# Project report

This is the main document of the project. I will go through all the process in this document from data loading and cleaning to final model and testing. Please check this document alongside with the README.txt and the project script (poi\_id.py) to have a complete understanding of the process.

The poi\_id.py script has 3 Sections and Section #2 has steps. They we'll be referenced through this document.

## Project goal

The objective of the project is to build and test a machine learning model that can help us identify with some degree of accuracy , “Persons of interest” (POI) in the Enron case. To do that I wil be working with a dataset (final\_project\_dataset.pkl) with email and financial information from Enron.

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-

Based on that I decided that is possible to find out a working model using the financial related features provided in the dataset and concentrate my investigation around that hypothesis.

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.

## Dataset basic information and cleaning

*Please see poi\_id.py Section #1*

Let start reviewing some information about the dataset. To accomplish that I have converted the loaded dataset to a Panda's dataframe that provides support for better information retrieval and manipulation.

Total number of data points : 146

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

Total number of features: 21 (including 'poi', the label)

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']

\*actually 'poi' is the label in this case and not a feature

Regarding outliers, I will look at pandas describe() statistics and will compute if max() values in the features are 2 \* std from the mean. The script print out the information which confirm that almost all features has outliers. Based on that I will take care of include some preprocessing step during model creation to deal with them.

I also drop a 'TOTAL' data point from the dataset because it is a summarization and drop any data point that has all features with 'NaN' or missing values using Dataframe dropna function. Lastly, I will fill remaining ‘NaN’ with 0 (zero) .

## Dataset features – Selection and engineering

Based on the hypothesis mentioned in the first section I will base my analysis in the financial features of the dataset and I wont use the email information. I do not investigate the impact of disregard that information ; I am just treating the case as if I were not available in the first place to understand what results can be obtained from financial information only. The idea behind using the financial features is through correlating payments based features like ‘salary’ and ‘bonus’ with those stock features related with stocks or cash advances.

Based on what I mentioned, I first dropped the email features:

mail\_features = ['to\_messages','shared\_receipt\_with\_poi','from\_messages','from\_this\_person\_to\_poi','email\_address','from\_poi\_to\_this\_person']

my\_dataset = my\_dataset.drop(mail\_features, axis=1)

Lets work on 3 new features which will be defined as follows:

1. Intention of the new feature: exercised\_stock\_options and restricted\_stock, based on the information provided with the dataset, represents cash from stock obtained probably when the stock price were high before the crash. So I created a new feature called 'cash\_from\_stock' as :

my\_magic\_dataset['cash\_from\_stock'] = my\_magic\_dataset['exercised\_stock\_options'] + my\_magic\_dataset['restricted\_stock']

2. Intention of the new feature: since 'exercised\_stock\_options' is so important in my hiphotesys I will create a mask feature with 'True' or 1 when the value is included in the 80 quantile for the series or 'False' or 0 when is not. So I created a new feature called 'high\_exercised\_percentile' as:

water\_mark = my\_magic\_dataset['exercised\_stock\_options'].quantile(.80)

my\_magic\_dataset['high\_exercised\_percentile'] = my\_magic\_dataset['exercised\_stock\_options'] > water\_mark

my\_magic\_dataset['high\_exercised\_percentile'] = my\_magic\_dataset['high\_exercised\_percentile'].astype(np.int64)

3. Intention of the new feature: lastly I will sum up features that represent additional stock advanced from payments. So I created a new feature called 'advanced\_cash' as:

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

## Dataset features – Testing

*Please see poi\_id.py Section #2*

For the actual process of testing the dataset with the new features against the dataset without them I follow the following procedure:

1. Took the remaining features after dropping email data and separate features (initial\_features) from labels (poi). Split them, stratifying on labels, in training and and test set.

2. Train a RandomForestClassifier and print out an ordered list of feature\_importances\_

3. Copy the dataset used in 1 and 2 to a new dataset (my\_magic\_dataset).

4. Create the new 3 features mentioned in the previous section ('cash\_from\_stock', 'high\_exercised\_percentile' and 'advanced\_cash').

5. Extract new features (my\_magic\_features) and labels, split in a train and test set.

6. Train a RandomForestClassifier and print out an ordered list of feature\_importances\_

7. Compare results.

Here are the obtained results for feature\_importances\_ on the original dataset:

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

This is the ranking of the 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

Now, during previous runs with this new dataset (my\_magic\_dataset) I had run a LinearSVC classifier with GridSearchCV and a pipe including Imputer -to replace missing values- , a StandardScaler and a RobustScaler in different runs and PCA tested with different n\_components.

When I started to tested the n\_components parameter of PCA I found that it drops precision and/or recall under 6 but also reach its best results at 6 because after that I tested it with 7 and 8 and it started to drops again. Also, StandardScaler worked best that RobustScaler in this case.

