# Machine Learning Project: Building a 'person of interest' classifier based on the Enron data set

## Project overview

The Enron scandal was one of the largest corporate fraud cases in American history, and led to the bankruptcy of Enron itself and the conviction of many of its former executives.

As a result of the scandal, considerable information about Enron executives fell into the public domain. This information included:

i) A list of executives who were thought to have been involved in the scandal (here defined as 'persons of interest', or POIs)[1]

ii) Information about the finances and email communications of Enron executives (both POIs and non-POIs)

This project aims to develop an algorithm to reliably predict if an individual is a POI as defined in (i) by making use of the attributes in (ii). More specifically, it aims to build a classifier that can determine, based on financial and communications data, whether a given Enron executive is likely to be a POI or not.

Machine learning is a useful tool in this task for at least two reasons:

1. The data set is rather large, particularly with regard to the communications data that covers thousands of email records. Machine learning algorithms are not only able to process large data sets quickly, but typically *benefit* from the presence of a large volume of data

2. The data contains clearly-defined 'features', or at least the ability to extract them. Machine learning is well suited to build classifiers on data structured in this way

## Data exploration

The data consists of a dictionary of 146 Enron executives. The keys are executive names (taking the form 'LASTNAME FIRSTNAME', with an optional 'MIDDLEINITIAL' at the end of the string if present). The values are themselves also dictionaries, detailing financial and email communications data for each individual, in addition to identifying whether or not the individual is a POI.

[1] This included executives who were convicted, reached a settlement or plea deal with the government, or who testified against others in exchange for immunity from prosecution

Of the executives, 18 are listed as POIs, while the remaining 128 records are listed as non-POIs. 18 POIs is quite a small target sample to work with, but will have to suffice for our purposes.

There are 20 features already available – 14 financial features and 6 email communications features – in addition to the POI labels. The below table describes what each of the features is, and describes the proportion of records for which they are present.

| Feature | Definition | Proportion records present (%) |
|---|---|---|
| Salary | Reflects base salary, executive cash allowances, and benefits payments | 65% |
| Bonus | Reflects bonus payments, additional to salary | 59% |
| long_term_incentive | Reflects long-term incentive cash payments from various incentive programs | 45% |
| deferred_income | Reflects voluntary executive deferrals of salary or incentives. May also reflect deferrals under a stock option or phantom stock unit in lieu of cash arrangement | 34% |
| deferral_payments | Reflects distributions from a deferred compensation arrangement due to termination of employment or due to in-service withdrawals as per plan provisions | 27% |
| loan_advances | Reflects total amount of loan advances, excluding repayments | 3% |
| other | Reflects any other payments (e.g. for severance, relocation, tax advances) | 64% |
| expenses | Reflects reimbursements of business expenses | 65% |
| director_fees | Reflects cash payments and/or value of stock grants made in lieu of cash payments to non-employee directors | 12% |

| | | |
|---|---|---|
| total_payments | Reflects the sum of all of the above payment features | 86% |
| exercised_stock_options | Reflects amounts from exercised stock options | 70% |
| restricted_stock | Reflects the gross fair market value of shares and accrued dividends on the date of release due to lapse of vesting periods, regardless of whether deferred | 75% |
| restricted_stock_deferred | Reflects value of restricted stock voluntarily deferred prior to release under a deferred compensation arrangement | 12% |
| total_stock_value | Reflects the sum of the above three stock features | 86% |
| email_address | Email address for the individual | 76% |
| to_messages | Number of emails sent to the individual | 59% |
| from_messages | Number of emails sent by the individual | 59% |
| from_this_person_to_poi | Number of emails sent from an individual to a POI | 59% |
| from_poi_to_this_person | Number of emails sent from a POI to an individual | 59% |
| shared_receipt_with_poi | Number of times an individual is copied on an email alongside a POI | 59% |
| poi | Classifier on whether or not an individual is a POI | 100% |

It's quite clear already that a large number of features are poorly covered in the data. In particular, six of the financial features are present in less than 50% of the records here.

While at first we might be inclined to exclude poorly-covered features from the analysis, we know from the source data that these values aren't 'missing' as such. Rather, they simply indicate zero values. For example, loan advances only cover 3% of the records here because only a handful of executives ever took loan advances in the first place.

There's therefore no *prima facie* reason to exclude features based on low coverage, but we should be careful to ensure values are read as zeroes rather

than ‘NaN’ for the purpose of the real analysis. (By default, scikit-learn should evaluate them as zeroes for us in any case.)

