Enron Dataset: Identifying Persons of Interest Using Machine Learning

Usman Rizwan

August 19, 2016

1 Enron Scandal

Enron was once the seventh largest company in America. In August 2000 Enron share prices peaked at \$90.75. However by the end of 2001 the share prices had dropped to below \$1. On December 2nd, 2001 Enron declared bankruptcy.

Top Enron officials started dumping their shares prior to the precipitous drop in share prices. Restrictions were placed on lower-level employees that prevented them from selling their stock. Many lower-level employees lost their life savings as a result.

Many top-level Enron employees were brought to trial and convicted. Despite large amounts of information on the internal workings of the organization released into the public domain, no smoking gun evidence was found. It is still worthwhile to investigate the Enron data set using machine learning algorithm to see if machine learning can help us find some hidden smoking gun in the data set that the human prosecutors failed to notice during the trials.

1.1 Project Overview

In this project we will look at the financial and email data set that was made public by the Federal Energy Regulatory Commission. We will use machine learning algorithm to identify persons of interest (POI) in the data set. A POI is someone who was indicted, settled or testified in exchange for immunity.

The data analysis will follow the following steps:

1. Clean up the data set.
2. Look at the presence of any outliers.
3. Try different machine learning algorithms.
4. Tune feature selection and introduce new features.
5. Validate the algorithm.

2 Important Concepts

2.1 Variance vs Bias

Variance arises because of sensitivity to small fluctuations in the training set. It leads to over-fitting, which means that the model performs well on the training set but poorly on previously unseen data, called the testing set.

Bias is error from erroneous assumption. It usually arises when trying to fit a complex training set with a simple model. This leads to under-fitting, which means the model doesn't perform well on the training or testing set. There is a trade off between variance and bias, trying to reduce one type of error leads to an increase in the other type of error.

2.2 Precision vs Recall

Precision and recall are defined as

$$\text{Precision} = \frac{\text{true-positives}}{\text{true-positives} + \text{false-positives}}$$

$$\text{Recall} = \frac{\text{true-positives}}{\text{true-positives} + \text{false-negative}}$$

A high-precision in this context means that few non-POI's were falsely classified as POI's by our algorithm while a high recall means that few POI's were mistakenly classified as non-POI's.

3 Cleaning the Data Set

Initially rhere are a total of 146 people in the data sets and each person is defined by 23 features. Only 18 people in the data set are classified as POI. The outliers were checked for by plotting all the features against salary. Samples are shown below.

