Model_final

August 5, 2018

```
In [6]: import dask
        from dask.distributed import Client
        %matplotlib inline

In [7]: client = Client() # launch local dask.distributed client

In [8]: client.cluster
        #Click on Dashboard to visualise
```

```
A Jupyter Widget
```

```
In [9]: import numpy as np
        import os
        import pandas as pd
        import dill
        import matplotlib.pyplot as plt
        import seaborn as sns
        import itertools
        from datetime import timedelta
        from dateutil.parser import parser

        from sklearn.preprocessing import Normalizer
        from sklearn.model_selection import StratifiedShuffleSplit
        from sklearn.metrics import confusion_matrix, f1_score, roc_auc_score, roc_curve
        from sklearn.metrics import auc, accuracy_score, precision_recall_curve
        from sklearn.metrics import precision_score, recall_score

        from imblearn.combine import SMOTETomek
        from imblearn.pipeline import Pipeline
        from scipy import interp
        import matplotlib.pyplot as plt
        from itertools import cycle
        from sklearn.preprocessing import RobustScaler
        from sklearn.model_selection import StratifiedKFold
        from tempfile import mkdtemp
        from shutil import rmtree
        from sklearn.decomposition import PCA
        from matplotlib import rcParams
        rcParams.update({'figure.autolayout': False})

In [10]: def plot_confusion_matrix(cm, classes,
                                  normalize=False,
                                  title='Confusion matrix',
                                  cmap=plt.cm.Blues):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             print(cm)
             fig = plt.figure(figsize=(9, 9))
             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)

             fmt = '.2f' if normalize else 'd'
             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, format(cm[i, j], fmt),
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.tight_layout()
             plt.ylabel('True label')
             plt.xlabel('Predicted label')
             plt.savefig(os.path.expanduser(r"./confusion_matrix.png"),
                         format='png', dpi=300, bbox_inches='tight')
             plt.close();


In [ ]: def plot_cross_validation(cv, X, y, pipeline):
            tprs = [];
            aucs = []
            mean_fpr = np.linspace(0, 1, 100)
            i = 0
            for train, test in cv.split(X, y):
```

```
            probas_= pipeline.fit(X[train], y[train]).predict_proba(X[test])
                # Compute ROC curve and area the curve
            fpr, tpr, thresholds = roc_curve(y[test], probas_[:, 1])
            tprs.append(interp(mean_fpr, fpr, tpr))
            tprs[-1][0] = 0.0
            roc_auc = auc(fpr, tpr)
            aucs.append(roc_auc)
            plt.plot(fpr, tpr, lw=1, alpha=0.3,
            label='ROC fold %d (AUC = %0.2f)' % (i, roc_auc))
            i += 1
        plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',label='Luck', alpha=.8)

        mean_tpr = np.mean(tprs, axis=0)
        mean_tpr[-1] = 1.0
        mean_auc = auc(mean_fpr, mean_tpr)
        std_auc = np.std(aucs)
        plt.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

        std_tpr = np.std(tprs, axis=0)
        tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
        tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
        plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

        plt.xlim([-0.05, 1.05])
        plt.ylim([-0.05, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        #plt.tight_layout()
        plt.title('ROC with stratified 5-fold cross validation')
        plt.legend(loc="lower right")
        plt.savefig(os.path.expanduser("./ROC_stratified_kfold.png"),
                    format='png',dpi=300);
        plt.close();
```

```
In [11]: from sklearn.ensemble import AdaBoostClassifier
         from sklearn.tree import DecisionTreeClassifier
         ##############
         from dask_ml.model_selection import GridSearchCV as dasksearchCV # GridSearchCV
         ##############
         from xgboost import XGBClassifier
         def main():
             import time
             start = time.time()
             with open('./pkl/X.pkl', 'rb') as fh: # Load data set
```

```
        X = dill.load(fh)
with open('./pkl/y.pkl', 'rb') as fh:
        y = dill.load(fh)
scaler = Normalizer()
smote_etomek=SMOTETomek(ratio='auto')
cachedir = mkdtemp()
cv = StratifiedKFold(n_splits=5,shuffle=True)
classifier = XGBClassifier()

# A parameter grid for XGBoost
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0, 0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [1, 3, 4, 5, 10],
        }
pipeline = Pipeline([('scaler',scaler),('smt', smote_etomek),
                     ('clf',classifier),],memory=cachedir)
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_state=0)
sss.get_n_splits(X, y)
for train_index, test_index in sss.split(X, y):
    #print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index] # make training and test set
    y_train, y_test = y[train_index], y[test_index]

    clf = dasksearchCV(classifier, params, n_jobs=8,
           cv=3,
           scoring='roc_auc',
            refit=True)

    clf.fit(X_train, y_train)
    print(clf.best_params_)
    print(clf.best_score_)
    best_parameters, score = clf.best_params_, clf.best_score_
    print('Raw AUC score:', score)
    for param_name in sorted(best_parameters.keys()):
        print("%s: %r" % (param_name, best_parameters[param_name]))
    classifier = XGBClassifier(**best_parameters,njobs=-1)
    plot_cross_validation(cv, X_train, y_train, pipeline) # do 5 fold stratified
    clf = pipeline.fit(X_train, y_train) #

    print(classifier.get_params())
    expected = y_test
    predicted = clf.predict(X_test) # test performance on test set
    plot_confusion_matrix(confusion_matrix(expected, predicted),classes = ["Non-Z
print(time.time()- start)
from sklearn import metrics
```

```
            print("Classification report for classifier %s:\n%s\n"
                      % (clf, metrics.classification_report(expected, predicted)))

        if __name__ == '__main__':
            main()
```

```
{'colsample_bytree': 0.8, 'gamma': 0, 'max_depth': 5, 'min_child_weight': 1, 'subsample': 0.8}
0.9872513524030729
Raw AUC score: 0.9872513524030729
colsample_bytree: 0.8
gamma: 0
max_depth: 5
min_child_weight: 1
subsample: 0.8
{'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bytree': 0.8, 'gamm
Confusion matrix, without normalization
[[ 50  13]
 [ 10 291]]


/home/abhijit/anaconda3/envs/idp/lib/python3.6/site-packages/sklearn/preprocessing/label.py:15
  if diff:


26.95408058166504
Classification report for classifier Pipeline(memory='/tmp/tmp4t26thuu',
     steps=[('scaler', Normalizer(copy=True, norm='l2')), ('smt', SMOTETomek(k=None, kind_smot
      random_state=None, ratio='auto', smote=None, tomek=None)), ('clf', XGBClassifier(base_sc
       colsample_bytree=1...
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1))]):
             precision    recall  f1-score   support

          0       0.83      0.79      0.81        63
          1       0.96      0.97      0.96       301

avg / total       0.94      0.94      0.94       364
```

In [ ]: