1) The goal of this project is to utilize machine learning techniques in order to analyze a data set and build a predictive model. Specifically, we analyze the Enron data set which contains financial and email data from a number of employees of that former corporation which suffered bankruptcy due to employee fraud. Our model attempts to predict whether an individual in the data set is a person of interest (i.e. whether they may have committed fraud). Every individual in the data set possesses a boolean value indicating whether or not they are a POI and as such this is a supervised learning task. The problem of identifying POIs requires that we split our data into training and test sets in order to avoid overfitting and accurately verify our results. Because of the numerous features in the data set, machine learning will help to uncover the underlying basis for those people who are persons of interest. As we build our model, accuracy, precision, recall and F1 scores guide our decisions (primarily accuracy and F1). The benchmark model used to arrive at these scores is a Gaussian Naive Bayes Classifier.

An exploration of the data set revealed the number of instances to be relatively scant. There are 146 individuals of which only 18 are persons of interest. There are 21 features (including the target), with financial features:

['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees']

email features:

['to_messages', 'from_poi_to_this_person', 'email_address', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']

and target: ['poi']

The features have various levels of missing/non-existent values, and several of the email features contain information about interactions with POIs constituting data leakage. Because of this I chose to exclude all of the email features for the remainder of the project except for 'email_address' which is later used to engineer features from the email corpus. The email corpus was also deficient of data, for example Andrew Fastow, who was arguably the head chef of cooking Enron's financial books, did not

have any email data and his communication with accountant Arthur Andersen would have proved to be valuable.

Before removing any outliers or individual features programmatically, I used the entire data set with all features (15 financial features) to get a benchmark score:

GaussianNB()
Accuracy: 0.33473 Precision: 0.15420 Recall: 0.88950 F1: 0.26284 F2: 0.45529
Total predictions: 15000 True positives: 1779 False positives: 9758 False negatives: 221
True negatives: 3242

To impute missing numerical and text data, I used a zero value (instead of an average) and empty string respectively. There were several outliers in the data set that were removed because they were either not a person but an organization, represented a cumulative total, or did not have enough relevant data. The first outlier removed was based upon the insiderpay.pdf file provided to us, and the organization 'THE TRAVEL AGENCY IN THE PARK' was removed because it is not a person (and not POI). By plotting 'total_payments' by 'total_stock_value' I was able to identify the 'TOTAL' outlier in the data set. This was also removed because it is not a person. Finally, I chose to remove all individuals without any financial data. There was only one person, 'LOCKHART EUGENE E', who was not a POI. The scores slightly decreased after removing the outliers but the amount was negligible:

GaussianNB()
Accuracy: 0.32793 Precision: 0.15279 Recall: 0.88900 F1: 0.26076 F2: 0.45272
Total predictions: 15000 True positives: 1778 False positives: 9859 False negatives: 222
True negatives: 3141

After removing all outliers I split the data into training and testing sets to ensure that the distribution of POIs was even. The training set contained 100 people with 13 POIs and the test set contained 43 people with 5 POIs. Both had roughly the same distribution of POIs to non-POIs (13% and 12%).

2) Feature selection and engineering was the most involved part of the project. I pruned financial features based upon available data, engineered text features from email data, and selected the best features using a combination of methods.

Reducing the number of features was done on the entire data set and naturally followed from the previous data exploration. The first step in pruning was to compare the amount of available data between features. I tallied the total number of non-nulls in each feature giving extra weight to POIs by their inverse proportion. I selected 8 features with the highest scores to keep:

> ['total_stock_value', 'total_payments', 'restricted_stock', 'expenses', 'other', 'salary', 'bonus', 'exercised_stock_options']

and excluded the rest with limited data:

> ['deferral_payments', 'loan_advances', 'restricted_stock_deferred', 'deferred_income', 'long_term_incentive', 'director_fees']

After reducing the number of features, accuracy and precision more than doubled. Recall decreased significantly although the F1 score remained nearly the same:

> GaussianNB()
> Accuracy: 0.84773 Precision: 0.36450 Recall: 0.19100 F1: 0.25066 F2: 0.21110
> Total predictions: 15000 True positives: 382 False positives: 666 False negatives: 1618
> True negatives: 12334

