject','phase','indicator'], axis=1)
sensor_test_1 = sensor_test_1.drop(['state','subject','phase','indicator'], axis=1)
```

Drop all features that are 0 and highly co-related

In [11]:
```
df_sensor_train = sensor_combine(sensor_train_1)
print(df_sensor_train.shape)
```

```
(4584, 230)
```

In [12]:
```
df_sensor_test = sensor_combine(sensor_test_1)
print(df_sensor_test.shape)

#Combine train and test to make a single dataframe
frames = [df_sensor_train, df_sensor_test]
df_sensor_all = pd.concat(frames)
print(df_sensor_all.shape)
#Remove all columns that are zero
df_sensor_all = df_sensor_all.loc[:, (df_sensor_all != 0).any(axis=0)]
print(df_sensor_all.shape)

#Remove all highly correlated features
corr_matrix = df_sensor_all.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.90
to_drop = [column for column in upper.columns if any(upper[column] > 0.90)]

to_drop1=to_drop[1:110]

# Drop features
df_sensor_all = df_sensor_all.drop(df_sensor_all.columns[to_drop1], axis=1)

df_sensor_train = df_sensor_all.iloc[:4584,:]
print(df_sensor_train.shape)

df_sensor_test = df_sensor_all.iloc[4584:,:]
print(df_sensor_test.shape)
```

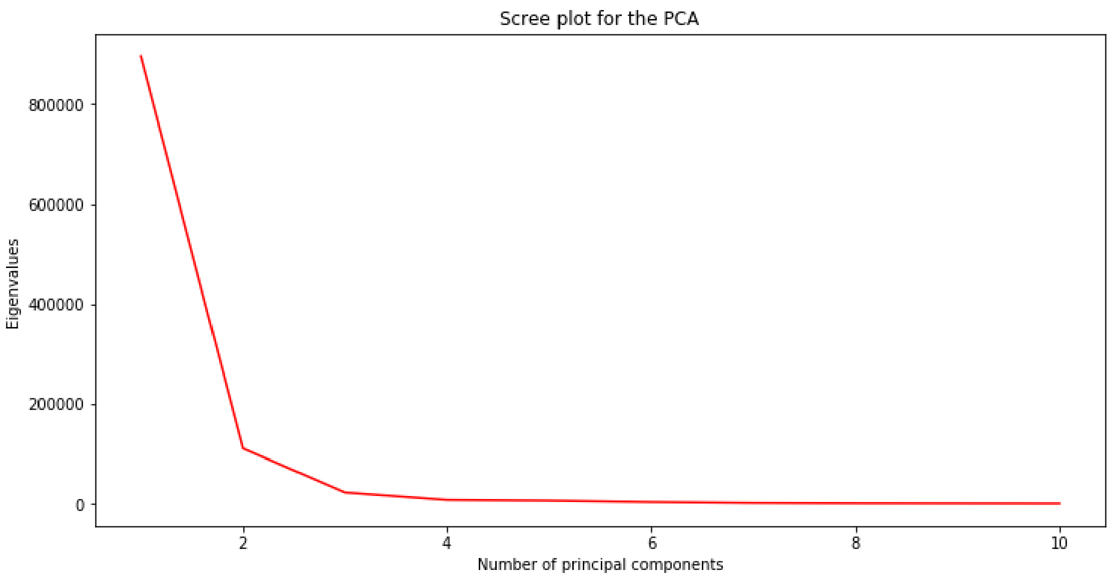
```
X = df_sensor_train.values
```

```
(1732, 230)
(6316, 230)
(6316, 214)
(4584, 105)
(1732, 105)
```

Perform PCA

```
In [13]: pca = sklearn.decomposition.PCA(n_components=10).fit(df_sensor_train)

         coef_PCA = pca.transform(df_sensor_train)
         # we make a scree plot to see how many Principal Components to consider
         plt.figure(figsize=(12, 6))
         eig = pca.explained_variance_
         # and calculate the variance explained by the PC analysis
         var_exp = pca.explained_variance_ratio_.cumsum()*100.
         print(var_exp)

         plt.plot(np.arange(1,len(eig)+1), eig, color='r')
         plt.title('Scree plot for the PCA')
         plt.xlabel('Number of principal components')
         plt.ylabel('Eigenvalues')
         plt.show()

         print ('The 1st Principal Component explains {:03.1f} % of the variance\n'.format(var_
         print ('The 1st and 2nd Principal Components explain {:03.1f} % of the variance\n'.fo
         print ('The 1st, 2nd and 3rd Principal Components explain {:03.1f} % of the variance\
         print ('The 1st, 2nd, 3rd and 4th Principal Components explain {:03.1f} % of the varia
         print ('The first five Principal Components explain {:03.1f} % of the variance\n'.for
```

