Identifying Enron Persons of Interest (POIs) - A Machine Learning Algorithm

# Udacity Introduction to Machine Learning Final Project

## Adam Wright

## August 1, 2016

1. **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. 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 goal of this project is to build a supervised machine learning algorithm to identify people who committed fraud from the public Enron email and compensation dataset. The trained algorithm could then be employed on novel to data to determine whether or not an individual is likely to be a fraudster.

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives. This is the dataset that I will be using to classify Enron employees either as Persons of Interest (POIs) - people who warrant further investigation for potential fraud - and not POIs.

The dataset is ideally suited for a machine learning algorithm. The data is labeled - 18 of the 146 people in the dataset where later indicted. And while the dataset does not include a particularly large sample (i.e. only 146 persons) it is feature rich with 26 financial and email related variables. Many of these features had a lot of missing values - for example some cases had no financial data while others had no records of email communication - but I decided that imputing zeros for missing data was warranted and was ultimately satisfied with the results.

In examining the data there were two cases that were major financial outliers: 'TOTAL', 'THE TRAVEL AGENCY IN THE PARK'. Both of these outliers were dropped before passing the data to the algorithms.

1. **What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test 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”, “properly scale features”, “intelligently select feature”]**

I ended up creating three new features for the emails sent and received. The variation in email volume is huge so some sort of correction needed to be made to account for it. A simple way to control for volume is to make the feature of interest the ratio of emails that a person sent or received from a POI. This way a high volume emailer will have to send/receive a lot of emails to POIs to stand out while a low volume emailer who sent/received emails mostly to POIs will also stand out. To control for the potential confounder of email volume, I created three new features: poi\_ratio (ratio of emails from and to POIs/emails sent and received), to\_poi\_ratio (emails to POI/emails sent), and from\_poi\_ratio (emails from POI/emails received).

In my most successful classifier - an AdaBoost using decision trees as the base - I ended up selecting 16 features. They were: salary, total payments, exercised stock options, bonus, restricted stock, shared receipt with poi, total stock value, expenses, loan advances, other payments, director fees, deferred income, long term incentive, ratio of emails to POIs, ratio of sent emails to POIs, and ratio of received emails from POIs. If I were to iterate on this classifier, this would seem a good point to attempt some sort of dimensionality reduction - perhaps a PCA on financial features to uncover latent variables, simplify the input, and reduce classifier variance. However, this feature set met the required threshold so no further feature selection was undertaken.

For tables of feature importances as well as a discussion of the selected parameter values, please see the IPython notebook in the same GitHub repository as these responses.

I did not have to do any scaling of features. After eliminating the two outliers discussed above, the remaining points that seemed to be outliers actually made sense considering the persons they were associated with (super highly compensated executives like Ken Lay and Jeff Skilling). The fact that such individuals were POIs and receiving such outlandish compensation was informative.

1. **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”]**

I ended up trying three algorithms: Naive Bayes, Random Forest, and AdaBoost. I used Naive Bayes as a baseline which to compare the more sophisticated algorithms against. In my initial testing, for which I did not do any feature selection or tune any parameters, Naive Bayes was indeed the worst performer and thus I did not attempt to tune any parameters for it.

After feature selection and parameter tuning, the best performing algorithm was AdaBoost. It had very similar rates of precision and recall and an F1 score over 0.3. The Random Forest actually had superior accuracy and precision, but it had a much lower recall rate resulting in a F1 score closer to 0.2 than 0.3. That being said, considering the potential application of this classifier - using it to pinpoint who to investigate for potential indictment on fraud charges - a conservative algorithm might actually preferable even if it lets more actual POIs get away. So while the project instructions were to submit an algorithm with both recall and precision > 0.3 I think for this application I actually prefer the Random Forest classifier in spite of its low precision.

1. **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? (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 item: “tune the algorithm”]**

Most machine learning models are parameterized so that their behavior can be tuned for a given problem. Finding the best combination of parameters for a given problem is essentially a search problem. In the case of classification problems, you are looking for a set of parameters that maximizes some metric of the accuracy of the predictions that the algorithm produces.

Sci-kit learn comes with a number of helpful functions to assist in this searching problem. In my case, I employed the GridSearchCV() function on both my Random Forest and AdaBoost algorithms to discover an optimal set of parameters for the Enron data.

1. **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 item: “validation strategy”]**

In machine learning, validation is testing your model on novel data. This is usually accomplished by dividing data into training and test sets (a common heuristic is 10% of data should be saved for testing). There are different ways to make this split, and choosing between them depends mostly on the number of cases that you have. In this case - because I had a relatively small sample - I made use of the SciKit Stratified Shuffle Split function to create 10 different test training splits (stratified so that I had a similar number of POIs in each testing and training split) on which to train and test my chosen algorithms.

As a final validation step I used the provided test\_classifier function provided in tester.py in order to ensure that at least one of my algorithms had both a precision and recall of at least 0.3.

1. **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”]**

Precision is defined as the True Positives / (True Positives + False Positives). In this case it is a measure of the proportion of people indicated by the algorithm as potential POIs are actually POIs. My AdaBoost classifier (the best performer) had a precision of 0.326 - that is 32.6% of people it identified as a POI were actually POIs.

Recall is defined as True Positives / (True Positives + False Negatives). In this case it is a measure of the proportion of people who are truly POIs that the classifier identifies as such. My AdaBoost classifier had a recall of 0.321 - that it correctly identified 32.1% of true POIs as being POIs.

Finally, F1 score is defined as (2 \* Precision \* Recall) / (Precision + Recall). Given that Precision and recall were nearly identical, it should come as no surprise that he F1 score was extremely similar at 0.324.