# Document the work

The idea of this file is the replace the README.md in terms of documenting the work and leaving it for files index in the repository.

In this document I will follow the questions suggested and I will use “#x” numbers reference to sections in the script (poi\_id.py).

***Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.***

***As part of your answer, give some background on the dataset and how it can be used to answer the project question.***

My first objective is to get some info on the Enron case and understand what have happened to be able

to look at the available data with the right context. To do that I watched, as recommended, the TV documentary :

"Enron : the smartest guys in the rooms" . Based on that, I could conclude the following statements about

my comprehension to help me start writing some hypothesis and some answers :

1) The fraud consisted in "cooking" the accounting books to keep stock prices growing

2) Executives profits were mostly made out of sealing those overpriced stock, exercise their stock options

3) Benefited executives were those who executed those stocks at higher prices -before the crash-

4) We have available 4 features that are directly related with item #3 which are :

total\_stock\_value

exercised\_stock\_options

restricted\_stock\_deferred

restricted\_stock

where total\_stock\_value = exercised\_stock\_option + (restricted\_stock - restricted\_stock\_deferred)

Based on this I'm planning to focus on financial features correlating payments based features like ‘salary’ and ‘bonus’ with those stock features above mentioned. Given the fact that exercised\_stock\_options + restricted\_stock are actually summed up to account for the total ‘cash’ made based on stock by the people that has information on stock value , I will end up creating a new feature named ‘cash\_from\_stock’ to collect that.

I will also will create a new feature to represent or signal the people which are part of the 80% of exercised stock options . The idea is t use it as a mask to help the model.

Lastly, I created a feature to collect other “advanced cash” features and see if it gives a additional information

my\_magic\_dataset['advanced\_cash'] = my\_magic\_dataset['loan\_advances'] + my\_magic\_dataset['other'] + my\_magic\_dataset['expenses']

Machine learning techniques are very useful in this case basically because we have a dataset that includes features and labels (poi or non poi); our objective is to predict, based on some features values, if the sample (enron employee in this case) can be categorized as poi or non poi. For that reason, since we have data already categorized, our problem is a binary (poi / nor poi), classification (discrete, no regression), supervised (train set with features and labels) machine learning problem.

***Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: “data exploration”, “outlier investigation”]***

The actual code for this question and answers is poi\_id.py Section #1

Lets load and explore the dataset.

Let start with some of the datasets characteristics.

Total number of data points : 146

Allocation between the POI and non-POI: POI = 18, non-POI = 128

Total number of features: 21

There are missing values in the features ?: Yes, there are ‘NaN’ and/or missing values

Which features contains missing values: Most of the features has missing values. I used the pandas ‘isnull’ function and returned True for all of the features but ‘poi’ and ‘email\_address’. Cross checking looking at some data I found that original NaN in email\_address were treated incorrectly so only the ‘poi’ feature has complete information-

Features with missing values: ['salary', 'to\_messages', 'deferral\_payments', 'total\_payments', 'exercised\_stock\_options', 'bonus', 'restricted\_stock', 'shared\_receipt\_with\_poi', 'restricted\_stock\_deferred', 'total\_stock\_value', 'expenses', 'loan\_advances', 'from\_messages', 'other', 'from\_this\_person\_to\_poi', 'director\_fees', 'deferred\_income', 'long\_term\_incentive', 'from\_poi\_to\_this\_person']

Features with no missing values: ['poi', 'email\_address']

Before working on outliers identification I will drop all non-financial features from the dataset, drop the ‘TOTAL’ data point (summarization) and drop all data points that contains all ‘NaN’.

To check for outliers we will look at pandas describe() statistics and will compute if max() values in the features are 2 \* std from the mean.

Almost all features has outliers. During the test of different algorithms I used StandardScaler and RobustScaler to deal with the outliers.

Lastly, I will fill ‘NaN’ with 0 (zero) .

***What features did you end up using in your POI identifier, and what selection process did you use to pick them?***

***- explain what feature you tried to make, and the rationale behind it.***

***In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest,***

***please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “intelligently select features”, “properly scale features”]***

The actual code for this question and answers is poi\_id.py Section #2

Now I have the following features in the cleaned datasets

Features remaining after cleaning mail: ['salary' 'deferral\_payments' 'total\_payments' 'exercised\_stock\_options'

'bonus' 'restricted\_stock' 'restricted\_stock\_deferred' 'total\_stock\_value'

'expenses' 'loan\_advances' 'other' 'poi' 'director\_fees' 'deferred\_income'

'long\_term\_incentive']

NaNs were filled with 0.