```
[ 85.49867502  96.04306513  98.16691227  98.87239446  99.44205461
  99.7388009   99.85919166  99.91912986  99.95739594  99.9729197 ]
```




```
The 1st Principal Component explains 85.5 % of the variance

The 1st and 2nd Principal Components explain 96.0 % of the variance

The 1st, 2nd and 3rd Principal Components explain 98.2 % of the variance

The 1st, 2nd, 3rd and 4th Principal Components explain 98.9 % of the variance

The first five Principal Components explain 99.4 % of the variance
```

Break the dataframe into train, validation and test set

```
In [16]: X_train, X_test, y_train, y_test = train_test_split(X, Y, stratify = Y, test_size=0.33
```

Define function for confusion matrix

```
In [17]: def show_confusion_matrix(cm, target_names):
             plt.figure(figsize=(5, 5))
             plt.imshow(cm, interpolation='nearest', cmap=plt.cm.binary)
             plt.title('Confusion matrix')
             plt.set_cmap('Blues')
             plt.colorbar()
             tick_marks = np.arange(len(target_names))
             plt.xticks(tick_marks, target_names, rotation=60)
             plt.yticks(tick_marks, target_names)
             plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label')
    plt.show()
```

Perform SMOTE for handling class imbalance

```
In [18]: import imblearn
         from imblearn.over_sampling import SMOTE

In [19]: #UP Sampling of minority class
         class_counts = np.bincount(y_train.astype(int))
         print(class_counts)

[ 476 2595]


In [20]: np.bincount(y_train.astype(int))*100/len(y_train)

Out[20]: array([ 15.49983719,  84.50016281])

In [21]: sm = SMOTE(random_state=101)
         X_train_upsampled, y_train_upsampled = sm.fit_sample(X_train, y_train)
         np.bincount(y_train_upsampled.astype(int))
         np.bincount(y_train_upsampled.astype(int))*100/len(y_train_upsampled)
         print(X_train_upsampled.shape)
         print(y_train_upsampled.shape)

(5190, 105)
(5190,)


In [30]: X_train = X_train_upsampled
         y_train = y_train_upsampled

In [31]: X_test.shape

Out[31]: (1513, 105)
```

Stacking models for better performance

```
In [32]: # Some useful parameters which will come in handy later on
         from sklearn.model_selection import KFold
         ntrain = X_train.shape[0]
         ntest = X_test.shape[0]
         SEED = 0 # for reproducibility
         NFOLDS = 5 # set folds for out-of-fold prediction
         kf = KFold(n_splits= NFOLDS,shuffle=True, random_state=SEED)

         # Sklearn classifier
         class SklearnHelper:
             def __init__(self, clf, seed=0, params=None):
```

```
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, X_train, y_train):
        self.clf.fit(X_train, y_train)

    def predict(self, x):
        return self.clf.predict(X)

    def predict_proba(self, x):
        return self.clf.predict_proba(x)

    def fit(self,x,y):
        return self.clf.fit(x,y)

    def feature_importances(self,x,y):
        print(self.clf.fit(x,y).feature_importances_)
```

Out of fold predictions

```
In [33]: def get_oof(clf, X_train, y_train, X_test):
             oof_train = np.zeros((ntrain,))
             oof_test = np.zeros((ntest,))
             oof_test_skf = np.empty((NFOLDS, ntest))

             for i, (train_index, test_index) in enumerate(kf.split(X_train)):
                 x_tr = X_train[train_index]
                 y_tr = y_train[train_index]
                 x_te = X_train[test_index]

                 clf.train(x_tr, y_tr)

                 oof_train[test_index] = clf.predict_proba(x_te)[:,1]
                 oof_test_skf[i, :] = clf.predict_proba(X_test)[:,1]

             oof_test[:] = oof_test_skf.mean(axis=0)
             return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)
```

Generate base first models