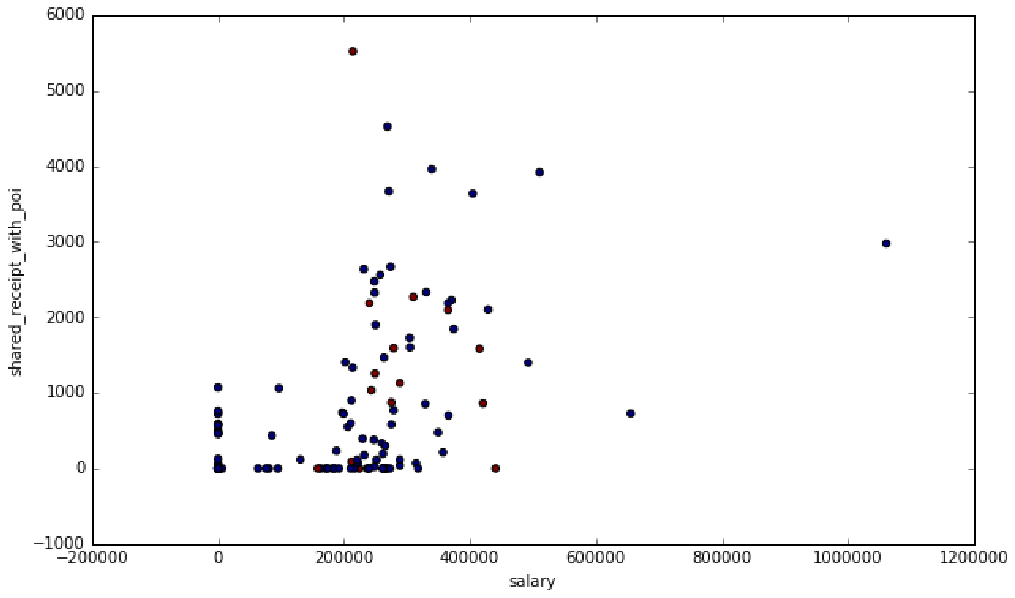

Figure 1: A plot of salary vs. shared receipts with POI. Blue point are non-POI's while red points are POI's.

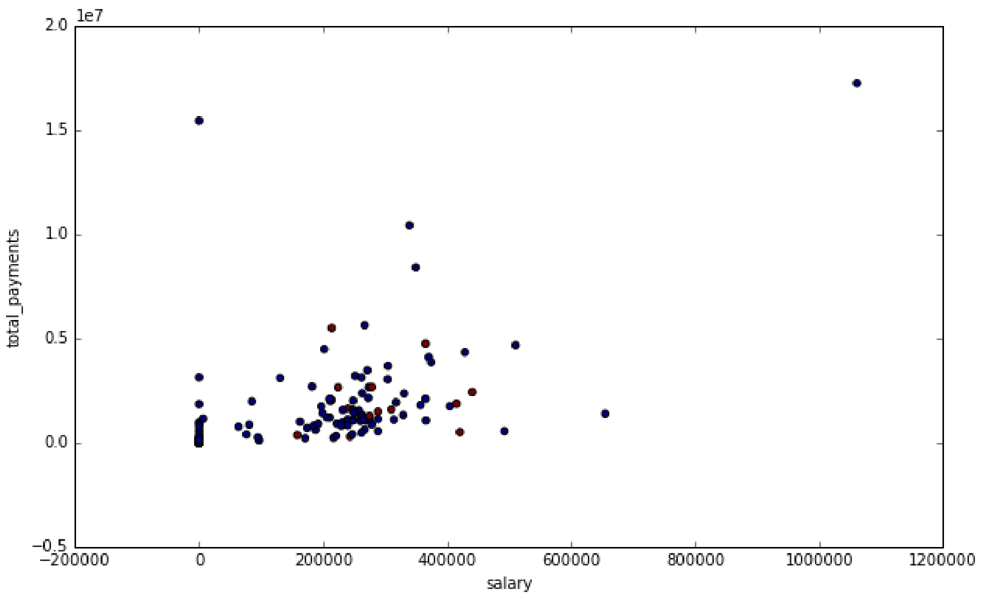

Figure 2: A plot of salary vs. total payments. Blue point are non-POI's while red points are POI's.

