Summary of the Project

The goal of the project is to create a machine learning classifier that can accurately predict whether or not a person was involved with the widespread fraud in Enron. Machine learning is useful in accomplishing this because it may be able to pick up characteristics and quirks that make up the profile of a person of interest (hidden in the data) that the human eye might not be able to find.

This dataset had 3066 data points (146 individuals and 21 features) and had 18 persons of interest (POI). I tested up to 6 features (one of them generated by myself) in this project, but found that the best classifier I found only needed about 3 features. There are some features with many missing values (i.e. long_term_incentive, more than half of the data points were not a number) and there was one major outlier that was removed, “TOTAL”, which was not a person of interest but merely a summation of the features of the other individuals.

Feature Selection

The features that I ended up using in the classifier were expenses, total_payments, and bonus, all of which I selected by hand. I did not use feature scaling, because the algorithm I ended up using (decision tree classifier) is insensitive to feature scaling.

I created a new feature that was called “networth”. It was "salary" plus "long_term_incentive" minus "expenses" minus "total_payments"… this was used to try to make a feature that combined all of the relevant salary, benefits, and expenses information from the other features. In the final product I did not use it because of a low feature importance. The features I tested and their importance in the decision tree classifier is listed below.

```
Features: 'salary','networth'

Accuracy: 0.76755       Precision: 0.18171      Recall: 0.07950 F1: 0.11061     F2: 0.08958
        Total predictions: 11000        True positives:  159    False positives:  716   False negatives: 1841   True negatives: 8284

Feature Importance: [ 0.4445512  0.5554488]


Features: 'salary','expenses','long_term_incentive','networth'

Accuracy: 0.76518       Precision: 0.27866      Recall: 0.18350 F1: 0.22128     F2: 0.19695
        Total predictions: 11000        True positives:  367    False positives:  950   False negatives: 1633   True negatives: 8050

Feature Importance: [ 0.08299988  0.34748196  0.10930805  0.46021011]


Features: 'salary','expenses','long_term_incentive'.'total_payments','bonus','networth'

Accuracy: 0.81823       Precision: 0.35422      Recall: 0.22050 F1: 0.27180     F2: 0.23851
        Total predictions: 13000        True positives:  441    False positives:  804   False negatives: 1559   True negatives: 10196

Feature Importance: [ 0.          0.33089087  0.          0.13045173  0.53865739  0.        ]


Features: 'expenses', 'total_payments', 'bonus'

Accuracy: 0.83238       Precision: 0.44346      Recall: 0.35100 F1: 0.39185     F2: 0.36627
        Total predictions: 13000        True positives:  702    False positives:  881   False negatives: 1298   True negatives: 10119

Feature Importance: [ 0.33089087  0.13045173  0.53865739]
```

Algorithm Choice

I ended up using a decision tree classifier as my choice classifier. I tried out an AdaBoost classifier as well but found that the decision tree classifier was slightly more precise (i.e. better precision). The statistics for AdaBoost as compared to my decision tree classifier's performance is below.

```
ADABOOST
Features: 'expenses', 'total_payments', 'bonus'

Accuracy: 0.83323       Precision: 0.44574      Recall: 0.34500 F1: 0.38895     F2: 0.36133
        Total predictions: 13000        True positives:  690    False positives:  858   False negatives: 1310   True negatives: 10142

Feature Importance: [ 0.54  0.2   0.26]
```

Parameter Tuning

Parameter tuning is the process of systematically changing parameters to an algorithm to have it better fit the data. If you don't tune the parameters well, you can either underfit the data (i.e. the algorithm will barely match the data) or overfit the data (i.e. the algorithm will be well fit to the training set but not the validation or the testing set).

I ended up tuning the criterion, min_samples_split, and max_depth parameters on my decision tree classifier to be able to better fit the data. I did not make that much of a change to the criterion parameter (since "gini" and "entropy" are very similar) but I raised the min_samples_split and changed the max_depth to be 5 nodes. Even though I raised the min_samples_split and effectively lowered the max_depth (I restricted it to 5 nodes) the algorithm did not underfit the data.

Validation

Validation is the process of predicting and evaluating a classifier on two sets of data. One of the sets of data (the training data) is used almost exclusively for training the algorithm – the other set of data is used only to predict its corresponding labels and to be evaluated on such. A classic mistake when using validation is training the algorithm a second time on the validation/testing data which essentially creates a different instance of the classifier altogether (which makes for incomparable results between the training and testing evaluation). I validated my evaluation of the algorithm by splitting the features and labels into two groups, one to train on (and predict values for) and one used explicitly for predicting and comparing labels (or train/test split validation).

Evaluation Metrics

I evaluated my classifier with precision and recall among other metrics (e.g. accuracy). The precision of the algorithm was ~.44 and the recall was ~.35, which means the precision was better than the recall. That means that the algorithm was better at finding real examples of POI (and nothing else) rather than being better at pulling most of the examples of POI in the dataset (and potentially making mistakes).