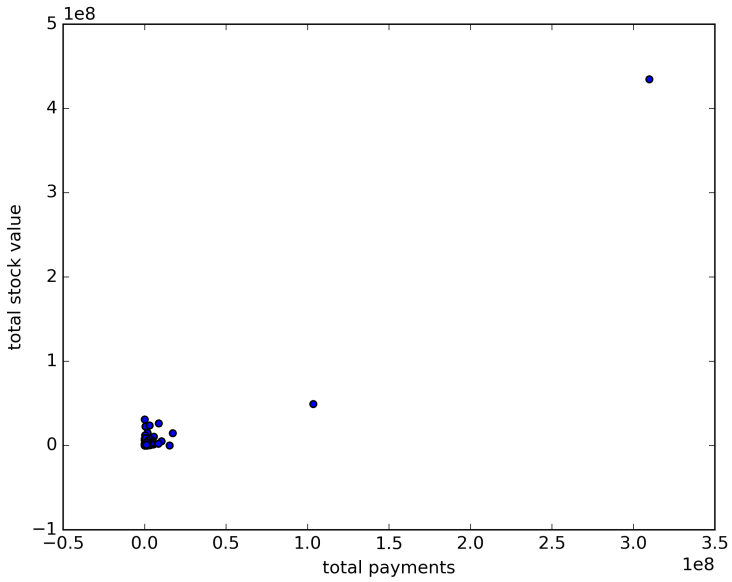
**Outlier detection**

Before proceeding further, it makes sense to detect and remove any obvious outliers in the data. The operative word here is ‘obvious’ – here I’m simply trying to detect any erroneous entries that should clearly be removed. We may later opt to fine-tune the classifier by removing additional entries with high residual error, if required.

Given the relatively small size of the data set, outlier detection can be do performed semi-manually. I graphed some of the features related to financial and email communications data, to try and spot any clear candidates for removal.

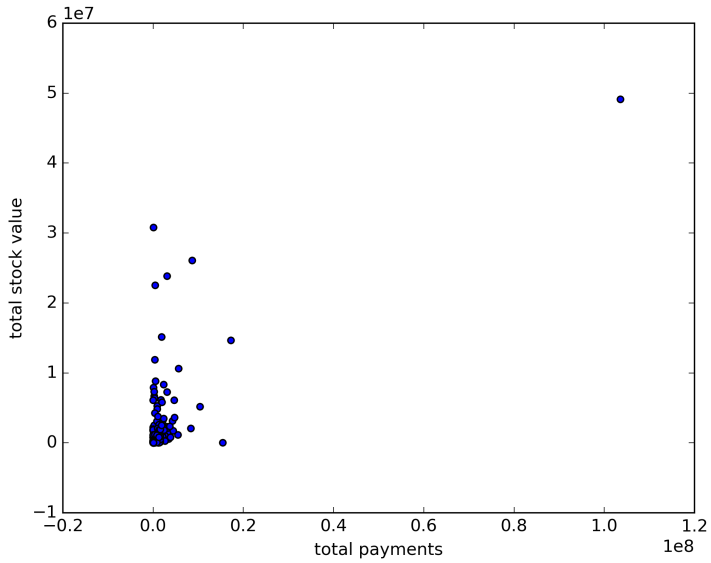
Financial data

Here I graphed total payments against total stock value, both of which are aggregating features that encompass many of the other financial features in the data.




There’s a clear outlier here in the top right, with values several orders of magnitude above the other individuals. This turns out to be a ‘total’ value, and is clearly erroneous. I removed it from the data accordingly.

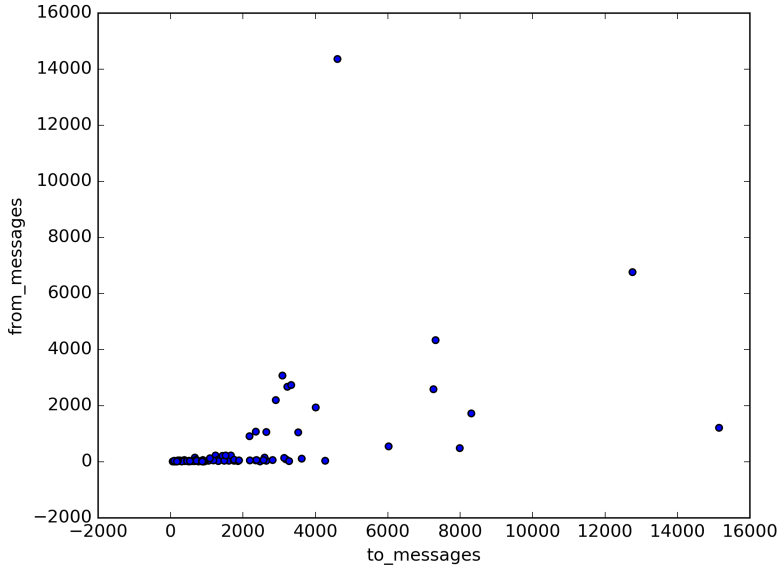
There was a second outlier, which stands out even more sharply once the ‘total’ entry is removed.