For the feature selection process I followed the following steps:

1. I trained a RandomForestClassifier with the initial list of features after cleaning and, using the feature\_importances\_ of the classifier I printed out an ordered list of features importance (see Step 2.a in the script. That give me the following list :

This is the ranking of the initial features

exercised\_stock\_options 0.177469

total\_stock\_value 0.135812

restricted\_stock 0.101740

other 0.098535

deferred\_income 0.092841

expenses 0.089134

bonus 0.083820

total\_payments 0.062447

salary 0.056872

long\_term\_incentive 0.053250

deferral\_payments 0.028490

loan\_advances 0.014323

restricted\_stock\_deferred 0.005267

director\_fees 0.000000

2. Then I trained a RandomForestClassifier with the modified list of features after creating the three new onec and, using the feature\_importances\_ of the classifier I printed out an ordered list of features importance (see Step 2.b in the script). That give me the following list :

This is the ranking of the new list of features including new ones

cash\_from\_stock 0.142161

exercised\_stock\_options 0.136512

total\_stock\_value 0.107990

deferred\_income 0.104803

bonus 0.083068

restricted\_stock 0.076103

advanced\_cash 0.064932

salary 0.061764

expenses 0.050952

long\_term\_incentive 0.050088

other 0.049380

total\_payments 0.042720

deferral\_payments 0.015475

loan\_advances 0.007726

high\_exercised\_percentile 0.004088

restricted\_stock\_deferred 0.002238

director\_fees 0.000000

3. Given the fact that , in the code provided on review#4 , the best PCA was running with n\_components=6 I tried two different feature lists based on the ranking provided by the classifier.

So, the first list I tried was (See Step 2.c in the script) :

initial\_top\_six = ['poi','exercised\_stock\_options', 'total\_stock\_value', 'restricted\_stock', 'other', 'deferred\_income', 'expenses']

Based on that, and based on the algorithm selected (see section below) I tested this initial list with the algorithm , GridSearchCV and pipe found on Section 3 of the script completing the test pasing this list, the new dataset and the classifier to the tester.py The results where :

Best Linear SVC params: {'linearsvc\_\_C': 0.01, 'linearsvc\_\_tol': 0.001}

Best Linear SVC cv score: 0.770555555556

and the tester.py results in :

Accuracy: 0.77673 Precision: 0.26765 Recall: 0.38850 F1: 0.31695 F2: 0.35632

Total predictions: 15000 True positives: 777 False positives: 2126 False negatives: 1223 True negatives: 10874

After that, I changed the top 6 list , from initial\_top\_six to magic\_top\_six and the results where :

Best Linear SVC params: {'linearsvc\_\_C': 0.01, 'linearsvc\_\_tol': 0.001}

Best Linear SVC cv score: 0.78884057971

Accuracy: 0.82000 Precision: 0.38475 Recall: 0.43400 F1: 0.40789 F2: 0.42317

Total predictions: 14000 True positives: 868 False positives: 1388 False negatives: 1132 True negatives: 10612

Conclusion:

The magic\_top\_six combination of features , ordered with the help of the feature\_importances\_ attribute of the RandomForestClassifier help me improve accuracy, precision and recall.

The magic\_top\_six only hav one new feature listed (cash\_from\_stock) so the other 2 new features were not used.

***Did you have to do any scaling? Why or why not?***

For the RandonForestClassifier I did not do scaling since I learned from the following sources that it is usually not required.

see : <http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html>

see : <https://stackoverflow.com/questions/8961586/do-i-need-to-normalize-or-scale-data-for-randomforest-r-package>

For SVC and LinearSVC i use RobustScaler because as we saw there are outliers present in most of the features selected so the RobustScaler is usefull because it implements a scaling based on statistical analysis

I end up using StandardScaler on LinearSVC because with the inclusion in the pipeline of the Inputer the StandardScaler end up providing the last move up of the precission over 0.30as required. I guess that it is related with the backing algorithm tha manage the predomination of missing data converted to 0 values.

