Summary of Data Set

Goal

The goal of this project was to identify "Persons of Interest" in the highly publicized fraud case against Enron. The data set given consisted of 146 observations and 19 metrics that described compensation and email statistics of employees. 14 of the metrics described how much a person was compensated and the other 5 described how many emails they sent and received.

Data Set Exploration

Each observation had an average of 9.03 missing data points with 58 of the 144 valid observations missing over half of the features. 3 of the features across all observations had fewer than 20 data points. Those features were 'restricted stock deferrals,' 'loan advances,' and 'director fees.' Needless to say, there were very few observations and the data set was riddled with missing data which tempered my expectations for model performance. There were also a number of variables that had a wide range between the min and max. An example was 'restricted stock,' which had a minimum of -2,604,490 and a maximum of 14,761,694. Of the 146 observations, only 18 were "Persons of Interest." A very small percentage of the observations are POIs making it an unbalanced data set. The unbalanced nature of the training set is an important consideration later on in the analysis when we are testing and validating our model.

Outliers

There were 2 invalid observations that I found in the data, 'TOTAL' and 'THE TRAVEL AGENCY IN THE PARK.' I removed them from the data set as neither of these observations described a person and could not be considered a "Person of Interest." I also implemented a z-score univariate outlier detection method to eliminate extremes within the dataset. Preliminary test runs showed that this actually decreased the performance of the algorithms. For this data set, I believe that outliers can play an important part in identifying POIs. For example, Kenneth Lay (CEO and Chairman of Enron) was certainly a POI, however, was eliminated for most features on the basis he was an outlier (i.e. exercised stock options) which most likely hurt the accuracy by removing a valid observation.

Feature Selection and Engineering

Custom Features

I created 5 different ratios that were derived from a number of the given features. I attempted to eliminate volume and magnitude from financial and email statistics. The custom features were:

- Messages sent to POI vs. Total Messages Sent
- Messages received from POI vs. Total Messages Received
- Bonus Received vs. Total Payments
- Expenses vs. Total Payments
- Salary vs. Total Payments

These ratios are evenly scaled so that a very large number alone would not skew predictions between observations. After numerous preliminary trials with the custom ratios, I found that only for the "Tree"

classifier algorithms did the performance improve when adding any custom features. The feature "Messages sent to POI vs. Total Massages Sent" increased the performance of the Decision Tree Classifier, Gradient Boosting Classifier, and Random Forest Classifier; as can be seen in the test run results below. For the other algorithms, using any of the features had a negligible difference.

Algorithm	Included Custom Features	Accuracy	Precision	Recall	F1	F2
DecisionTreeClassifier	TRUE	82.1%	32.0%	30.8%	31.4%	31.0%
DecisionTreeClassifier	FALSE	79.5%	22.1%	21.1%	21.6%	21.3%
GradientBoostingClassifier	TRUE	86.1%	46.5%	28.1%	35.0%	30.5%
GradientBoostingClassifier	FALSE	85.5%	41.7%	22.7%	29.4%	25.0%
RandomForestClassifier	TRUE	85.9%	40.7%	12.6%	19.2%	14.6%
RandomForestClassifier	FALSE	85.4%	36.3%	12.3%	18.4%	14.2%
LogisticRegression	TRUE	77.6%	17.8%	18.8%	18.3%	18.6%
LogisticRegression	FALSE	77.6%	17.8%	18.7%	18.2%	18.5%
GaussianNB	TRUE	33.6%	14.8%	83.4%	25.1%	43.2%
GaussianNB	FALSE	33.5%	14.7%	83.5%	25.1%	43.2%
KNeighborsClassifier	TRUE	87.6%	60.3%	21.6%	31.8%	24.8%
KNeighborsClassifier	FALSE	87.6%	60.3%	21.6%	31.8%	24.8%

NOTE: Tests runs with and without custom features used all original features of the data set. Parameters remained the same between runs of the same algorithm.

Feature Selection

After I had the universe of features for my dataset, I used the Decision Tree Classifier to determine which features had the most predictive power. For each run using different feature sets, I recorded the significance of each feature. They can be found in the excel file "ML Project 4 – Run Log" under the "DecisionTreeSignifiance" tab. After multiple test runs with different features, it was obvious which features had the most significance as they continued to receive the highest weights. This was confirmed when I compared the performance of algorithms with the features selected vs. all of the variables or random subsets. Of the ratios that I derived, only one (Messages sent to POI vs. Total messages sent) was deemed significant.

