Identifying Enron Persons of Interest

Data Exploration and Outlier Investigation

This project used machine learning to identify Persons of Interest ("POI"s) in the Enron scandal. A POI is someone who was charged for a crime related to the scandal, settled without admitting guilt, or testified in exchange for immunity. The dataset used contains 146 data points, each data point representing a person who worked at Enron. Out of the 146 data points, 18 of them are POI's and 128 are not. There are said to be about 35 POI's total (depending on how you define one), so not all of them are in the dataset. There is a mail directory that comes with this dataset, in which all the emails possible were collected from each person. The problem is that only four POI's have mail directories. Therefore, the machine learning process was more dependent on the other features in the dataset. Each data point has 21 features, those features falling into three categories: financial (e.g. salary, bonus, expenses), email related (e.g. number of "to" messages sent, number of messages sent from a POI) or POI label (an indicator variable for whether the person is a POI). This project uses supervised learning models, since the dataset already contains labels for POI's and the goal is to find the best method for predicting whether someone is a POI. Therefore, the right combination of features, algorithm and parameters would lead to the best classifier of a POI. It is important to note that six of the features had more than half of the values missing after my exploration of the dataset, including "deferral_payments", "loan_advances", "restricted_stock_deferred", "deferred_income", "long_term_incentive", and "director fees." Both "restricted_stock_deferred" and "director_fees" did not have any POI's in them, so they were removed from the feature list off the bat, since you can't predict who is a POI based on a feature that does not have any. Additionally, "email_addresses" was not used since a feature must be numeric in order to work with the feature_format function, since the function converts values to floats.

A good amount of outliers came up for the financial features. In order to find them, I plotted each variable at least once with another variable and then looked at which data points stood out. I then identified those points in the original data table. Kenneth Lay, the chairman of the board of directors at the time, and one of the biggest players in the scandal (so definitely a POI), was a clear outlier in most cases. He had abnormally high values for salary, bonus, loan advances, exercised stock options, long-term incentives, restricted stock value, total stock value, and

total payment. He was not removed because he is an important POI and his high values are indicative that he benefitted a lot from the scandals. Another big outlier was Mark A. Frevert, who was the vice chairman of Enron but is not a POI. He, like Ken Lay, had about a one million dollar salary, as well as very high values for deferral payment, loan advances, deferred income, and total payment. I removed him from the data dictionary since he was way higher than most people in many categories and would throw off predictions for non-POI's based on those high numbers. Other notable outliers were Jeffrey Skilling (POI and CEO of Enron for most of the scandal, with very high salary and bonus), John J. Lavorato (very high bonus, long term incentives and total payments), Joseph Hirko (POI with thirty million in both exercised stock options and total stock value), Amanda K. Martin (with a long term incentive of five million), Thomas E. White Jr. (with a restricted stock value of thirteen million), George McClellan (with high expenses that basically matched his average salary), and Mark R. Pickering (with a high loan advancement). Additionally, Kevin P. Hannon (POI), Kenneth D. Rice (POI), and Phillip K. Allen had normal salaries for Enron (200,000 to 400,000 range) but a very high deferred income (about negative three million each). No other people besides for Mark A. Frevert were removed from the data dictionary, since they were not clear outliers in many categories like he was. Two keys that were not names appeared in my exploration, including "TOTAL" and "THE TRAVEL AGENCY IN THE PARK." Both were removed, since I wanted the algorithms to make predictions based on people only.

Feature Creation and Selection

I ended up using the following features in my POI identifier: "bonus", "deferred_income", "expenses", "long_term_incentive", and "restricted_stock." I picked them by creating a Decision Tree and finding the features with an importance level greater than 0.10 (for perspective, if all features were equally important, they would all have an importance of 0.01). Here are the importances of each feature in order (rounded to the nearest thousandth):

- bonus: 0.206
- long_term_incentive: 0.168
- expenses: 0.164
- restricted_stock: 0.121
- deferred_income: 0.115

I did not need to use feature scaling because the two algorithms I tested were Naïve Bayes and Decision Trees, neither of which requires scaling. Naïve Bayes does not

require scaling because it deals with probabilities of a class having certain values, and these probabilities would not be changed with scaling. Similarly, Decision Trees does not require scaling because the same decision boundaries will be created, just at a different scale. I also engineered a new feature called "deleted_emails", which counts the number of emails deleted by each person in the dataset. I did this by creating functions in Python to go through the mail directory (called "maildir"), and counting the number of files in each person's "deleted_items" folder. I then checked if that person was in the dataset; if so, I added that count to their new value of "deleted_emails." My reasoning behind creating this feature is that maybe POI's deleted their emails more than non-POI's in order to not get caught at the time of the scandal or hide the evidence afterwards. This feature was not used in the POI identifier, however, since it did not have a high enough importance.

Algorithm Testing and Performance

I ended up using Naïve Bayes, but also tried Decision Trees with a min_samples_split of 10 (see below for explanation). Here is how each algorithm performed:

	Naïve Bayes	Decision Trees
Accuracy	0.847	0.847
Precision	0.454	0.441
Recall	0.352	0.266
F1	0.396	0.331
F2	0.368	0.288

Naïve Bayes either tied with Decision Trees or beat it in every category (though technically it had an accuracy score of 0.84686 but that was rounded up to 0.847).

Parameter Tuning

Tuning the parameters of an algorithm means systematically finding the best combination of parameters to use for a specific algorithm in order to maximize its performance. Even if you know your dataset will work well with a certain algorithm, it does not mean you are guaranteed to classify most of your points correctly just by choosing that algorithm. This is because you need to make the algorithm work with the shape of your data. If the tuning is not done well, you may overfit or underfit your data, because the algorithm is responding too much or too little to the quirks in your dataset. Your accuracy and other evaluation metrics may

suffer as a result. Nevertheless, not all algorithms have parameters that need tuning.

For the algorithm I chose, which was Naïve Bayes, there were no parameters to tune. However, for the other algorithm I tested, Decision Trees, I tuned the parameters using GridSearchCV(). I tested out 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 for the min_samples_split parameter and I got 10 as the ideal. I therefore used 10 for min_samples_split.

Validation

Validation is splitting a dataset (usually an independent dataset) into a training set and test set in order to measure the performance of a classification or regression algorithm. A classic mistake that can go wrong is overfitting an algorithm to the training data, making your classifier or regression not as powerful when more data gets involved. This occurs when too few points are in the training set or evaluation is done on the training set instead of the test set. I validated my analysis with a Stratified Shuffle Split (implemented in tester.py), which essentially performs k-fold cross-validation (dividing the data into k bins, where each bin alternates as the test set while the rest of the bins are used for training and the average of all results are taken) but shuffles the data points beforehand. The shuffling ensures that about an equal amount of POIs occur in the training and test sets for every fold.

Evaluation Metrics

For the Naïve Bayes algorithm, the average precision for my tests was 0.454 while the average recall was 0.352, or 45.4% and 35.2% respectively. This precision score means that about 45% of the time, if my algorithm marked someone as a POI, it was not a false alarm. However, that still leaves 55% of POI's identified as false alarms. For recall, this score means that about 35% of the time, if a POI showed up in the test set, my algorithm correctly identified them. Still, 65% of the time, it failed to identify a POI when there was one. While these numbers can be brought up with some improvements to the algorithm, it is important to remember that the algorithm only has 18 POI's to work with, so it is probably fairly difficult to get precision and recall scores above 0.70. Additionally, the average accuracy score for the Naïve Bayes algorithm means that it is correctly classifying 85% of data points as POI or non-POI, which is pretty good for a small number of training points.