***What algorithm did you end up using? What other one(s) did you try?***

***How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]***

The actual code for this question and answers is poi\_id.py Section #3

I have reviewed LinearSVC, SVC and RandomForestClassifier and GradientBoostingClassifier. (I provided backup information of some of the tests with the other classifiers.)

I end up using LinearSVC because it was the one that provided more balance between precision and recall results. The ensemble algorithms use to result in more precision (> 50 and 60) but I was not able to improve the recall on them.

***What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well?***

***How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: “discuss parameter tuning”, “tune the algorithm”]***

Tunning parameters means to do as most as combinations on most important algorithms parameters as computational resources permit to obtain the best combination of them which is always a hot spot between over fitting and under fitting.

Since LinearSVC provided the most balanced results between precession and recall I started to review different preprocessing with different parameters given the ability of GridSearchCV to do the combinations.

For Linear SVC I tried different options of 'C' and 'tol', I tried different options of 'tol' for PCA and tried different values of n\_components for PCA. When I changed 'svd\_solver' from the default to arparck precision started to move up until I tested n\_components over 6 where precision started to fall. So, at that point I started to fix the parameters that worked and move to other options. 'penalty' = 'l1' used to gave me 'divided by 0' errors so I keep using the 'l2' default. class\_weight = balanced provided better results than default and I read in the documentation that given the dataset proportions of features with the label dual=False provided better results.

After feature selection process, based on the initial PCA results, I end up not using the PCA but using it results to define a number of features to test (6) and it works well because it provides me better results.

***What is validation, and what’s a classic mistake you can make if you do it wrong?***

***How did you validate your analysis? [relevant rubric items: “discuss validation”, “validation strategy”]***

Validation is the process of using a separated test data set to validate predictions on the training data set. A classic mistake is to validate the results on the training test which wont provide a glue on how the system will generalize when new data comes or also to select test data that is not representative of the training set. Our dataset is not well balanced in terms of lots of missing data and a class label with a small number of cases (not balanced, only 18 data points over 146.

To validate our analysis I used the recommended StratifiedShuffleSplit class that shuffle the dataset n times and combine test and train data.

***Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm’s performance. [relevant rubric item: “usage of evaluation metrics”]***

For every algorithm execution I checked 4 values -provided by the tester.py- accuracy, precision, recall.

Predictions are true negatives and true positives ; accuracy represents the number of correct predictions (true negatives + true positives) divided by number of all samples.

There is a trade off between precision and recall because of the nature (see below) of them so it is important to understand that it is important to know in advance the objetive of the model to seeif we should focus in precision penalization looking for better recall or viceverse.

First is important to understand:

False positive : predicted positive (POI) but it was in the negative class (NON POI) ('false alarm')

False negative : predicted negative (NON POI) but it was in the positive class (POI) (lost an alarm)

Given that, precision and recall objectives are:

Precision : limit the number of false positives (meassures how many of the samples predicted as positives are actually positives)

Recall : limit the number of false negatives (meassures the true positive rate o how many of the positive samples are captured by the positive predictions).

So, if 'False Negatives' grows, recall is affected, if 'false positives' grows, precision if affected. I our case , since we are identifying people that we wont conclude were involved in the fraud, but are important in the investigation, my conclusion is that is better to limit the number of false negatives ( or we will be losing someone else to further investigate) or improved recall.

