Machine Learning Project - Response Questions

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 utilize a publicly available Enron dataset to build a person of interest (POI) identifier. A POI is someone who committed fraud or might have committed fraud. The dataset has financial and email history for 145 people involved with the case, and 18 are initially identified as POIs because they were indicted, reached a settlement or plea with the government, or testified in exchange for prosecution immunity. The question is, are there additional POIs in the dataset? Machine learning will be used to find patterns in the data that could indicate the person was involved in fraud. For instance, if POIs tend to have higher salaries, are there any people not initially identified as a POI with a high salary that could also be a POI.

   The dataset initially contains 146 records, and each record is for a person. Reading through the names, two stand out as unusual. One of the names is “TOTAL”. Since the financial data was pulled from a “Payments to Insider” report that includes a row for totals, one of the 146 records in the dataset is that total. The total record is removed, leaving 145 people. Second, the name Travel Agency in the Park seems unusual. A footnote on the report states that this entry relates to Ken Lay’s sister, Sharon Lay, who is a co-owner of the agency that received payments. Since this entry could be a person of interest, it will be kept.

   There are 21 features for each employee, including the initial POI indicator. 14 of the features are financial in nature, and 6 are related to emails. For each person, features can have NaN as a value, meaning the value is missing. For the email features, 76.6% have email addresses, and 59% have other email data. If a person has data for one of the non-email-address email features, they have data for all 5 of the non-email-address email features. The financial features range from 86% to 2% with values. Comparing the “Payments to Insider” report to the dataset, zeros on the report are NaNs in the dataset. None of the people in the dataset have all NaNs for their financial data, so it appears all people are represented on the report, and therefore all NaNs are zeros.

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.
   [relevant rubric items: “create new features”, “intelligently select features”, “properly scale features”]

I ended up using the 'bonus' feature in the Decision Tree algorithm after trying multiple algorithms and feature combinations. To select the feature, I first manually ran the algorithm on different features and noted the precision and recall of each. I also tried a few combinations of features, then tried using SelectKBest to pick features. After trying several combinations of features based on the SelectKBest score, pvalue, and selected features, 'bonus' by itself returned the best precision and recall. Since the 'bonus' feature provided the best precision and recall above the 0.3 requirement, I selected it for the final list. Feature scaling was not used since Decision Trees are not affected by scaling.

I created a new feature called poi_mutual_correspondence that reflects if a person is sending a similar number of emails to POIs as they receive from POIs, and multiplies that proportion by the number of email sent to POIs. The idea is that the more emails a person sends to POIs when they are receiving a similar number of responses could indicate a common interest. The new feature initially looked promising. Out of the 86 samples with email data, 33 had lower values on the new feature before encountering the first POI. This made me think a decision tree could easily identify 38% of people as non-POIs, and in combination with another feature it could improve results. Actual results showed a decrease when combined with the 'bonus' feature. Feature importance showed 'bonus' with 0.664, and 'poi_mutual_correspondence' with 0.336.

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"]

I ended up using the Decision Tree algorithm, but also tried Gaussian Naïve Bayes, SVM and Adaboost.

Decision tree had a noticeably better result than GaussianNB. SVM didn't seem to work on this dataset; received a divide by zero when running the classifier through test_classifier due to a lack of true positive predictions. Adaboost showed reduced performance compared to the Decision Tree it was applied.

Decision Tree – Accuracy: 0.79456 Precision: 0.56255 Recall: 0.33950 F1: 0.42345 F2: 0.36874
DT with Adaboost - Accuracy: 0.74200 Precision: 0.39394 Recall: 0.29900 F1: 0.33997 F2: 0.31414
GaussianNB – Accuracy: 0.78867 Precision: 0.57980 Recall: 0.17800 F1: 0.27238 F2: 0.20664

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? What parameters did you tune? (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).
[relevant rubric items: "discuss parameter tuning", "tune the algorithm"]

Most algorithms have parameters that can be adjusted based on the dataset. Tuning parameters simply means adjusting the parameters for the best result. Badly tuned parameters will result in poor results with either high bias (over simplification) or high variance (overfitting).

On the Decision Tree algorithm, I tuned the min_samples_split parameter by trying settings from 2 to 20 and selecting the setting that gave the best precision. When I applied Adaboost to Decision Tree, I tried adjusting the learning_rate parameter along with the min_samples_split parameter. For SVM, I tried many combinations of C and gamma, but could not get a prediction.

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 items: "discuss validation", "validation strategy"]

   Validation is the process of applying sample test data with known labels to the classifier to see how accurate the classifier predicts classifications. Validation gives an estimate of performance on an independent dataset and serves as a check on overfitting. A classic mistake when doing validation is to use the same data to validate that was used to train the classifier.

   To validate my results, I first show accuracy, precision, recall, F1 and F2 using the train_test_split method, and then show the same metrics using the StratifiedShuffleSplit method provided in the tester.py module. Interestingly, the results using train_test_split initially show precision and recall above 0.3 with less favorable StratifiedShuffleSplit results. I then adjusted the min_samples_split parameter to maximize results using StratifiedShuffleSplit, and the results for train_test_split dropped. The StratifiedShuffleSplit function provides train and test indices to split data in train and test sets and is more reflective of performance given the limited number of data points. With this fact in mind, I went with the StratifiedShuffleSplit method used in the tester.py module.

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"]

   The Decision Tree algorithm resulted in Accuracy of 0.79456, Precision of 0.56255, and Recall of 0.33950. Accuracy is the number of data points labeled correctly divided by the total number of data point. In this case, 79% of the data points were labeled correctly as POI or non-POI. Precision shows how many of the people labeled as POI actually are a POI. In this case, 56% of the identified POIs were actually a POI. Recall is the number of POIs that were recalled as a POI. In this case, 34% of the POIs were recalled correctly.