A quick investigation reveals that this entry refers to Ken Lay, who was Chairman and CEO of Enron (except for Jeff Skilling’s brief CEO tenure). Clearly this data point isn’t erroneous, and so I’ve left the record in the data for the time being. We may of course opt to remove it later if doing so improves the accuracy of our classifier.

## Communications data

In a similar way to the above, I also graphed emails sent against emails received to uncover any obvious outliers.



Here 8 records visually appear to be outliers, but a quick investigation reveals that all refer to genuine individuals who simply happened to have high email activity. Indeed, if we assume email activity follows a pareto or even a normal distribution, it's unsurprising to see a small number individuals sending or being sent a high volume of messages.

I did however get curious here about the individuals who have *no* emails recorded in the data, and printed out a list of these. I expect that most are simply individuals for whom no email data was released, but wondered if some were simply erroneous records in general.

While most do turn out to be real individuals, this analysis did throw up one erroneous record – 'THE TRAVEL AGENCY IN THE PARK'. I removed this from the data after detection.

While the others were genuine individuals, I thought it prudent to test if any were 'empty' records, i.e. individuals for whom no features were present. I found one such record – 'LOCKHART EUGENE E' – and removed this from the data accordingly.

With three records removed, we now embark on our feature selection with a set of 143 individuals in our data.

**Feature selection**

Now that we've removed some obvious outliers, let's move on to select the features for use in our classifier. Here we'll perform a few steps – scaling features, provisionally evaluating their usefulness, and creating new features as appropriate. As with outlier detection, we may 'fine tune' further once we've moved on to building and testing our classifiers.

*Feature removal*

We may opt to reduce the number of features being used later on in this analysis, but first I evaluated if there were any features that should be removed *prima facie*.

Up front, it seemed to make sense to eliminate 'email address' as a feature, and so I ensured it was not included in the list of features for analysis. It's a non-numeric feature, and not one that I intend to operationalize into a vector in any form.

*Feature evaluation*

I used scikit-learn's SelectKBest function to identify the most useful features in the data. I set k='all' here to view the influence of each feature. The scores for each are listed as follows:

| **Feature** | **Score** |
|---|---|
| bonus | 30.7 |
| salary | 15.8 |
| shared_receipt_with_poi | 10.7 |
| total_stock_value | 10.6 |
| exercised_stock_options | 9.7 |
| total_payments | 8.9 |
| deferred_income | 8.8 |
| restricted_stock | 8.1 |
| long_term_incentive | 7.6 |
| loan_advances | 7.0 |
| from_poi_to_this_person | 4.9 |
| expenses | 4.2 |
| other | 3.2 |
| to_messages | 2.6 |
| director_fees | 1.6 |
| restricted_stock_deferred | 0.7 |
| from_messages | 0.4 |
| from_this_person_to_poi | 0.1 |
| deferral payments | 0.01 |

A couple of observations jumped out to me here immediately.

First, some of the features clearly have very low influence on the numbers. We'll probably opt to remove some of these when fine-tuning classifier algorithms later on, if we find doing so has no material impact on our evaluation metrics.

Second, the email correspondence features generally seem to have a lower level of influence than the financial features. This might simply reflect the fact that financial factors really are greater sources of insight here, but it could also suggest a need to try and draw out more useful features from the email communications data.

*New features*

As a result of the above, I opted to create two new features from the email data. In particular, I chose here to create two ratio metrics:

- to_POI_proportion: this reflects the number of messages an individual sends to POIs as a proportion of their overall email output

- from_POI_proportion: this reflects the number of emails received from POIs as a proportion of overall emails received

My hypothesis here is that what matters is not simply the sheer number of emails an individual sends, but the *proportion* of emails going to or coming from persons of interest.

Based on a re-run of the k_best evaluation above, the to_poi_proportion feature performs quite well (scoring 15.8, making it the third most useful feature). The from_poi_proportion feature appears to contain much less information, with a score of 0.5 on the same analysis.