## Backup information.

***RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',***

***max\_depth=2, max\_features=None, 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=50, n\_jobs=1,***

***oob\_score=False, random\_state=0, verbose=0, warm\_start=False)***

***Accuracy: 0.85333 Precision: 0.31618 Recall: 0.08600 F1: 0.13522 F2: 0.10066***

***Total predictions: 15000 True positives: 172 False positives: 372 False negatives: 1828 True negatives: 12628***

***keeping all the features***

***RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',***

***max\_depth=8, max\_features=None, 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=10, n\_jobs=1,***

***oob\_score=False, random\_state=0, verbose=0, warm\_start=False)***

***Accuracy: 0.85140 Precision: 0.38135 Recall: 0.18400 F1: 0.24823 F2: 0.20524***

***Total predictions: 15000 True positives: 368 False positives: 597 False negatives: 1632 True negatives: 12403***

***LinearSVC(C=0.1, class\_weight=None, dual=True, fit\_intercept=True,***

***intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.0001,***

***verbose=0)***

***Accuracy: 0.74000 Precision: 0.20442 Recall: 0.32850 F1: 0.25201 F2: 0.29294***

***Total predictions: 15000 True positives: 657 False positives: 2557 False negatives: 1343 True negatives: 10443***

***LinearSVC(C=0.1, class\_weight=None, dual=False, fit\_intercept=True,***

***intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,***

***multi\_class='ovr', penalty='l1', random\_state=None, tol=0.0001,***

***verbose=0)***

***Accuracy: 0.85947 Precision: 0.42082 Recall: 0.14350 F1: 0.21402 F2: 0.16528***

***Total predictions: 15000 True positives: 287 False positives: 395 False negatives: 1713 True negatives: 12605***

***LinearSVC(C=1, class\_weight=None, dual=False, fit\_intercept=True,***

***intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,***

***multi\_class='ovr', penalty='l1', random\_state=None, tol=0.0001,***

***verbose=0)***

***Accuracy: 0.86713 Precision: 0.50409 Recall: 0.21550 F1: 0.30193 F2: 0.24337***

***Total predictions: 15000 True positives: 431 False positives: 424 False negatives: 1569 True negatives: 12576***

***Pipeline(memory=None,***

***steps=[('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_centering=True,***

***with\_scaling=True)), ('linearsvc', LinearSVC(C=0.1, class\_weight=None, dual=False, fit\_intercept=True,***

***intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,***

***multi\_class='ovr', penalty='l1', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.87060 Precision: 0.53897 Recall: 0.20400 F1: 0.29597 F2: 0.23296***

***Total predictions: 15000 True positives: 408 False positives: 349 False negatives: 1592 True negatives: 12651***

***labels has this shape: (145,)***

***features has this shape: (145, 14)***

***exercised\_stock\_options 0.177469***

***restricted\_stock 0.101740***

***total\_stock\_value 0.135812***

***dtype: float64***

***labels has this shape: (145,)***

***features has this shape: (145, 15)***

***exercised\_stock\_options 0.124175***

***total\_stock\_value 0.137300***

***cash\_from\_stock 0.133830***

***dtype: float64***

***Best Forest params: {'n\_estimators': 10, 'max\_depth': 8, 'min\_samples\_leaf': 1}***

***Best Forest cv score: 0.886206896552***

***Best Linear SVC params: {'linearsvc\_\_C': 0.1, 'linearsvc\_\_tol': 0.0001}***

***Best Linear SVC cv score: 0.844827586207***

***tano@debtano:~/mlearning\_enron$ python tester.py***

***/home/tano/anaconda2/lib/python2.7/site-packages/sklearn/cross\_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.***

***"This module will be removed in 0.20.", DeprecationWarning)***

***Pipeline(memory=None,***

***steps=[('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_centering=True,***

***with\_scaling=True)), ('linearsvc', LinearSVC(C=0.1, class\_weight=None, dual=True, fit\_intercept=True,***

***intercept\_scaling=1, loss='squared\_hinge', max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.0001,***

***verbose=0))])***

***Accuracy: 0.79087 Precision: 0.24677 Recall: 0.27700 F1: 0.26101 F2: 0.27038***

***Total predictions: 15000 True positives: 554 False positives: 1691 False negatives: 1446 True negatives: 11***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.1, loss='deviance', max\_depth=1,***

***max\_features=None, 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=20,***

***presort='auto', random\_state=0, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.87547 Precision: 0.63924 Recall: 0.15150 F1: 0.24495 F2: 0.17878***

***Total predictions: 15000 True positives: 303 False positives: 171 False negatives: 1697 True negatives: 12829***

***without cash\_from\_stock***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.1, loss='deviance', max\_depth=1,***

***max\_features=None, 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=20,***

***presort='auto', random\_state=0, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.87560 Precision: 0.64017 Recall: 0.15300 F1: 0.24697 F2: 0.18047***

***Total predictions: 15000 True positives: 306 False positives: 172 False negatives: 1694 True negatives: 12828***

***without cash\_from\_stock and high\_exercised\_percentile***

***droping non\_required\_features = ['exercised\_stock\_options','restricted\_stock','total\_stock\_value'] but keeping cash and percentile***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.1, loss='deviance', max\_depth=1,***