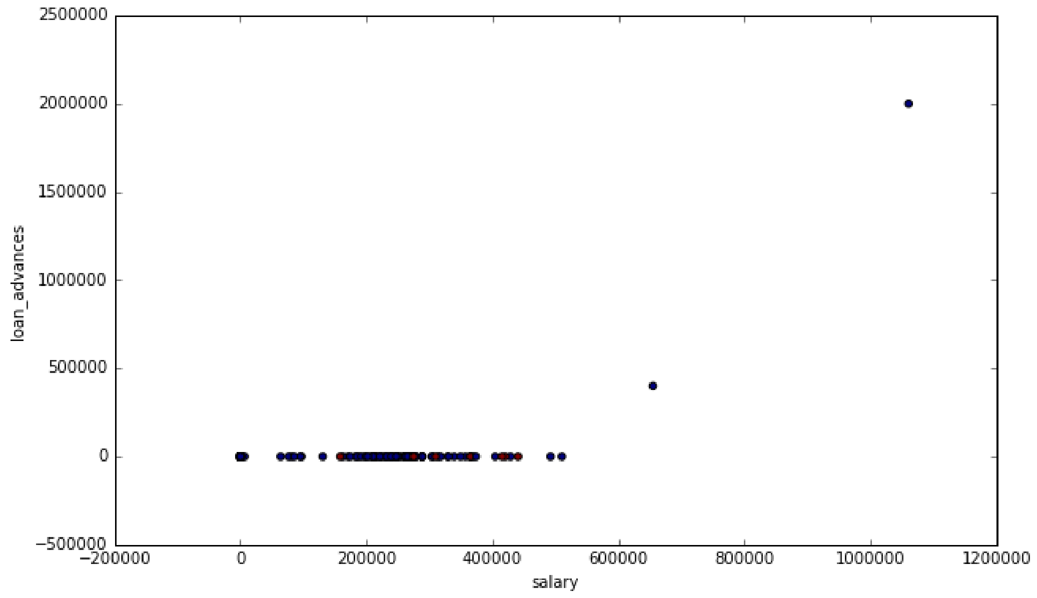

Figure 3: A plot of salary vs. loan advances. Blue point are non-POI's while red points are POI's.

There are 22 non-POI features in total but it seems for some features like loan advancement, deferred income, restricted and stocks differed we have no information at all. So in the plots above we see most of the points for these features concentrated at the y=0 line.

From the plots we see that there are some outlier salaries, bonuses, total payments, exercised stock options and others. On further inspection it seems that these outliers are due to TOTAL key in the data set. This key contains the sum of salaries, bonuses and other known payments given out to people in the data set. This feature is useless for our purposes so we will take it out of the dictionary.

```
### Store to my_dataset for easy export below.
my_dataset = data_dict
```

```
### Removing the TOTAL key
my_dataset.pop("TOTAL", None)
```

Rather than working with all the data set in our machine learning algorithm we will split the data set into a training and testing set. This is done to get an honest assessment of our predictive model. If we use the whole data set to build our predictive model and then test our predictive model on the same set, our model would just repeat the labels of the samples it was trained on. This would not give us a good way to judge how our predictive model would behave on future data.

We check the validity of our predictive model by training it on the training set and then testing it on previously unseen testing set. This will give a much better idea about how accurate our prediction model would on data that we haven't seen yet.

In the present case we will do a 80-20 split, i.e. 80% of the total data will be used for training and 20% will be used for testing.

```
features_train, features_test, labels_train, labels_test =
     train_test_split(features, labels, test_size=0.20, random_state=1)
```

4 Decision Tree Algorithm

The first algorithm we will use to build a classification model is the decision tree algorithm. Decision tree is a classification algorithm that builds a classification or regression models in the form of a tree structure. It breaks down a given set into smaller and smaller subsets while developing a decision tree incrementally. A decision tree is made up of decision nodes and leaf nodes. A decision node has two or more branches while a leaf node represents a classification (for e.g. True or False).

```
tr = DecisionTreeClassifier(random_state=1)
tr.fit(features_train, labels_train)
accuracy : 1.0
precision : 1.0
recall : 1.0
```

The out of the box accuracy of the decision tree algorithm is perfect on the training set. Lets see how the model performs on the test set.

```
pred = tr.predict(features_test)
accuracy : 0.897
precision : 0.0
recall : 0.0
```

The model does not generalize over to the test set. Precision and recall are both 0. Now lets see if feature selection can help us improve the accuracy, precision and recall. The feature selection method we will use is the recursive feature selection. In this method a model is trained on the initial set of features and weights are assigned to features. Features whose weights are the smallest are removed from the feature set and the procedure is recursively repeated on the pruned feature set until the desired number of features to select is eventually reached. By default the algorithm selects the half of the total number of features and runs the selected algorithm on these features.

```
#rank all features, i.e continue the elimination until the last one
rfe = RFE(DecisionTreeClassifier(random_state=0))
rfe.fit(features_train, labels_train)
Features used in the decision tree algorithm:
1. from_messages
2. restricted_stock
3. from_this_person_to_poi
```

```
4. bonus
5. director_fees
6. expenses
7. deferral_payments
8. restricted_stock_deferred
9. deferred_income
accuracy : 1.0
precision : 1.0
recall : 1.0
```

