P5 Enron Fraud Submission – Heather Capell

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?**

The goal of the project is to use data sets to make a predictive model that can determine Persons of Interest (POI) in the Enron fraud case, using a relevant set of features related to financial measures and email data. Machine learning is useful because it allows for many different ways to classify the data to use in the model, and to measure how well iterations of the model can predict. The Enron corpus includes financial data related to individuals involved in the Enron fraud investigations, as well as information about counts of emails sent to or from individuals at Enron. Added to the project is a manually gathered list of Persons of Interest that we are interested in predicting. This project gives me hands-on experience in applying the principles and techniques discussed in the Machine Learning course.

Exploring the dataset a bit, there are records for 145 persons, of which 18 are flagged as a POI. There are 21 features available in the dataset, although I narrowed it down to POI plus 5, and then later refined it to POI plus 3 (discussion of feature selection in Section 2 below). For the final 3 features selected (('salary','loan_advances', 'exercised_stock_options'), almost a third of the first (51/145 for 'salary') and last (44/145 for "exercised_stock_options') had missing values, and almost all (142/145) of the loan_advances had missing values. I still like using the loan_advances, because observation of the PDF file on financial information reveals a common linkage between a POI and a loan advance, and my intuition is that they are in fact related.

For outliers, the financial data included a Total or summary line that needed to be removed. I considered removing additional outliers, as in the Machine Learning class lessons for high bonuses and salaries, but I think those features should be kept in the model because we are trying to find POI's that often had those higher-end salaries and bonuses.

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 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.**

I initially selected by hand 5 features ('salary', 'loan_advances', 'expenses', 'exercised_stock_options', 'from_poi_to_this_person'), based on intuition that I wanted to test. After evaluating features using KBest and reviewing importances of features under the Decision Tree model, I picked the 3 features that were the best-performing, i.e., had the greatest contribution to the model predictions: 'salary','loan_advances', 'exercised_stock_options'. Using the KBest, I had feature scores for my final selected features as follows: 'exercised_stock_options' - 19.47; 'salary' - 12.87; 'loan_advances' - 6.07. In earlier iterations of the Decision Tree algorithm, I found that (other than POI) 'loan_advances' and 'expenses'

features had the greatest importances (those greater than 0.10), although with later tuning in the final version of the model, only POI had an importance greater than 0.10.

I created a new feature 'poi_to_percent' that measures what percentage of emails to the person is from a POI, to see if that new information added to the predictive capability of the model. When I added the new feature into the model however, although Accuracy was good (0.80262), it made Recall fall below the 0.3 threshhold (0.29750), so I chose not to use it in my final set of features.

I also performed feature scaling on the SVM algorithm, which did improve Accuracy to 0.83815, but Precision was lower (0.3667) and Recall was unacceptably lower (0.0715) when I used this model, so I did not keep it.

Selected Results: SVM (scaled) vs. Decision Tree

	SVM (scaled)	**Decision Tree**
Accuracy	0.83815	0.81885
Precision	0.3667	0.39758
Recall	0.0715	0.3445

3. **What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?**

I tried several different Classifiers to compare results: Naïve-Bayes (Gaussian), SVM, Decision Tree, and KNeighbors. Initially, the Decision Tree had the best Accuracy, particularly after removing the Outlier for the Total row in the financial data, with the loan_advances and expenses features having the greatest importances. Accuracy further improved when I changed the training set from 0.3 of the data to 0.5 (half of the data used to train the model). However, although the Precision of all the models was above 0.3, Recall was consistently not good for each Classifier I tried. I was able to improve Recall somewhat by reducing the min_samples_split parameter for the Decision Tree from 40 to 10, but still could not get above the 0.3 threshhold without a more targeted feature selection (see discussion under Section 2 about use of KBest).

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 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).**

Parameter tuning allows you to adjust the algorithm to fit the data better. Without tuning, you may miss a possibility for a better-fitting algorithm with greater precision and predictive capability. I tuned manually at first by trying different parameters, and then used GridSearchCV on my SVM Classifier to find the best parameters to use with that Classifier. However, it did not yield a different result in performance of that Classifier for this particular data set, so I did not need to further change the parameters used.

5. **What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?**

Validation involves testing your model against a separate portion of the dataset, to see how well it performs. By setting aside a portion of the data to train the model, you can estimate the performance of your model using the remaining portion of the data set, which is independent of the training data set. If you use too much of the dataset to train, you run the risk of overfitting the model, which can result in great accuracy in the training set, but a much lower accuracy and performance in the test data set. Overfitting makes the model too inflexible when new data is encountered. That said, I used Train/Test Split validation method and opted to increase the size of the training data set a bit, from the original 0.3 to a larger 0.5 of the data, because I was able to get better accuracy, precision and recall on the testing set by including some additional data points. I did not want to go beyond 0.5 since I got the performance desired with that portion of training/testing split.

The Stratified Shuffle Split method of cross-validation is used in the test_classifier function in tester.py. This method combines StratifiedKFold and ShuffleSplit, resulting in stratified randomized folds. The folds are made using the same percentage of samples for each class, and the data gets shuffled into random order before organizing into k number of batches.

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.**

Precision is the portion of correct positives (true positives) out of all predictions for positives; it lets you know the percentage of positive predictions that are correct. Recall is the portion of correctly predicted positives (true positives) out of all actual positives; it lets you know the percentage of actual positives that your model is able to predict, in other words, how complete your model results are. My Decision Tree Classifier algorithm, with tweaks to feature selection, training set size, and min_sample_split size of 10 yielded Precision and Recall `0.39701` and `0.345`, above the required 0.3 threshhold. Hooray!