```
In [34]: # Put in our parameters for said classifiers
         # Random Forest parameters
         rf_params = {
             'n_jobs': -1,
             'n_estimators': 800,
             'max_depth': 40,
             'min_samples_leaf': 2,
```

```
    'max_features' : 'sqrt',
    'verbose': 0
}

# Extra Trees Parameters
et_params = {
    'n_jobs': -1,
    'n_estimators':800,
    'max_depth': 40,
    'min_samples_leaf': 2,
    'verbose': 0
}

# AdaBoost parameters
ada_params = {
    'n_estimators': 800,
    'learning_rate' : 0.75
}

# Gradient Boosting parameters
gb_params = {
    'n_estimators': 800,
    'max_depth': 40,
    'min_samples_leaf': 2,
    'verbose': 0
}
```

In [35]:
```
# Create 5 objects that represent our 4 models
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, ExtraTreesClassifier)
rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf_params)
et = SklearnHelper(clf=ExtraTreesClassifier, seed=SEED, params=et_params)
ada = SklearnHelper(clf=AdaBoostClassifier, seed=SEED, params=ada_params)
gb = SklearnHelper(clf=GradientBoostingClassifier, seed=SEED, params=gb_params)
#svc = SklearnHelper(clf=SVC, seed=SEED, params=svc_params)
```

In [36]:
```
# Create our OOF train and test predictions. These base results will be used as new f
et_oof_train, et_oof_test = get_oof(et, X_train, y_train, X_test) # Extra Trees
rf_oof_train, rf_oof_test = get_oof(rf,X_train, y_train, X_test) # Random Forest
ada_oof_train, ada_oof_test = get_oof(ada, X_train, y_train, X_test) # AdaBoost
gb_oof_train, gb_oof_test = get_oof(gb,X_train, y_train, X_test) # Gradient Boost

print("Training is complete")
```

```
Training is complete
```

Second level learning model via XGBoost

In [37]:
```
base_predictions_train = pd.DataFrame( {'RandomForest': rf_oof_train.ravel(),
     'ExtraTrees': et_oof_train.ravel(),
     'AdaBoost': ada_oof_train.ravel(),
     'GradientBoost': gb_oof_train.ravel()
     #'SVM': svc_oof_train.ravel()
    })

base_predictions_train.sort_values
base_predictions_train.head()
```

Out[37]:

	AdaBoost	ExtraTrees	GradientBoost	RandomForest
0	0.498930	0.565516	0.999602	0.487250
1	0.500084	0.237121	0.999433	0.432854
2	0.511931	0.836896	0.998967	0.715427
3	0.511456	0.845354	0.934809	0.630500
4	0.500669	0.471522	0.998808	0.409905

Check model correlation

In [38]:
```
base_predictions_train.corr()
```

Out[38]:

	AdaBoost	ExtraTrees	GradientBoost	RandomForest
AdaBoost	1.000000	0.728739	0.568340	0.645212
ExtraTrees	0.728739	1.000000	0.813826	0.949778
GradientBoost	0.568340	0.813826	1.000000	0.847096
RandomForest	0.645212	0.949778	0.847096	1.000000

Second level learning via XGBoost

In [41]:
```
X_train_1 = np.concatenate(( et_oof_train, rf_oof_train, ada_oof_train, gb_oof_train )
X_test_1 = np.concatenate(( et_oof_test, rf_oof_test, ada_oof_test, gb_oof_test), axi
```

In [42]:
```
import xgboost as xgb
gbm = xgb.XGBClassifier(
 learning_rate = 0.02,
 n_estimators= 800,
 max_depth= 40,
 min_child_weight= 2,
 gamma=0.9,    #Regularization parameter
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(X_train_1, y_train)

