Enron Submission Free-Response Questions – Tyler Moazed

- 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"]
    1. The goal of this project is to use machine learning to identify Persons of Interest (POI) using the data provided in the Enron email and finances combined data set. Machine learning is helpful in this task as it allows someone to "comb" through data sets and use features within each data point to make classification predictions much quicker and more reliably than simply doing the investigation by hand or though data visualization alone.

       The Enron data set used in this project is a combination of two data sets, one containing financial information about Enron employees and executives and the other containing company emails of Enron employees. As the focus of the project is to identify individuals who may have been involved in criminal financial activities, having access to financial data is critically important in helping identify POIs. Equally valuable is access to emails, which allows us to not only track who emailed who, but to also track email correspondences to and from other POIs. This is also important as the criminal activities done by those in Enron were done in collaboration, so it is very likely that POIs were in frequent contact with other POIs.

       The original data set contains 146 data points, of which 144 are actual people connected with the Enron company and potential POIs. The two "obvious" outliers that were removed from the dataset prior to the machine learning investigation were the "total" data point and the "travel agency in the park". Since these are not real people and have no possibility of being POIs, they were removed. Of the 144 remaining data points/individuals 18 are identified as POIs in the data set. The original data set has 21 features for each data point, but six features (deferral_payments, deferred_income, director_fees, long_term_incentive, loan_advances, restricted_stock_deferred) had missing value for more than half (72) of the data points.

- 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. [relevant rubric items: "create new features", "intelligently select features", "properly scale features"]
    1. My feature selection process began by creating two new features, "fraction_to_poi" and "fraction_from_poi". Based on the information provided in the lesson, I have come to understand the importance of email correspondences to and from POIs as a way to identify other POIs. I also understand that the built-in features in the data set, "emails_to_POI" and "emails_from_POI", did not consider the ratio of these correspondences to other non-POI emails. So, I created these two new features and used them to as replacements for "emails_to_poi", "emails_from_poi", "to_emails" and "from_emails". The creation of these new feature was extremely important for in

my final algorithm model as the “fraction_to_poi” model was fifth most powerful features in my final algorithm, with a score of 4.64 and a p-value of .031. In comparison, the data set provided “shared_recipient_poi” features was the 9th most powerful, with a score of only 2.43. These were the only two email related features in the final model. When I use my final alogorithm with only the original email features, I get lower evaluation metrics(accuracy:0.82, precision: 0.30, recall:0.26) than when only using the features I created.

Beyond creating these two new features, and substituting them for the other email focused features (excluding 'shared_receipt_with_poi') I included all other features, 18 in total, in my initial features list. This is because I utilized univariate feature selection with SelectKBest (using a chi-squared score to determine the best features). Since SelectKbest will select the highest performing features, I felt it best to include all possible features and let the algorithm figure out the best features to include. In addition, I used GridsearchCV to tune the SelectKBest feature selection. Ultimately, the algorithm utilized the following 12 features listed below with their scores and P-values:

- 'exercised_stock_options', '6.85', '0.009'
- 'loan_advances', '6.69', '0.010',
- 'total_stock_value', '5.48', '0.019',
- 'bonus', '5.12', '0.024',
- 'fraction_to_poi', '4.64', '0.031',
- 'salary', '3.05', '0.081',
- 'total_payments', '2.78', '0.095',
- 'long_term_incentive', '2.54', '0.111',
- 'shared_receipt_with_poi', '2.43', '0.119',
- 'other', '1.72', '0.190',
- 'director_fees', '1.50', '0.220',
- 'expenses', '1.49', '0.223'

In addition to SelectKBest, I also utilized the MixMaxScaler preprocessing package. Since I included both financial and email related features, which include very different distributions and units of measurement, I felt that feature scaling was necessary. The use of a scaling package allowed the feature selection package (SelectKBest) to compare features despite different distributions and the MinMaxScaler helped provide robustness to very small standard deviations of features and preserving zero entries in sparse data (which was the case here).

It is also worth mentioning that in the supplemental file, final_project_draft.py (which shows several alternative classifiers and parameter tunings), I show that using principal component analysis (PCA), does reduce the total number of features used, it does not improve the precision and recall scores (although they are both still greater than .3)

- 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”]
    1. I ultimately ended up using the Decision Tree algorithm. As provided in the initial code, I started with a Gaussian Naïve Bayes algorithm. With this algorithm and all

the features and doing little else, I was able to get a very good recall score (.834), but very bad accuracy (.378) and precision (.16). From there I decided to explore two more robust algorithms, Decision Tree (DT) and Random Forest (RF). As documented in the supplemental file, final_project_draft.py, I explored a variety of feature combinations, scalers (standard and MinMax), feature selection packages (SelectKBest) and composite feature packages (PCAs) with both the Decision Tree and Random Forest algorithms. Ultimately, the best test scores I got for Random Forest were:

