Enron Submission Free-Response Questions
Cynthia O'Donnell

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 purpose of this project was to identify which of Enron's employees were likely to have committed fraud and, therefore, warrant further investigation. There are 146 observations (i.e., people) in the dataset and 18 of the people were labeled as a 'poi' or person of interest ("POI"). There were 21 features in the dataset, 15 financial features and 6 email features. Many of the features had a lot of `NaN` values.

There were three outliers that I removed. Two were not people (`TOTAL` and `THE TRAVEL AGENCY IN THE PARK`) and one was a person who had `NaN` values for all features. The dataset is relatively small and the number of POIs in the dataset is also small. I did not remove any of the extreme value outliers because I felt it was likely the the extreme values could be useful in machine learning. This belief is partially based on prior knowledge of the Enron scandal, namely, that the people with the highest salary and bonuses (among other relevant features) were engaged in fraud.

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"]*

Initially, I manually reviewed the data to determine which features might be most useful. I determined that `from_poi_to_this_person` and `from_this_person_to_poi` might not be particularly useful as counts so I created `to_poi_ratio` and `from_poi_ratio` to represent the proportion of emails to to or from a poi. The `from_poi_ratio` was ranked as the 10th best feature by the SelectKBest algorithm, but the other proportion was not selected.

After I created the new features, I was concerned that several features had too few observations to be reliable. For example, `loan_advances` had only 3 observations. Rather than peremptorily excluding this feature, I left it in and let the SelectKBest algorithm determine whether it had better predictive value than other features.

This chart lists the features selected using SelectKBest with their scores:

Rank	Feature	Score
1	'exercised_stock_options'	24.815079733218194
2	'total_stock_value'	24.182898678566879
3	'bonus'	20.792252047181535
4	'salary'	18.289684043404513
5	'deferred_income',	11.458476579280369
6	'long_term_incentive'	9.9221860131898225
7	'restricted_stock'	9.2128106219771002
8	'total_payments'	8.7727777300916756
9	'loan_advances'	7.1840556582887247
10	'expenses'	6.0941733106389453
11	'from_poi_ratio'	5.1239461527568899

I tried both the MinMaxScaler() and the StandardScaler() when testing KNearestNeghbors, but not the other algorithms. I did not use scaling for my final choice of Adaboost.

For each of my final 4 algorithms I began by testing each with the top 11 features in the chart above. I then reduced the number of features to determine how that impacted precision and recall. The chart below shows the results of the 4 algorithms run with different numbers of features. The chart starts with 8 features because that is where I found the first model that met the minimum criteria of precision and recall above 0.3. The lines highlighted in green indicate models that met the minimum criteria. The cells highlighted in red indicate the highest precision and recall scores.

Algorithm	KBest Features	Accuracy	Precision	Recall

Decision Tree	8	0.80333	0.24653	0.23100
Naive Bayes	8	0.84100	0.38355	0.31700
Adaboost	8	0.80733	0.25305	0.22800
K Neighbors	8	0.86387	0.46974	0.16300
Decision Tree	7	0.78400	0.21041	0.18600
Naive Bayes	7	0.84671	0.45660	0.38400
Adaboost	7	0.79429	0.22940	0.18650
K Neighbors	7	0.84986	0.40860	0.11400
Decision Tree	6	0.79179	0.24312	0.21650
Naive Bayes	6	0.84714	0.45679	0.37000
Adaboost	6	0.79964	0.26558	0.22800
K Neighbors	6	0.85364	0.44731	0.10400
Decision Tree	5	0.79486	0.24856	0.21550
Naive Bayes	5	0.85464	0.48876	0.38050
Adaboost	5	0.80321	0.26940	0.22050
K Neighbors	5	0.85293	0.43867	0.10550
Decision Tree	4	0.79615	0.32602	0.30450
Naive Bayes	4	0.84677	0.50312	0.32300
Adaboost	4	0.80177	0.33910	0.30400
K Neighbors	4	0.86408	0.69983	0.20400
Decision Tree	3	0.80062	0.36258	0.39050
Naive Bayes	3	0.84300	0.48581	0.35100
Adaboost	3	0.82377	0.44640	0.42150
K Neighbors	3	0.86277	0.68685	0.19850
Decision Tree	2	0.76838	0.23353	0.22150
Naive Bayes	2	0.84069	0.46889	0.26750
Adaboost	2	0.77154	0.22506	0.19850

K Neighbors	2	0.84623	0.50120	0.10400

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

My final algorithm was Adaboost with Decision Tree. This algorithm had the highest recall and had sufficiently high precision. As can be seen in the chart above, I also tried Decision Tree on its own, K Nearest Neighbors and Naive Bayes among others that failed to meet the minimum criteria. I found that Naive Bayes performed the most consistently on varying lengths of features. With Adaboost, however, it was possible to improve on the recall, which I feel is the most important metric concerning this data.

