Enron Submission Free-Response Questions

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 about 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 evaluator looks at 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**] Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that part of the rubric. If your response does not meet expectations for all rubric points, you will be asked to revise and resubmit your project. Make sure that your responses are detailed enough that the evaluator will be able to understand the steps you took and your thought processes as you went through the data analysis.

Once you've submitted your responses, your coach will take a look and may 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.
   1.1. The goal of this project was to determine the likelihood of guilt among those working at Enron at the time of its demise. By doing so, I would be solving a real world problem while gaining experience with the tools used by data scientists throughout the world. The dataset I was given contained a large number of features. There were a total of 146 data points, representing employees, each with a total of 21 features. Among the employees, there were a total of 18 'people of interest,' or 'POI', which the machine learning algorithm had to eventually recognise. Finally, there were a number of data points whose feature values did not exist; these were labeled 'NaN', which would be ignored during training.
   1.2. In the process of recognising outliers, I first transformed my data from varying magnitudes to those between 0 and 1. I did this by calculating the minimum and maximum value for each feature and using that to transform the data into percentages with equal weighting. This was done in part to see the data more clearly, but it was also done in case I would eventually use an algorithm that would be affected by the lack of this precaution, such as the support vector machine. I then graphed my data points systematically, looking at every combination of features and determining whether there were any outliers that would affect the overall decision making of the machine. This took a long time to do, and a large number of data points were removed/replaced with 'NaN' with the risk of increasing bias and reducing accuracy. However, this would be worth the

potential overfitting if I hadn't done so, and this step was ultimately crucial in the proficient scores of my algorithm.

2. There were three major ways that I selected my features. The first process was by hand, after rescaling my data points as described in part (1). After looking at each plot, I determined whether the provided features contained enough valid points to be useful, and I looked at the correlation between features to determine whether a machine learning algorithm could accurately find a pattern in the data. In doing so, I removed restricted_stock_deferred, loan_advances, director_fees, and deferral_payments. By the time I was done, the initial features amounted to salary, total_payments, bonus, total_stock_value, expenses, deferred_income, expenses, exercised_stock_options, long_term_incentive, shared_receipt_with_poi, and restricted_stock. I then created two of my own features, diving from_poi_to_this_person by to_messages to create percent_poi_to_person, and dividing from_this_person_to_poi by from_messages to create percent_person_to_poi. I had initially included those four separate features in my list, but I realized that they were correlated and not doing so would harm algorithms such as Naive Bayes. After doing all of this, I deployed a PCA algorithm to combine my features while retaining a large amount of information, and I then used SelectPercentile to get rid of any features that created noise. My parameter values for these algorithms were decided through GridSearchCV and, while changing, they generally are: PCA n_components=9, SelectPercentile percentile=33. The scores for my features as calculated by SelectPercentile are: 26.999264466850914, 6.186849459027546, and 5.974327943625211.
3. The algorithm that I ended up using was Naive Bayes. I chose this algorithm after testing a number of potential candidates and choosing the one with the best performance. The algorithms that I tried using were: Naive Bayes, SVM, Decision Tree, Random Forest, and AdaBoost. Out of these, the two with the best performance were Naive Bayes and Random Forest. Naive Bayes had a lower accuracy than Random Forest (0.86320 vs 0.87173). However, the recall and precision that resulted from the Naive Bayes algorithm vastly outperformed the others, with a recall value of 0.40000 and a precision value of 0.48426, while the Random Forest had a recall value of 0.20950 and a precision value of 0.54987.
4. Tuning the parameters of an algorithm means systematically changing the variables that affect how the algorithm works as a whole, including the frequent tradeoff between variance and bias. Too much variance can result in overfitting and lower reliability when testing the algorithm on multiple datasets, while too much bias can result in lower precision and may ignore important but subtle patterns in the data. Not tuning parameters well can result in an imbalance of these two characteristics which is unwanted. The way that I tuned my parameters, as stated above, was through a GridSearchCV, which systematically chose the parameters to use at the cost of increased runtime. The parameters that I tuned were the number of components created by the PCA, the percentage of features to include in my algorithm through SelectPercentile, and the number of estimators used by my random forest algorithm. The

potential values for n_components were: 5, 7, 9, 11, 13. The potential values of percentile were: 33, 50, 75, 100. The potential values of n_estimators were: 1, 3, 5, 10, 15, 20, 50, 100.

5. Validation is a method of determining the proficiency of an algorithm. This is done via splitting the data points into those meant for training and for testing, and by determining the accuracy and other metrics of the algorithm in identifying the test data while only being given the training data. One classic mistake that can be made is by forgetting to split these data points, resulting in an artificially high accuracy and f1 score. Additionally, if one splits their data points in such a way that only one type of data falls on one side and another type on the other, the algorithm could potentially get 100% or 0% accuracy; this was shown in the lesson covering this topic. In order to split my data, I used the train_test_split method with a 30% test set size, which gets rid of the potential for the latter possibility of only having one type of data in my test points. I then created a prediction based on my machine learning algorithm for the labels that would be assigned to each point and compared it to the actual labels, producing an accuracy score and an f1 score. My accuracy score was 0.8863636363636364 and my f1 score was 0.5454545454545454.
6. The 2 evaluation metrics that I used to evaluate my algorithm were precision and recall. Precision corresponds to the percentage of algorithmically identified POIs that actually were POIs, while recall corresponds to the percentage of actual POIs that were accurately identified by the algorithm. These two metrics are more important than accuracy in that they aren’t affected by a difference in size between label groups. Since the number of POIs were much smaller than the number of employees that were innocent, these two metrics are incredibly important in the proficiency of my algorithm. The average precision of my algorithm was 0.48426 and the average recall of my algorithm was 0.40000. Both are above the 0.3 threshold.