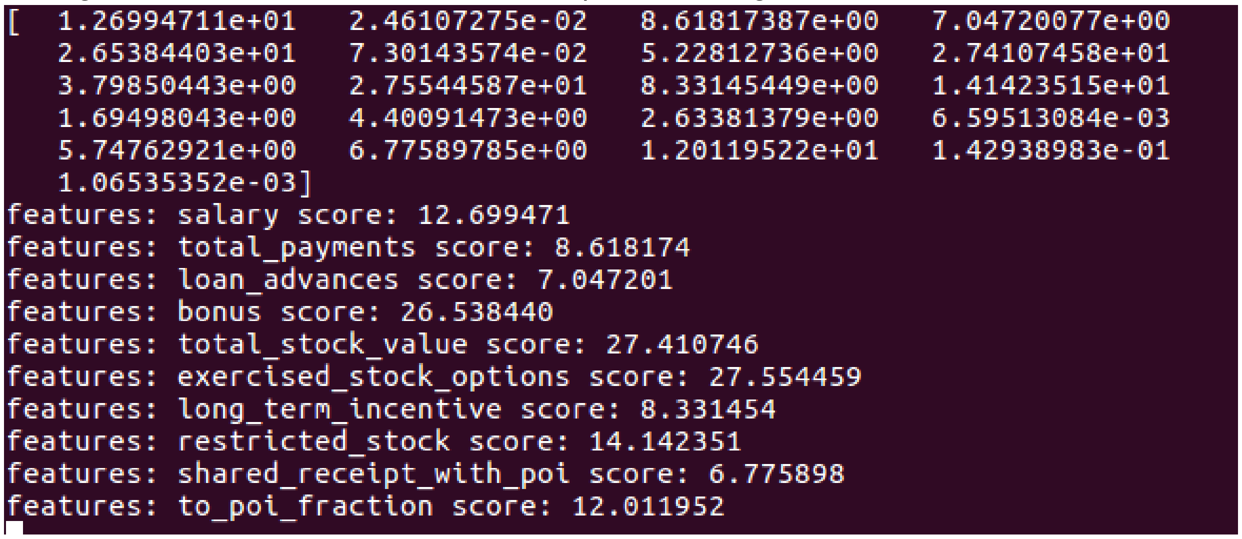
1. The goal of this project was to develop a machine learning algorithm (utilizing the Sci-kit Learn library) that is able to find person(s) who had committed fraud at Enron. Machine learning is a wonderful tool to utilize for such a problem since it is able to apply mathematical algorithms to huge sets of interconnected data in order to sort and organize it, which is essentially what we did here. We had our algorithm take certain features of the data and use it to find two classes of people in the data – those who had committed fraud and those who hadn't. The data is available because of it all being released publicly after people at Enron were sued for committing fraud at a large scale, and has become a popular dataset filled with information about salaries, bonuses, emails and who they were to and received from, etc. Such a rich dataset lends itself well to machine learning applications. And there were certainly outliers identified in the data, primarily by utilizing the 'matplotlib' library to graph our data and visually identify outliers. These outliers were 'TOTAL' which summed all the values, and 'THE TRAVEL AGENCY IN THE PARK' which is an odd entry and not a real person.

2. The features I ended up using were "['poi', 'salary', 'total_stock_value', 'expenses', 'bonus', 'exercised_stock_options', 'to_poi_fraction', 'from_poi_to_this_person', 'from_poi_fraction', 'shared_receipt_with_poi']", of which two were created by me - 'to_poi_fraction' and 'from_poi_fraction' and the rationale was that people who likely committed fraud likely had a lot of contact with other people that committed fraud, much more so than the general populace of employees. I had also created 'salary_bonus_fraction' as I thought that people who had a tiny fraction of their salary relative to bonus most likely had a higher chance of committing fraud, since their bonuses were from fraud. However, by using SelectKBest, I found that this was not a good feature to include. Here is the output from using SelectKBest:



```
[  1.26994711e+01   2.46107275e-02   8.61817387e+00   7.04720077e+00
   2.65384403e+01   7.30143574e-02   5.22812736e+00   2.74107458e+01
   3.79850443e+00   2.75544587e+01   8.33145449e+00   1.41423515e+01
   1.69498043e+00   4.40091473e+00   2.63381379e+00   6.59513084e-03
   5.74762921e+00   6.77589785e+00   1.20119522e+01   1.42938983e-01
   1.06535352e-03]
features: salary score: 12.699471
features: total_payments score: 8.618174
features: loan_advances score: 7.047201
features: bonus score: 26.538440
features: total_stock_value score: 27.410746
features: exercised_stock_options score: 27.554459
features: long_term_incentive score: 8.331454
features: restricted_stock score: 14.142351
features: shared_receipt_with_poi score: 6.775898
features: to_poi_fraction score: 12.011952
```

I just used the standard parameters, since I achieved good results using the top 10 features, which is the default for SelectKBest. I also did use a MinMaxScaler as normalizing the data before training stops the algorithm from assigning more weight to features that may have a much larger range, and in essence standardizes all the features so they are equal numerically.

3. I ended up using a AdaBoost Decision Tree, with which I got the best results. I also tried Random Forest, Logistic Regression, a normal Decision Tree, and Gaussian Naive-Bayes classifiers. The other results in terms of accuracy were – normal Decision Tree (85%), Gaussian Naive-Bayes(79%), Random Forest(86%), and finally Logistic Regression (76%).

4. Tuning the parameters of an algorithm is essentially changing the values of different constants and such throughout the equation, and choosing the good values for these is critical for good performance from a machine learning algorithm. If you don't tune these properly, you may get poor results from your classifier since the parameters don't fit the data well. For my final classifier I tuned were 'max_depth', 'min_samples_leaf', and 'class_weight' (for the Decision Tree inside the AdaBoost) and 'n_estimators' and 'learning_rate' (for the AdaBoost classifier).

5. Validation is critical to testing the performance of a trained machine learning algorithm, in order to see how well it generalizes the dataset. If done incorrectly, a classic mistake the the problem of overfitting, where the algorithm doesn't generalize the data enough and is too 'sticky' to the data points, leading to poor results when assessing performance. I split the data using StratifiedShuffleSplit. For my validation, I used Precision and Recall values to see how well my model was generalizing.

6. I used the F1-Score and the accuracy score for my evaluation. The accuracy is a simple analysis the the amount of data points the algorithm classified correctly divided by the total amount of data points classified, where getting as close to 1 as possible is the goal. And the F1-Score is a weighted average of the precision (correct positive results divided by all positive results) and recall (positive results divided by positive results that should have been returned), and the goal is again to get as close to 1 as possible.

   Here are scores of two trainings I had done:
   Test 1:
   Accuracy: 87.1% F1-Score: 0.35824
   Test 2:
   Accuracy: 86.5% F1-Score: 0.35764