Algorithmic Feature Selection

I also chose to test K-Select Best and Randomized Principal Component Analysis for intelligent feature selection. For each classifier I used, I tested both selection algorithms with different parameters (k for KSelectBest and n_components for PCA). For most all of the classifiers, however, I found that manual selection of inputs using the Decision Tree Significance outperformed the feature selection algorithms

Feature Scaling

There was only a few instances where scaling was required. When I tested the K-Nearest Neighbors and K-Means Clustering I had to use feature scaling because these algorithms calculate the nearest neighbors using a distance metric between observations. This metric would be dominated by any feature with a much larger or wider scale, therefore, it is important to scale all features to an equivalent

scale. I also chose to test how scaled features would affect the performance of the other algorithms I tested. It was also important to use feature scaling when testing Principal Component Analysis as PCA measures the plane of highest variance which would similarly be dominated by features with the biggest scales.

The two methods of feature scaling I used were Min-Max Scaling and Z-Score Scaling. Min-Max scaling simply standardizes all of the observations to fit within a scale of 0 to 1. Z-Score scaling assumes normality of the data set and normalizes observations so that the mean of each feature is 0. Z-Score places a much higher importance on outliers, as there is no upper or lower bound and values that are multiple standard deviations from the mean will remain as such. Min-Max scaling downplays the significance of outliers as all values are restricted to a range of 0 to 1.

Algorithm Selection and Tuning

Algorithm Selection

I tested 8 different algorithms on the data set with a combination of outlier detection, feature selection, and scaling methods. I recorded the results within the "PrelimRunLog" tab of the excel file. I chose to do little to no parameter tuning at this stage as I wanted to see which algorithm would produce the most consistent results without attempting to fit it to the data. I believed that this would help see which ones are by default a better fit. The only model that I was not able to get results for was the Support Vector Machine, which was not surprising as SVMs usually require few features and many observations to make meaningful predictions. The other algorithms that I had tried were:

- Decision Tree Classifier
- Gaussian Naïve Bayes
- Logistic Regression
- K Nearest Neighbors
- K Means Clustering (not really valid for this data set as it is unsupervised and attempts to create its own classes)
- Gradient Boosting Classifier
- Random Forest Classifier

After I completed all of the test runs and log the results, I put them into a pivot table to see which ones had the best average performance which I measured using the average accuracy, max precision and max recall. I chose the most performant algorithms that had parameters to be tuned. Those algorithms were:

- Decision Tree Classifer
- K Nearest Neighbors
- Gradient Boosting Classifier

Algorithm	# of Runs	Average of Accuracy	Max of Precision	Max of Recall	Average of F1	Average of F2
GaussianNB	11	72.04%	54.63%	94.40%	30.47%	32.26%
GradientBoostingClassifier	7	83.08%	55.04%	35.75%	29.82%	26.68%
DecisionTreeClassifier	12	79.25%	42.06%	40.40%	29.74%	29.64%
RandomForestClassifier	9	84.80%	58.71%	22.70%	23.88%	19.16%

KNeighborsClassifier	45	86.01%	91.77%	39.75%	22.57%	17.67%
Kmeans	16	77.15%	28.00%	52.40%	19.79%	20.12%
LogisticRegression	8	82.45%	100.00%	18.75%	11.62%	8.82%

NOTE: I chose to exclude Gaussian Naïve Bayes despite solid performance because it lacked parameters to tune.

Parameter Tuning

There is always a tradeoff between how well you would like your model to fit the training set (variance) and how generic you would like to make it so that you only fit the signal within the data and not the noise (bias). Parameter tuning allows the user to define the bias-variance tradeoff they believe is optimal for future predictions. Tuning also allows the user to customize the algorithm to fit the data they are trying to fit. For example, a movie recommendation engine that uses K Nearest Neighbors would benefit from using a custom measure of proximity since Euclidean distance is too generic for the problem at hand.

For tuning the Decision Tree Classifier, I used the following parameters: Criterion, Max Features, Min Samples Split, Max Depth, Min Samples Leaf, and Max Leaf Nodes.

For tuning the K Nearest Neighbors Classifier, I used the following parameters: Number of Neighbors, Weights, Algorithm, and Leaf Size.