predictions = gbm.predict_proba(X_test_1)
```

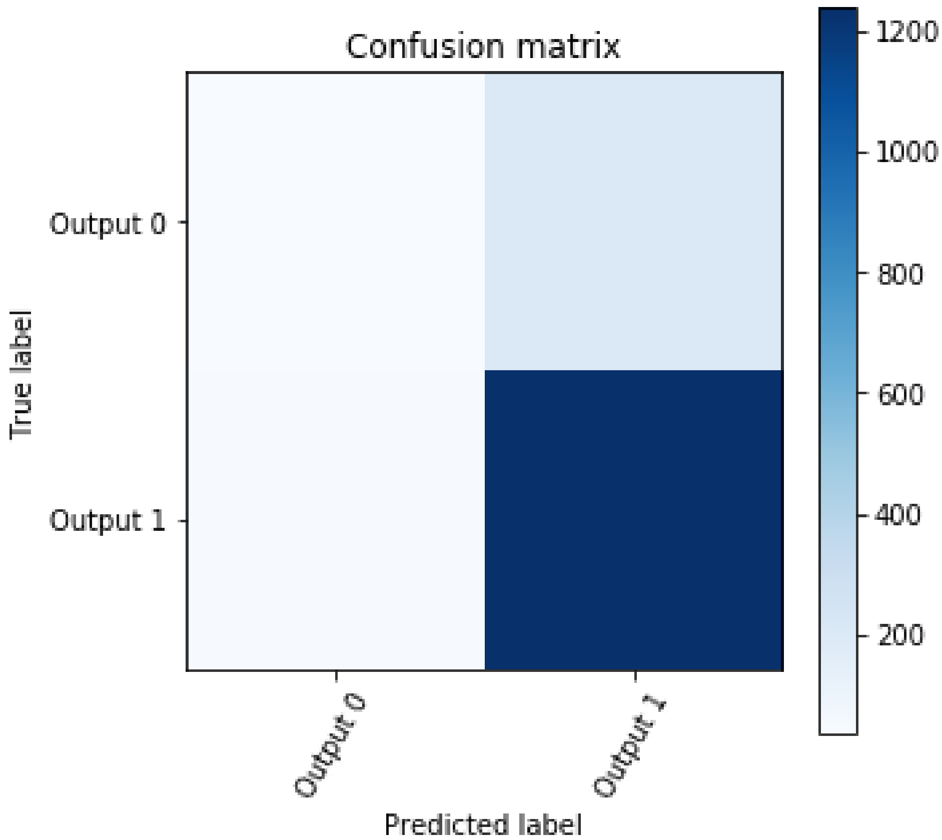
In [43]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predictions[:,1]>0.5)
```

Out[43]: 0.84137475214805024

Check confusion matrix

```
In [44]: targets =['Output 0','Output 1']

         score = metrics.accuracy_score(y_test,predictions[:,1]>0.5)
         print ("Accuracy score:  {:.2%} \n".format(score))

         print ("Classification report: ")
         print(metrics.classification_report(y_test,predictions[:,1]>0.5, target_names=targets

         # Print out confusion matrix
         confusion_matrix = metrics.confusion_matrix(y_test,predictions[:,1]>0.5)
         print ('Confusion_matrix: \n', confusion_matrix)
         show_confusion_matrix(confusion_matrix, targets)
```