The result of those runs were the following :

*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*

Based on that results and reasoning, I understood that PCA found , on the magic dataset, that the most promising dimensionality reduction was to 6 dimensions ; given that, my objective in Section #3 of the script is to compare the results using :

1. The dataset with the new features (my\_magic\_dataset) but reduce to the top 6 features found on the first RandomForestClassifier run :

exercised\_stock\_options 0.177469

total\_stock\_value 0.135812

restricted\_stock 0.101740

other 0.098535

deferred\_income 0.092841

expenses 0.089134

2. against the top 6 features found on the second RandomForestClassifier run.

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

So I created 2 lists of features (lines 194, 195) : initial\_top\_six (first list above, only names) and magic\_top\_six (second list, only names) adding ‘poi’ to both as first element since is required by the featureFormat utility.

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

magic\_top\_six = ['poi','cash\_from\_stock', 'exercised\_stock\_options', 'total\_stock\_value', 'deferred\_income', 'bonus', 'restricted\_stock']

At the beginning of section #3 in the script 2 things happen before moving to the actual algorithm:

1. my\_magic\_dataset is converted back from pandas dataframe to dict given back also its original name for convenience (my\_dataset, line 210)

2. my\_feature\_list could be assigned the initial\_top\_six or magic\_top\_six to test the algorithm results (line 214)

The script is pushed back to the repository with my\_features\_list = magic\_top\_six so the list of features in the final feature set is :

'poi','cash\_from\_stock', 'exercised\_stock\_options', 'total\_stock\_value', 'deferred\_income', 'bonus', 'restricted\_stock'

## Algorithm and parameters

*Please see poi\_id.py Section #3*

For the RandonForestClassifier I used for feature importance determination 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 checked 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 but 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 0.30 as required. I guess that it is related with the backing algorithm tha manage the predomination of missing data converted to 0 values.

I have reviewed LinearSVC, SVC and RandomForestClassifier and GradientBoostingClassifier.

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.

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.

## Validation and Evaluation

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.

For every algorithm execution I checked 3 values -provided by the tester.py- accuracy, precision, recall.

First, look at the results when my\_features\_list = initial\_top\_six . So, we are executing with the following list :

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

and the results were :

My features list in this test: ['poi', 'exercised\_stock\_options', 'total\_stock\_value', 'restricted\_stock', 'other', 'deferred\_income', 'expenses']

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

Best Linear SVC cv score: 0.770555555556

tano (master \*) ml\_enron\_reviewed $ python tester.py

/home/tano/dataAnalysis/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)), ('standardscaler', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('linearsvc', LinearSVC(C=0.01, class\_weight='balanced', dual=False, fit\_intercept=True,

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

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

verbose=0))])

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

then , I executed with my\_features\_list = magic\_top\_six so the features fot this set were:

magic\_top\_six = ['poi','cash\_from\_stock', 'exercised\_stock\_options', 'total\_stock\_value', 'deferred\_income', 'bonus', 'restricted\_stock']

and the results:

My features list in this test: ['poi', 'cash\_from\_stock', 'exercised\_stock\_options', 'total\_stock\_value', 'deferred\_income', 'bonus', 'restricted\_stock']

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

Best Linear SVC cv score: 0.78884057971

tano (master \*) ml\_enron\_reviewed $ python tester.py

/home/tano/dataAnalysis/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)), ('standardscaler', StandardScaler(copy=True, with\_mean=True, with\_std=True)), ('linearsvc', LinearSVC(C=0.01, class\_weight='balanced', dual=False, fit\_intercept=True,

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

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

verbose=0))])

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

As you can see Accuracy, Precision and Recall goes up in the second execution but we also conclude that 1 out of the 3 new features (cash\_from\_stock, present in the final feature set) helps improve the model.

But what represents those results in simpler terms ?

Accuracy represents the number of correct predictions ; it means that the model classified the data points or samples (persons records in this case from the Enron nomina) in the right class as ‘poi’ or ‘non poi’. In other terms, accuracy helps understand how close the results agree with the true value.

Now, given the fact that accuracy gives a “how close” meassure there are two other meassures that helps us understand the exactness of that “how close and how far” or basically how strictly close it was. And there it comes precision and recall.

Precision express strict exactness, how many of those data points predicted as ‘poi’ were actually classified as ‘poi’ and it gives a measure of how many samples were not exactly predicted or we have got a false alarm. Recall is similar in trying to express strict exactness to the accuracy value but in this case it gives a measure of how many samples were predicted as ‘non poi’ but it was actually a ‘poi’ so in this case, following the alarm example, we have lost an alarm.

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 see if we should focus in precision penalization looking for better recall or viceverse.

In our case , since we are trying to identify people (persons of interest) that we need to further investigate if they were involved in the fraud, my conclusion is that it is better to limit the number of lost alarms -because those were actually ‘poi’- or try to improve the recall.