For tuning the Gradient Boosting Classifier, I used the following parameters: Learning Rate, Number of Estimators, Max Depth, Min Samples Split, Min Samples Leaf, Subsample, and Warm Start.

For each algorithm, I also used GridSearchCV in order to test a number of different parameter combinations.

Evaluation

Metrics Used

I chose to use the sum of F1 and F2 scores to assess the performance of each algorithm. I used them rather than simply precision and recall because abnormally high values for precision or recall could skew the measurement depending on the magnitude of one. In other words, a precision of 0.9 and recall 0.1 would indicate a combined score of 1. A precision of 0.5 and recall of 0.5 would achieve the same score, however, is much better performance for this task since it balances the number of POIs identified correctly with the overall percentage of POIs identified.

Recall measures the number of POIs the algorithm was able to identify out of all the POIs that were in the data set. Precision measures how accurately the algorithm was able to identify a POI when it guessed an observation was a POI. The F1 score is the harmonic mean of precision and recall, weighting both precision and recall equally. The F2 score weights recall higher than precision.

Gradient Boosting Classifier Performance

The most performant algorithm I found was the Gradient Boosting Classifier with Warm Start set to true. This algorithm gave the following performance:

Accuracy	**Precision**	**Recall**	**F1**	**F2**	**F1+F2**

98.45%	94.82%	94.30%	94.56%	94.40%	188.96%

I found this performance remarkable. After conducting sensitivity analysis, however, I became skeptical of the results. I tested a number of other parameter combinations using GridSearchCV with Warm Start set to true. The performance was not nearly high as that of the other test run:

Accuracy	Precision	Recall	F1	F2	F1+F2
84.70%	45.47%	35.65%	39.97%	37.26%	77.23%

This led me to the conclusion that this could be a case of overfitting of the data, which I was very wary about. This data set had very few observations, making fitting difficult since the model could only be fit to a very small subset of the data. The need for bias when dealing with a very small data set is high as you would like to generalize the primary trends and not pay too close attention to the noise that could be specific to this subset only.

K Nearest Neighbors Performance

The most performant parameter combination I found for KNN was Number of Neighbors as 2 and Weight as "Distance" along with the data Min-Max Standardized. The performance was:

Accuracy	Precision	Recall	F1	F2	F1+F2
86.80%	56.80%	31.75%	40.73%	34.82%	75.55%

Decision Tree Classifier Performance

The Decision Tree Classifier had the most consistent performance of all the algorithms. I found that Criterion as "Entropy," Max Depth as 5, Min Samples Leaf as 12, and the data Z-Score standardized produced the best results with a performance of:

Accuracy	Precision	Recall	F1	F2	F1+F2
86.77%	53.48%	56.85%	55.11%	56.14%	111.26%

I chose to use this as my final algorithm. As I conducted sensitivity analysis, I noticed that the performance was consistently high giving me confidence that this wasn't a case of overfitting. Due to this, I believed that he Decision Tree was the best model for this particular data set. Using GridSearchCV and a different values for the same parameters used in the test above, the algorithm achieved a performance of:

Accuracy	Precision	Recall	F1	F2	F1+F2
85.18%	48.05%	46.20%	47.11%	46.56%	93.67%

Validation

Validation of the model is critical as it allows you to have confidence your model does not overfit the data and the performance results of the algorithm are consistent. Validation forces you to split the data set up into subsets of training and testing observations; where the model is trained on a subset of the data and tested another portion of the data it has yet to encounter allowing you to assess how the model will perform when making future predictions on unknown observations. Since the data was unbalanced and small for this project, it required a cross validation method that would iterate over the data set multiple times and train/test on different parts of the data to arrive at an accurate and reliable performance metric.

The model was validated in the tester.py script utilizing stratified shuffle split. This cross validation technique randomly split or “folded” the data multiple (1000) times into different training/testing sets. Each training/test pairing was used to train and test the model. Predictions through all of the iterations were tallied and labeled as True Positive, True Negative, False Positive, and False Negative. After all iterations were done, the final tallies were used to calculate the precision, recall, accuracy, F1 and F2 scores for the model. Therefore, stratified shuffle split allowed the data set to be tested many different times on different subsets of the data overcoming issues of a small and unbalanced data set.