Project 4 Machine Learning
Enron Submission Free-Response Questions
John Gritch

A critical part of machine learning is making sense of your analysis process, and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your coach evaluates your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: Link to the rubric
Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that the rubric. If your response does not meet expectations, you will be asked to resubmit.

Once you've submitted your responses, your coach will take a look and ask a few more focused follow-up questions on one or more of your answers.

We can't wait to see what you've put together for this project!

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 correctly assign a label of person of interest or not a person of interest (for an investigation into business fraud) to a small subset of former Enron employees using a machine learning algorithm and the provided set of financial and email related data. The dataset is comprised of financial information about the employees such as salary and bonus pay. The data was made public through Case No. 01-16034 of the Southern District of New York US District Court (as reported by FindLaw.com). The email data are manufactured features, such as total email counts and contacts with other persons of interest, provided for us based off of the now-public corpus of Enron emails. The running dogs of any good conspiracy being money and lies, I think this data is well suited to our investigatory purpose.

The outliers caused by true errors in data entry or data conversion (as opposed to merely extreme values) were entries for *TOTAL* and *THE TRAVEL AGENCY IN THE PARK.* I removed these entries because they were not people, but decided not to remove extreme values from the dataset for several reasons. Firstly, taking into account the scale of the fraud conducted at Enron it seemed natural that extreme values would be very significant. Second, the purpose of our machine learning algorithm is to flag individual people who we think tried or succeeded at gaining a criminal windfall.

In other words our algorithm is interested in the outside edges of the data as opposed to an algorithm that might be better served by ignoring edge cases as it seeks to find the more stable semi-constants in a changing environment.

2. **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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]**

Features: exercised_stock_options, bonus, salary

In the end I did not have to do any scaling as each of my features were a form of payment to the employees in units of dollars. I did experiment with creating and scaling email features (feature value / max feature value in dataset), but ultimately did not include the features in my final model.

One composite feature that I made was *poi_exposure*. Poi_exposure attempts to combine a consideration for emails that shared a receipt with a POI along with two other created features that represented:
1) *prop_from_poi* the proportion of a person's received emails that came from a POI
2) *prop_to_poi* the proportion of a person's outgoing mail that was to a POI.

The formula for *poi_exposure* was: weighting factor * (emails to POI / all sent emails) + weighting factor * (emails from POI / all received emails) + weighting factor * (shared receipt along with poi / all received emails).

My hypothesis was that there may be a link between the total number of contacts a person had with

a POI and they themselves being a POI. However, after trying a wide variety of weighting factors this feature failed to improve the results.

3. **What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: “pick an algorithm”]**

I ended up using a K Nearest Neighbors Classification algorithm. I also tried and attempted to fit Naive Bayes, Linear Support Vector Machines, and Decision Tree Classification algorithms, but I never achieved results above those returned by k-NN.

4. **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 don’t 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 if you used, say, a decision tree classifier). [relevant rubric item: “tune the algorithm”]**

Most algorithm implementations in scikit learn come with a number of parameters that have an affect on how the algorithm is processed. Tuning the parameters is the process of setting those parameters so that you get the “best possible results”, however you define that. Some of what makes a result “the best” is simply fitting the algorithm to your data and processing equipment. Some parameters allow you to extract more information from your given set of features by aligning with or somehow complementing the correlations (and maybe the actual causal determinants) present in the data. Other parameters allow for decreasing the amount of computational processing needed to maintain a certain level of performance. However, a large part of tuning an algorithm is the give and take in finding the right balance, for this place and time, between two opposing tendencies like precision and recall.

In k-NN I first tuned the *weights* parameter setting it to 'distance' so that closer points were more heavily weighted than points farther away. Then I set the *algorithm paramter* to 'brute' because the dataset's small size and low dimensionality made that feasible and there was no need to implement a K-D Tree or other formulation to improve computation efficiency at the possible expense of correlative ability.

Finally I set *n_neighbors* to 6 because, through pure experimentation, I found that to work best. I did find that n_neighbors = 2 did produce a slightly higher recall, but at the cost of much worse

precision.

5. **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"]**

Validation is testing your classifier (or regression) on a subset of training data that was kept separate from the data that was used to train the algorithm, i.e. the testing set. Validation provides an idea of how your algorithm will perform at its intended function with real-world data. A classic mistake is to train on your testing data (or to test on your training data) and the result of this mistake will be artificially high values for your evaluation metrics and an algorithm that is over-fit to the idiosyncrasies of the training data.

The tester.py script used sklearn's StratifiedShuffleSplit which is a mix of K Fold cross validation and ShuffleSplit. This sklearn function splits the data into new training and testing splits 1,000 times (or as set in tester.py) and then trains and tests the algorithm on each of those splits. The splits are randomized stratified folds that preserve the percentage of samples from each class. (source sklearn.org).

6. **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"]**

Accuracy: 0.88708 Precision: 0.74270 Recall: 0.40700 F1: 0.52584 F2: 0.44745
Total predictions: 13000 True positives: 814 False positives: 282 False negatives: 1186
True negatives: 10718

The recall is a measure of how likely you are to find something given that you want to find it. If you are a shrimp trawler, recall is the percentage of shrimp in your fishing area that you catch in your net. Precision on the other hand is the ratio of shrimp to bycatch or - out of all of the things you pull up in your net; crab, ladyfish, old Pepsi bottles - what percentage of that assorted mass is shrimp.

More formally recall is the quotient of true positives divided by the sum of true positives plus false negatives. Precision is the quotient of true positives divided by the sum of true positives and false positives.