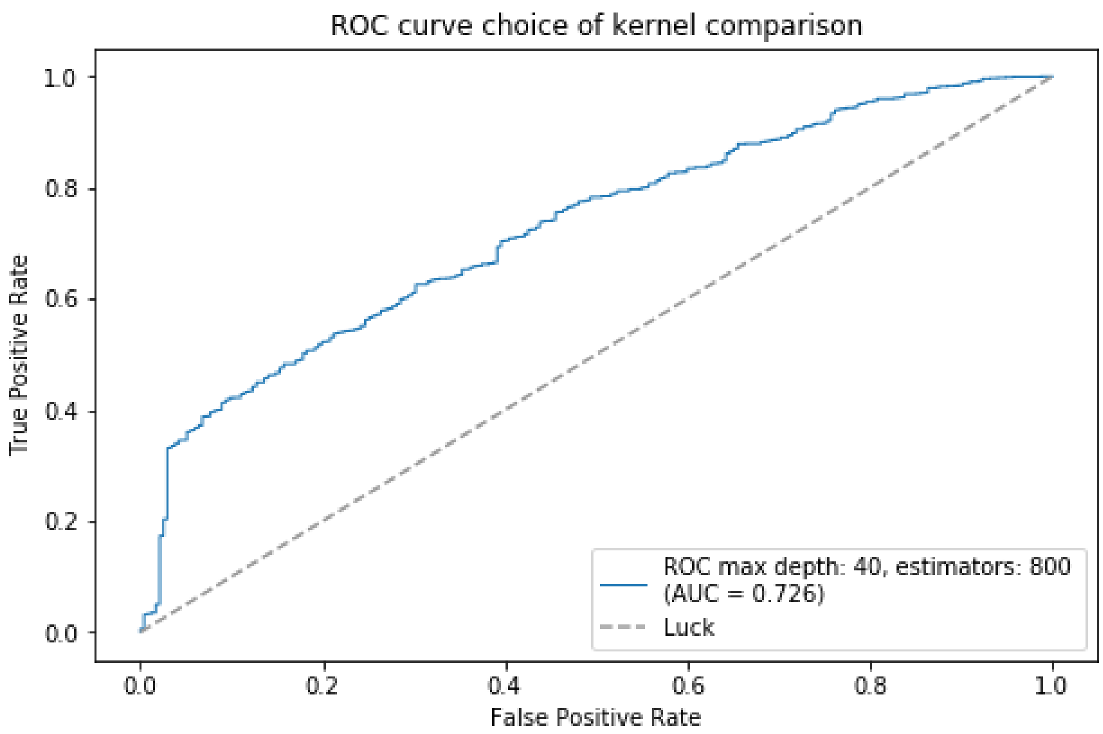
```
Accuracy score:  84.14%

Classification report:
              precision    recall  f1-score   support

    Output 0       0.47      0.15      0.23       235
    Output 1       0.86      0.97      0.91      1278

   micro avg       0.84      0.84      0.84      1513
   macro avg       0.66      0.56      0.57      1513
weighted avg       0.80      0.84      0.81      1513

Confusion_matrix:
 [[  35  200]
 [  40 1238]]
```



Generate ROC curve

```
In [45]: from sklearn.metrics import roc_curve, auc, precision_recall_curve, average_precision_
         from sklearn.metrics import confusion_matrix, precision_recall_fscore_support, accurac

In [46]: plt.figure(figsize=(8,5))
         random_state = np.random.RandomState(37)
         mean_tpr = 0.0
         mean_fpr = np.linspace(0, 1, 100)
         all_tpr = []


         probas_ = gbm.fit(X_train_1,y_train).predict_proba(X_test_1)
         # Compute ROC curve and area the curve
         fpr, tpr, thresholds = roc_curve(y_test, probas_[:, 1])
         mean_tpr += np.interp(mean_fpr, fpr, tpr)
         mean_tpr[0] = 0.0
         roc_auc = auc(fpr, tpr)
         plt.plot(fpr, tpr, lw=1, label='ROC max depth: %d, estimators: %d \n(AUC = %0.3f)' %
```

```
plt.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6), label='Luck')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve choice of kernel comparison')
plt.legend(loc="lower right")
plt.show()
```



Try individual XGBoost

In [47]:
```
import xgboost as xgb
gbm_1 = xgb.XGBClassifier(seed=101, n_estimators=1100 , max_depth=3, colsample_byleve
                           colsample_bytree=0.7,learning_rate=0.01, reg_lambda=0.1 ,
                            scale_pos_weight = 0.18357862) #missing = -999
```

In [48]:
```
sensor_train_1.shape
```

Out[48]:
```
(4584, 676)
```

In [49]:
```
sensor_test_1.shape
```

```
Out[49]: (1732, 676)

In [50]: gbm_1.fit(sensor_train_1,Y, eval_metric='auc')

Out[50]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.8,
               colsample_bytree=0.7, gamma=0, learning_rate=0.01, max_delta_step=0,
               max_depth=3, min_child_weight=1, missing=None, n_estimators=1100,
               n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
               reg_alpha=0, reg_lambda=0.1, scale_pos_weight=0.18357862, seed=101,
               silent=True, subsample=1)

In [51]: predictions_1 = gbm_1.predict(sensor_test_1)
```