Text features were engineered based upon training data as the process involved a model to predict written email words indicative of a POI. The features are a numerical count of the frequency of a particular word used by every individual. The features were created by aggregating text from a specified number of emails, stemming each word (linguistic root), computing term frequency-inverse document frequency (TF-IDF) of all words for all people in the data set, fitting a decision tree model using the TF-IDF to predict POIs, extracting the most important words from the model, and using those important words to count how many times each was written by a person in the original text aggregate. The important words vary based upon the number of emails processed and the particular self-identifying and nonsense words that are excluded. I chose to exclude any words including numbers and those that I believed to be self identifying or nonsense. After processing 100 emails per person, the words that I consistently encountered were:

[u'blown', u'boardroom']

I did not go through the emails to see if any of these were in fact self-identifying, so it remains possible that they are. 'Boardroom' may have been a place were meetings took place to discuss things that were not meant to be documented, and 'blown' may have been a way to describe the situation of how finances were managed as things came to an end. After adding these text features, accuracy, precision, and recall all improved:

GaussianNB()
Accuracy: 0.88667 Precision: 0.61244 Recall: 0.40850 F1: 0.49010 F2: 0.43765
Total predictions: 15000 True positives: 817 False positives: 517 False negatives: 1183
True negatives: 12483

The feature selection process involved a combination of Pearson correlation analysis, recursive feature selection, and tree-based classifiers. Each step added a rank to the importance of each feature and a cumulative total was used to select the best ones. I reduced the number of features to 6 and the following were consistently the best:

['total_stock_value', u'boardroom', 'bonus', u'blown', 'exercised_stock_options',
'restricted_stock', 'salary']

After reducing the number of features to these 6, precision and recall significantly improved:

GaussianNB()
Accuracy: 0.88377 Precision: 0.69057 Recall: 0.44300 F1: 0.53975 F2: 0.47722
Total predictions: 13000 True positives: 886 False positives: 397 False negatives: 1114
True negatives: 10603

3) Among several classifiers that were tested, k-nearest-neighbors and adaBoost were the most promising. K-nearest-neighbors requires range scaling in order to preprocess data and so this was used in a pipeline along with a min-max scaler. Using this pipeline but without any parameter specifications (5 nearest neighbors by default), the k-nearest-neighbor classifier scored:

Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=5, p=2, weights='uniform'))])
Accuracy: 0.85515 Precision: 0.62012 Recall: 0.15100 F1: 0.24286 F2: 0.17792
Total predictions: 13000 True positives: 302 False positives: 185 False negatives: 1698
True negatives: 10815

This was a decline over the benchmark classifier but I decided to keep it, as tuning parameters in the next step might allow this to outperform the benchmark. The adaBoost classifier was initially used outside of a pipeline and scored:

AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None)
Accuracy: 0.85207 Precision: 0.47666 Recall: 0.36250 F1: 0.41181 F2: 0.38074
Total predictions: 14000 True positives: 725 False positives: 796 False negatives: 1275
True negatives: 11204

I decided to use a pipeline and combine adaBoost with a standard scaler and principle component analysis. It performed slightly better with these than alone (although still a decline compared to the benchmark) but previous successes with this classifier and the promise of parameter tuning kept it in my repertoire:

Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('reducer', PCA(copy=True, n_components=None, whiten=False)), ('ada', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None))])
Accuracy: 0.86057 Precision: 0.51786 Recall: 0.34800 F1: 0.41627 F2: 0.37243
Total predictions: 14000 True positives: 696 False positives: 648 False negatives: 1304
True negatives: 11352

4) Parameter tuning is an important part of the model building process as it allows us to optimize

our model's machinery. In some cases the parameters are very sensitive to the nature of the data and only certain values will allow the model to perform well at all. A slight tweak in the model can mean the difference from an average fit to a great one. In order to tune these parameters for an optimal fit, I used a grid search and cross-validated over a stratified shuffle split on the training portion of the data only (again all model generation was done with the same training set). After doing this with k-nearest-neighbors, the scores were:

Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=1, p=2, weights='uniform'))])
Accuracy: 0.84907 Precision: 0.47193 Recall: 0.47500 F1: 0.47346 F2: 0.47438
Total predictions: 14000 True positives: 950 False positives: 1063 False negatives: 1050
True negatives: 10937