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"]*

Tuning is the process of choosing appropriate parameters to maximize your desired result given your specific situation (e.g., size of data, available hardware, and time available for processing). If you don't tune your parameters well, the algorithm could take too long to run. On the other hand, some algorithms allow you to more fully process data, which would be appropriate for smaller datasets or where time is less of an issue. Finally, you would tune parameters to give you the desired recall, precision and accuracy for the problem.

For all four algorithms, I used GridSearch to tune the parameters. Specifically, for Adaboost, I `algorithm` parameter to `SAMME`, `learning_rate` to 1.5 and `n_estimators` to 40. For the underlying Decision Tree algorithm, I similarly used the GridSearch results for that algorithm.

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"]*

There are a couple of ways to validate your results. One way is to split the data into training and testing sets. Using this method, you would train your model on the training data alone and then test the model against the testing data. This method can work well if you are sure that the data is being split in a way that does not bias the results. Also, it is very important to ensure that you keep the testing and training data

separate. It would be very easy to accidently test on the training data or train on the testing data through a simple typo. This would cause the model to be overfit to the data.

Another way to validate is by using KFold or ShuffleSplit. These methods do away with the necessity of setting aside data to test on which allows you to use all of the data to train and test the model. Additionally, accidentally mixing up the training and testing data is less likely because, once you set the algorithm, it performs the splitting, training and testing automatically. The testing script for this project uses StratifiedShuffleSplit, a combination of the KFold and ShuffleSplit methods, which splits the data into new training and testing sets 1,000 times and runs the POI Identifier on each of these sets.

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"]*

For reference in this section, here are the top accuracy metrics from algorithms and the accuracy of the baseline model that is introduced below:

	Classifier	**Accuracy**
1	GaussianNB	0.85464
2	Decision Tree	0.80333
3	Adaboost	0.82377
4	K Neighbors	0.86408
5	Baseline	0.87671

Accuracy is a common metric which benefits from being very simple to compute. It is simply the proportion of correct predictions to total items in the dataset. For instance, the data in this project has 146 people and 18 are POIs. The Baseline model, an algorithm that always predicts the most frequent outcome (not a POI), has an accuracy of 0.8767 and this accuracy is higher than that of any of the other algorithms represented above. However, this basic algorithm would never predict that someone is a POI which is the entire point of this project. Therefore, this algorithm would fail in predicting every important datapoint. This extreme example demonstrates the shortcomings of using accuracy alone as a metric.

	Classifier	Precision
1	GaussianNB	0.50312
2	Decision Tree	0.36258
3	Adaboost	0.44640
4	K Neighbors	0.69983
5	Baseline	undefined

The chart above represents the top precisions of the tested algorithms and the Baseline. Adaboost is highlighted green in this chart because it represents my chosen algorithm. Precision is the ratio of true positives to the sum of true positives and false positives. In this case, it is the ratio of POIs that the algorithm correctly identified to the sum of POIs correctly identified and the people that the algorithm predicted to be a POI, but were really innocent. In the algorithms shown above, the Baseline's precision is undefined because it will never predict POIs, either correctly or incorrectly, so both the numerator and denominator will always be 0. The KNeighborsClassifier shows a significantly higher precision than the others. This precision explains that when a person is flagged as a POI, there is a probability of 0.69983 that the person is actually a POI. This shows how precision is a more nuanced validation metric than accuracy.

	Classifier	Recall
1	GaussianNB	0.38400
2	Decision Tree	0.39050
3	Adaboost	0.42150
4	K Neighbors	0.20400
5	Baseline	0

The chart above represents the top recall scores for the tested algorithms and the Baseline. Again, Adaboost is highlighted green because it represents my final algorithm. Recall is the ratio of true positives to the sum of true positives plus false negatives. In this project, it is the ratio of POIs correctly identified to the sum of POIs correctly identified and the POIs that the algorithm failed to identify (the proportion of actual POIs who were identified). Considering this, I believe this is the most important metric of the three discussed here. In a fraud investigation, you want to flag every

person who committed fraud, even if you end up investigating people who are innocent. Given this, I found it disconcerting that the recall rates were so low. The highest recall value only flags 42.15% of the POIs. The Baseline's score of 0 reflects that it will not identify any POIs because it simply predicts that there are no POIs. This extreme example further demonstrates that a high accuracy alone an insufficient metric for this problem.