*Feature scaling*

It makes sense to scale the features in the data, since at present they're on wildly different unit scales (often differing by orders of magnitude). Feature scaling will make sure no features are weighted disproportionately because of this.

I performed feature scaling on the data with scikit-learn's min-max scaler.

**Evaluation metrics**

Before moving on to begin selecting classifiers, at this stage I wanted to build some basic evaluation metrics on which to assess the classifiers we're building.

I made use of three evaluation metrics as part of this work:

- Precision, which assesses the number of true positives out of the total number of entries *classified* as positive. More specifically in our context, this metric determines the number of executives classed as POIs who actually are POIs

- Recall, which evaluates the proportion of actual positives we've evaluated as true positives. More specifically in our context, this metric evaluates the proportion of actual POIs we've *classified* as POIs

- F1, which is the harmonic mean of the above two metrics

Accuracy was not prioritized as an evaluation metric, partly because of the unbalanced structure of the data. We have a small number of positive classes in the data, and accuracy scores may mask problems with our evaluation of positives by focusing on correct evaluation of negative classes.

There seems no obvious up front reason to weight precision or recall more strongly here, and one can make arguments in either direction– do we want to avoid classifying innocent people are non-involved, or do we want to broaden the net of our search to avoid missing any POIs? As a result of this, I gave primacy to the F1 score when making decisions, which provides a weighted score of both of these metrics. To be clear, we're aiming here for classifiers that are high on *both* precision and recall, but if there's a close call on performance I opted to prioritize based on the F1 score.

**Classifiers**

*Initial shortlist*

With evaluation metrics in place, I went on to test out a variety of classification algorithms. Initially I tested three, and performed some tuning on each – Naïve Bayes, Logistic Regression, and Support Vector Machines (SVM).

I found SVM and logistic regression to be the most reliable and high performing algorithms early on.

*Tuning the algorithms*

I proceeded at this stage to trial the algorithms, tune and test them against the data as I went through.

Tuning is critical at this stage. The algorithms themselves here are fairly blunt instruments, and need to be tailored to fit the data at hand. Tuning can increase performance on our evaluation metrics, and make sure we've got the best possible classifier to classify effectively.

Rather than tweak every parameter manually, I used scikit-learn's Gridsearch function to optimize parameters for the algorithms on my shortlist. (Naïve Bayes is a partial exception here, in that there aren't such obvious parameters

to ‘tune’. I did experiment with a Bernoulli rather than Gaussian algorithm, although found no additional performance gain from this.)

I tuned what I saw as the most critical parameters on each. For the SVM, I tested different values for ‘C’ and gamma, and a range of different kernels. For logistic regression, I tuned based on ‘C’, tol, and verbosity.

Gridsearch didn’t always land on material improvements from my own manual tuning of the algorithms. I ended up using the gridsearch-tuned parameters nonetheless, since I believe they’ll be more reliable and less prone to overfitting – I used gridsearch to tune them on 3-fold cross validation.

*Further feature reduction*

At this stage I wanted to experiment with reducing the number of features in the data further. As we saw above, several features in our Select K-best analysis appeared to add little information to our algorithm – at best they may be slowing it down.

To explore this, after some experimentation I ran a further analysis including only those features with scores higher than 8, which resulted the 11 most influential features being included. I then re-tuned the algorithms for the assessment, and found that some had *increased* in performance as a result of this. At this stage logistic regression and Naïve Bayes were performing the highest out of the remaining algorithms being tested.

**Validation**

Before concluding, it’s important to run some final validation to ensure our algorithm generalizes well. While we might hope to have reduced overfitting via the grid-search parameter tuning process, and by using a separate training data set throughout, it makes sense to spend extra time being absolutely sure no overfitting has occurred. We want our classifier to work as well as possible in reality, not just to optimize for a single given training set.

Here I tested two of my algorithms on a cross-validation set to gain further confidence of their robustness, using k-fold cross validation in sci-kit learn. I found that both seemed suitable for our performance threshold in my analysis, although Naïve Bayes performed highest in tester.py. Somewhat to my surprise, I therefore used an unmodified Naïve Bayes in the final poi_id.py file as the optimal algorithm of those tested.

**Conclusions**

This was an interesting exercise, and highlighted how challenging analyzing a data set with so few true positives can be.

Reflecting on the work, I was struck by how much of a difference up-front work on the data and feature set can make. I also reflected that it's well worth 'multi-tracking' by keeping several algorithms in play as you try to build a classifier. In my case, it would have been very comfortable to drop Naïve Bayes early on, but in the end it proved a genuinely useful algorithm to reach the desired performance threshold – despite its simplicity.