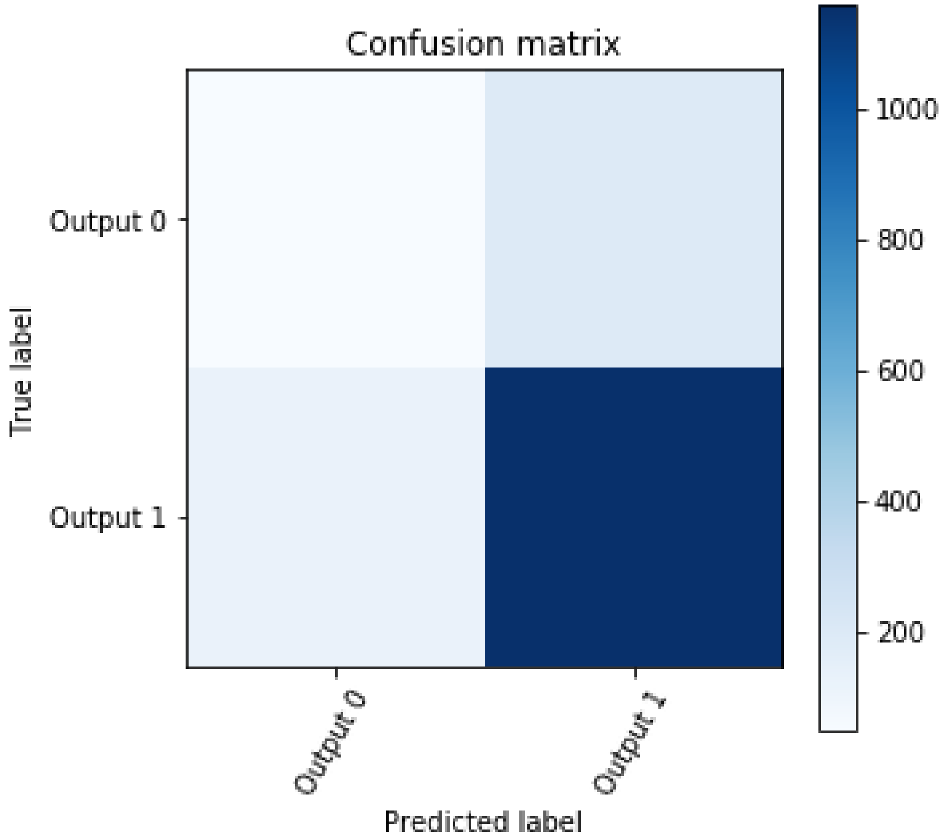
Try individual RF

```
In [52]: clf_1 = RandomForestClassifier(n_estimators=800, max_depth=40, random_state=101,class_
         clf_1.fit(X_train,y_train)

Out[52]: RandomForestClassifier(bootstrap=True, class_weight={0: 5, 1: 1},
                     criterion='gini', max_depth=40, max_features='auto',
                     max_leaf_nodes=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=800, n_jobs=None, oob_score=False,
                     random_state=101, verbose=0, warm_start=False)
```

Create confusion matrix for RF

```
In [53]: targets =['Output 0','Output 1']

         score = metrics.accuracy_score(y_test,clf_1.predict(X_test))
         print ("Accuracy score:  {:.2%} \n".format(score))

         print ("Classification report: ")
         print(metrics.classification_report(y_test,clf_1.predict(X_test), target_names=targets

         # Print out confusion matrix
         confusion_matrix = metrics.confusion_matrix(y_test,clf_1.predict(X_test))
         print ('Confusion_matrix: \n', confusion_matrix)
         show_confusion_matrix(confusion_matrix, targets)

Accuracy score:  79.64%

Classification report:
              precision    recall  f1-score   support

    Output 0       0.28      0.20      0.24       235
    Output 1       0.86      0.91      0.88      1278
```

```
   micro avg       0.80      0.80      0.80      1513
   macro avg       0.57      0.55      0.56      1513
weighted avg       0.77      0.80      0.78      1513

Confusion_matrix:
 [[  48  187]
 [ 121 1157]]
```



In [54]:
```
#X_pred = sensor_test_1.values

#Y_pred= gbm_1.predict(X_pred)

df_Y_pred = pd.DataFrame(data=predictions_1, columns=['output'])
```

In [55]:
```
accuracy_score(Y, gbm_1.predict(sensor_train_1))
```

Out[55]: 0.73036649214659688

In [56]:
```
url2 = 'C:\\Users\\pinakibhagat\\Downloads\\Personal\\Harvard\\CSCIE-82\\Homework 4\\a
df_Y_pred.to_csv(url2, sep=',')
```