***max\_features=None, 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=50,***

***presort='auto', random\_state=0, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.85900 Precision: 0.38523 Recall: 0.09650 F1: 0.15434 F2: 0.11352***

***Total predictions: 15000 True positives: 193 False positives: 308 False negatives: 1807 True negatives: 12692***

***max\_features = auto, all the features***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.1, loss='deviance', max\_depth=1,***

***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=20,***

***presort='auto', random\_state=0, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.87547 Precision: 0.63924 Recall: 0.15150 F1: 0.24495 F2: 0.17878***

***Total predictions: 15000 True positives: 303 False positives: 171 False negatives: 1697 True negatives: 12829***

***improve number of estimators only 50-100 option***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.1, loss='deviance', max\_depth=1,***

***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=50,***

***presort='auto', random\_state=None, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.86873 Precision: 0.52256 Recall: 0.17950 F1: 0.26721 F2: 0.20663***

***Total predictions: 15000 True positives: 359 False positives: 328 False negatives: 1641 True negatives: 12672***

***200***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.01, loss='deviance', max\_depth=1,***

***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=200,***

***presort='auto', random\_state=None, subsample=1.0, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.87713 Precision: 0.67028 Recall: 0.15450 F1: 0.25112 F2: 0.18260***

***Total predictions: 15000 True positives: 309 False positives: 152 False negatives: 1691 True negatives: 12848***

***max\_features='sqrt', subsample=0.8, max\_depth 4-8***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.01, loss='deviance', max\_depth=4,***

***max\_features='sqrt', 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=200,***

***presort='auto', random\_state=None, subsample=0.8, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.86433 Precision: 0.46479 Recall: 0.11550 F1: 0.18502 F2: 0.13593***

***Total predictions: 15000 True positives: 231 False positives: 266 False negatives: 1769 True negatives: 12734***

***GradientBoostingClassifier(criterion='friedman\_mse', init=None,***

***learning\_rate=0.01, loss='deviance', max\_depth=2,***

***max\_features=None, max\_leaf\_nodes=None,***

***min\_impurity\_decrease=0.0, min\_impurity\_split=None,***

***min\_samples\_leaf=1, min\_samples\_split=100,***

***min\_weight\_fraction\_leaf=0.0, n\_estimators=100,***

***presort='auto', random\_state=10, subsample=0.8, verbose=0,***

***warm\_start=False)***

***Accuracy: 0.87473 Precision: 0.70790 Recall: 0.10300 F1: 0.17983 F2: 0.12423***

***Total predictions: 15000 True positives: 206 False positives: 85 False negatives: 1794 True negatives: 12915***

trees

***Accuracy: 0.84320 Precision: 0.22923 Recall: 0.07450 F1: 0.11245 F2: 0.08613***

***Accuracy: 0.84300 Precision: 0.22481 Recall: 0.07250 F1: 0.10964 F2: 0.08386***

***forest***

***Accuracy: 0.77840 Precision: 0.24120 Recall: 0.30850 F1: 0.27073 F2: 0.29220***

***Total predictions: 15000 True positives: 617 False positives: 1941 False negatives: 1383 True negatives: 11059***

***Accuracy: 0.73493 Precision: 0.23498 Recall: 0.43800 F1: 0.30587 F2: 0.37347***

***Total predictions: 15000 True positives: 876 False positives: 2852 False negatives: 1124 True negatives: 10148***

***Accuracy: 0.80113 Precision: 0.25264 Recall: 0.25100 F1: 0.25182 F2: 0.25133***

***Total predictions: 15000 True positives: 502 False positives: 1485 False negatives: 1498 True negatives: 11515***

PCA n\_components = 0.80

***Accuracy: 0.82500 Precision: 0.30505 Recall: 0.24450 F1: 0.27144 F2: 0.25461***

***Total predictions: 15000 True positives: 489 False positives: 1114 False negatives: 1511 True negatives: 11886***

***PCA n\_components = 0.70***

***Accuracy: 0.81400 Precision: 0.28225 Recall: 0.25600 F1: 0.26848 F2: 0.26085***

***Total predictions: 15000 True positives: 512 False positives: 1302 False negatives: 1488 True negatives: 11698***

***0.75***

***Accuracy: 0.81820 Precision: 0.28705 Recall: 0.24500 F1: 0.26436 F2: 0.25240***

***Total predictions: 15000 True positives: 490 False positives: 1217 False negatives: 1510 True negatives: 11783***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=0.8, random\_state=None,***