1. RF with MinMax and PCA: Acc:0.832; Precision:0.232; Recall: 0.114
2. RF with MinMax and SelectKBest: Acc:0.854; Precision: 0.4293; Recall:0.278
3. RF with MinMax, SelectKBest and PCA: Acc:0.846; Precision:0.354; Recall:0.158

Similarly, I explored multiple iterations of the DT algorithm, using the same combinations I did with random forest. The only other combination to score over 0.3 in precision and recall was a MinMax Scaler, SelectKBest and PCA. Ultimately, the best combination was the DT algorithm with the MinMax Sckaler and SelectKBest. The only other combination to score over 0.3 in precision and recall was a MinMax Scaler, SelectKBest and PCA. Here are the results of my two best algorithms using decision tree algorithms:

1. DT with MinMax, SelectKBest and PCA: Acc: 0.82; Precision: 0.32; Recall: 0.303
2. DT with MinMax and SelekctKBest: Acc: 0.85; Precision:0.432; Recall:0.35

- 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? What parameters did you tune?
    1. Each machine learning algorithm has a variety of "parameters" that can be "tuned" or adjusted to best handle the data and features being passed to the algorithm. Sometimes called "hyperparameter optimization", this is typically the final step of the machine learning process as it come after you have selected features, selected an algorithm and decided to implement other elements including feature scaling, automated features selection and/or PCA. This is an extremely important step in developing the best possible algorithm for the data being analyzed as the basic algorithms are NOT one size fits all. If you fail to tune your algorithm, the outcome and evaluation metrics will suffer. For example, my un-tuned best algorithm was only able to get a precision of 0.39, a recall of 0.162 and an F1 score 0f 0.23. This same algorithm tuned resulted in a precision of 0.43, a recall of 0.35 and an F1 score of 0.39. The significantly better results were exclusively the result of tuning my algorithm.

       The tuning of my algorithm was done using the GridSearCV with parameter scoring based on F1 score (as opposed to simply accuracy) and cross validation using a StratifiedShuffleSplit. The parameters tuned for my Decision Tree algorithm with SelectKBest were:
        1. min_samples_split: [2, 3, 4, 5]
        2. max_depth: [4, 5, 6, 7, 8, None]
        3. random_state: [42, 53]
        4. kbest_k: [2, 4, 6, 8, 10, 12]

- What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?
    1. Validation is the process of setting aside a certain amount of the data you have available and using is for testing or "validating" your algorithm. This allows for the algorithm to be trained on certain percentage of the data and tested on different data. If you do not validate correctly your algorithm can suffer from overfitting. In these cases, the model is extremely well tuned to the data it was trained on, but when given new data it performs badly and has bad evaluation metrics on the testing data. I validated my analysis in two ways. First, I used train_test_split to set aside 30% of the data for testing. I then used to the training data (70% of total data) to fit my initial classifier and get some initial accuracy, precision and recall scores by testing the classifier with the test data. Then I used a StratifiedShuffleSplit for cross validation within my GridSearchCV, and it is also used within the test.py script for final validation. The use of a stratified validator is import for analyzing the Enron data as the data is small with very important class distribution (POI v. Non-POI). To accurately cross validate my model, the StratifiedShuffleSplit creates many folds (100 for my cross validation and 1000 for the tester.py validation) of random train/test splits to determine the validation of my model. But, unlike train_test_split and other non-stratified validators, StratifiedShuffleSplit uses stratification to rearrange the data to ensure the train/test split closely represents the data as a whole, with the proportion of classes maintained. This type of cross validation will help in terms of reducing both bias and variance. Again, this is extremely important in this exercise as the data set is small and the number of POIs within the data set is also very small, so the number of POIs with the training and testing split should be maintained as best as possible for optimal classifying.

- 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.
    1. For my final, tuned, algorithm my average precision and recall scores using a StratifiedShuffleSplit for cross validation were 0.432 (precision) and 0.350 (recall). Out of 15000 total predictions in the tester.py code, I got 699 true positives, which were incidences where my algorithm classified a data point as a POI when in fact it was a POI. The recall score, 0.350 said that my algorithm could correctly predict 35% of the POIs out of all POIs in the entire testing dataset. Or I got 699 right out of 2000 actual POIs. The precision score is the percentage of data points my algorithm correctly labeled as a POI out of all data points my algorithm labeled a POI. Or while my algorithm correct predicted 699 data points as POI it incorrectly labeled 919 data points as POIs (when they were not POIs).