At this point I manually removed the 'salary' feature after some experimentation and the model increased it's performance significantly:

Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_neighbors=1, p=2, weights='uniform'))])
Accuracy: 0.87631 Precision: 0.62596 Recall: 0.48700 F1: 0.54781 F2: 0.50963
Total predictions: 13000 True positives: 974 False positives: 582 False negatives: 1026
True negatives: 10418

I similarly tuned the parameters for the adaBoost classifier which resulted in the following scores:

Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('reducer', PCA(copy=True, n_components=None, whiten=False)), ('ada', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=DecisionTreeClassifier(compute_importances=None, criterion='gini', max_depth...dom_state=None, splitter='best'), learning_rate=1.0, n_estimators=50, random_state=None))])
Accuracy: 0.82557 Precision: 0.38131 Recall: 0.35500 F1: 0.36769 F2: 0.35997

Total predictions: 14000 True positives: 710 False positives: 1152 False negatives: 1290
True negatives: 10848

Removing the 'salary' feature as in k-nearest-neighbors, performance of the adaBoost classifier's precision and recall improved:

Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)),
('reducer', PCA(copy=True, n_components=None, whiten=False)), ('ada',
AdaBoostClassifier(algorithm='SAMME.R',
base_estimator=DecisionTreeClassifier(compute_importances=None, criterion='gini',
max_depth...om_state=None, splitter='best'),
learning_rate=10.0, n_estimators=50, random_state=None))])
Accuracy: 0.84738 Precision: 0.50404 Recall: 0.49900 F1: 0.50151 F2: 0.50000
Total predictions: 13000 True positives: 998 False positives: 982 False negatives: 1002
True negatives: 10018

Both of the algorithms that I chose accomplished their best accuracy and F1 scores after parameter tuning and final feature pruning. K-nearest-neighbors performed best when it used either 1 or 2 nearest neighbors. Adaboost performed best at learning rate much higher (10 or 20) than the default of 1.

5) Validation is the process by which we evaluate our model's performance. It allows us to verify whether or not to remove particular outliers, include some features, and use one model over another. To do this we often train the model on a portion of the entire data set and test on another (just as we have developed all predictive capabilities of our model on a training set only). By testing our model on unseen data we can see how well it performs in a 'real-world' situation. Because of the limited size of our data set, and in order to accurately test our model's performance, we use a stratified shuffle split. This allows us to randomly select train/test portions of the entire data set, fitting the model on the training set and testing on the test set to generate evaluation scores. With stratified shuffle split, this random selection is done numerous times, and each time the proportion of POIs is preserved across both the train and test sets.

6) The benchmark gaussian naïve bayes classifier was the most accurate model and with the highest precision score:

GaussianNB()
Accuracy: 0.88377 Precision: 0.69057 Recall: 0.44300 F1: 0.53975 F2: 0.47722
Total predictions: 13000 True positives: 886 False positives: 397 False negatives: 1114
True negatives: 10603

The k-nearest-neighbor classifier was very close in accuracy and had the highest F1 score of all the models:

Accuracy: 0.87631 Precision: 0.62596 Recall: 0.48700 F1: 0.54781 F2: 0.50963
Total predictions: 13000 True positives: 974 False positives: 582 False negatives: 1026
True negatives: 10418

The adaBoost classifier had the highest recall score although it's accuracy and F1 scores were lower than the other two:

Accuracy: 0.84738 Precision: 0.50404 Recall: 0.49900 F1: 0.50151 F2: 0.50000
Total predictions: 13000 True positives: 998 False positives: 982 False negatives: 1002
True negatives: 10018

Accuracy is the proportion of times that we're able to correctly predict the class of all observations. In terms of our data set, it means that if you give us a person, our model can correctly predict whether or not they are a POI some proportion of the time. Precision represents the proportion of times we are correct when our model has chosen a positive classification. In other words, when we evaluate somebody and our model says that they are indeed a person of interest, we are correct some proportion of the time.

I would argue that recall is probably the most important metric in terms of our data set. Recall represents the proportion that the model can correctly identify a positive classification, given it is has one. In the context of this data, set it represents how often we can identify a POI when we are actually looking at one. Ultimately, each of the 3 classifiers had their strengths and weaknesses, which one is

considered best depends upon the evaluation metrics that are most important.