***svd\_solver='auto', tol=0.0, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.78813 Precision: 0.28234 Recall: 0.38200 F1: 0.32469 F2: 0.35681***

***Total predictions: 15000 True positives: 764 False positives: 1942 False negatives: 1236 True negatives: 11058***

***Best Linear SVC params: {'linearsvc\_\_C': 10, 'linearsvc\_\_tol': 0.001}***

***Best Linear SVC cv score: 0.785694444444***

***tano@debtano:~/mlearning\_enron$ python tester.py***

***/home/tano/anaconda2/lib/python2.7/site-packages/sklearn/cross\_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.***

***"This module will be removed in 0.20.", DeprecationWarning)***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=0.8, random\_state=None,***

***svd\_solver='auto', tol=0.0, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.78813 Precision: 0.28234 Recall: 0.38200 F1: 0.32469 F2: 0.35681***

***Total predictions: 15000 True positives: 764 False positives: 1942 False negatives: 1236 True negatives: 11058***

***n\_components=1, svd\_solver='arpack'***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=1, random\_state=None,***

***svd\_solver='arpack', tol=0.0, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.78860 Precision: 0.28307 Recall: 0.38200 F1: 0.32518 F2: 0.35704***

***Total predictions: 15000 True positives: 764 False positives: 1935 False negatives: 1236 True negatives: 11065***

n\_components=4

Pipeline(memory=None,

steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=4, random\_state=None,

svd\_solver='arpack', tol=0.0, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...ax\_iter=1000,

multi\_class='ovr', penalty='l2', random\_state=None, tol=0.0001,

verbose=0))])

Accuracy: 0.78287 Precision: 0.28673 Recall: 0.42250 F1: 0.34162 F2: 0.38595

Total predictions: 15000 True positives: 845 False positives: 2102 False negatives: 1155 True negatives: 10898

n\_components = 6

Pipeline(memory=None,

steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=6, random\_state=None,

svd\_solver='arpack', tol=0.0, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...max\_iter=1000,

multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,

verbose=0))])

Accuracy: 0.78540 Precision: 0.28976 Recall: 0.42000 F1: 0.34293 F2: 0.38536

Total predictions: 15000 True positives: 840 False positives: 2059 False negatives: 1160 True negatives: 10941

***Best Linear SVC params: {'linearsvc\_\_C': 1, 'linearsvc\_\_tol': 0.001, 'pca\_\_tol': 0.001}***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=6, random\_state=42,***

***svd\_solver='arpack', tol=0.001, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('robustscaler', RobustScaler(copy=True, quantile\_range=(25.0, 75.0), with\_ce...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.78540 Precision: 0.28976 Recall: 0.42000 F1: 0.34293 F2: 0.38536***

***Total predictions: 15000 True positives: 840 False positives: 2059 False negatives: 1160 True negatives: 10941***

***changed RobustScaler by StandardScaler***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=6, random\_state=42,***

***svd\_solver='arpack', tol=0.001, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('standardscaler', StandardScaler(copy=True, with\_mean=True, with\_std=True)),...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.78500 Precision: 0.29328 Recall: 0.43450 F1: 0.35019 F2: 0.39633***

***Total predictions: 15000 True positives: 869 False positives: 2094 False negatives: 1131 True negatives: 10906***

***Pipeline(memory=None,***

***steps=[('pca', PCA(copy=True, iterated\_power='auto', n\_components=6, random\_state=42,***

***svd\_solver='arpack', tol=0.001, whiten=False)), ('imputer', Imputer(axis=0, copy=True, missing\_values=0, strategy='mean', verbose=0)), ('standardscaler', StandardScaler(copy=True, with\_mean=True, with\_std=True)),...max\_iter=1000,***

***multi\_class='ovr', penalty='l2', random\_state=None, tol=0.001,***

***verbose=0))])***

***Accuracy: 0.79613 Precision: 0.30903 Recall: 0.42800 F1: 0.35891 F2: 0.39740***

***Total predictions: 15000 True positives: 856 False positives: 1914 False negatives: 1144 True negatives: 11086***