The algorithm selected 9 most important features from the training set and performed a fit on those features. It works perfectly on the training set. Let's see how it performs on the test set.

```
accuracy : 0.897
precision : 0.0
recall : 0.0
```

Again the model does not generalize well over to the test set. Its accuracy, precision and recall are still 0. It seems that the model over-fits on the training set and doesn't generalize at all to the test set. To work around this problem we will use K-fold cross-validation.

In K-fold cross-validation training data is split into K folds of equal size. Each K fold acts as the test set once and as the training set K-1 times. The advantage of K-Fold cross validation is that all the examples in the training data set are used for both training and testing. This provides us with a predictive model that has lower variance. This is particularly useful when we are dealing with a small data set and our classes are skewed (most of the people in the data set are not POI). There is actually a lot of variation in the performance of our algorithms depending on the partition of the data. Cross-validation helps reduce this variation by making the predictive model less sensitive to the partitioning of the data.

In this case we will use 5-fold cross validation and use precision instead of accuracy to measure the performance of the algorithm. This is because we are dealing with a skewed data set, so precision is a better measure of performance than accuracy. We will select the features that give the best precision for when trained on the training set and run the algorithm on the testing set.

```
rfecv = RFECV(estimator=DecisionTreeClassifier(random_state=0),
step=1, cv=5, scoring="precision")

rfecv.fit(features_train, labels_train)
Optimal number of features : 3
```

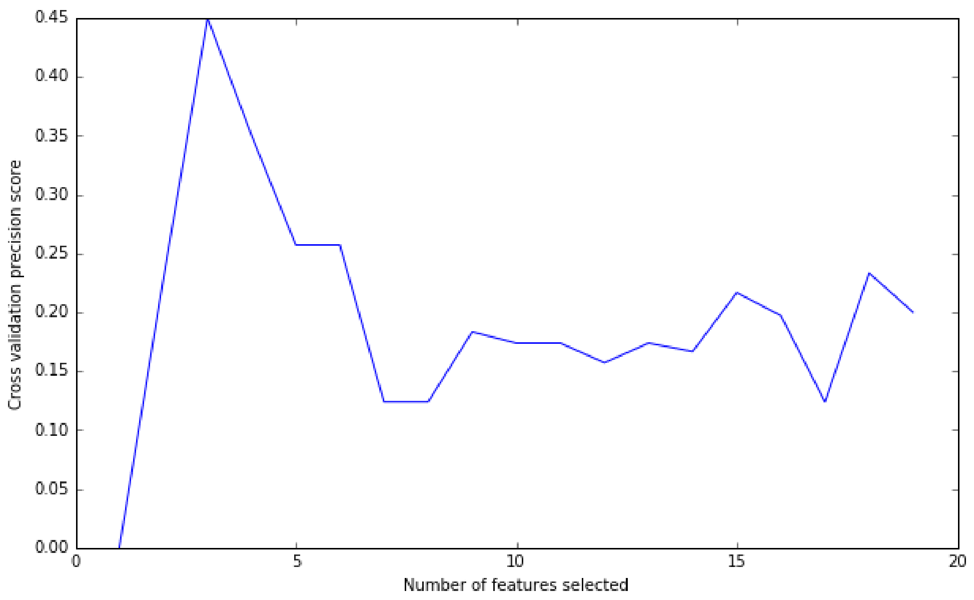

Figure 4: A plot of cross-validation precision score vs. number of features selected.

```
Features used in the decision tree algorithm:
1. bonus
2. expenses
3. deferred_income
```

```
pred = rfecv.predict(features_test)
accuracy : 0.828
precision : 0.0
recall : 0.0
```

There is no improvement in the performance of algorithm. It should be noted here that we are using the decision tree algorithm as it comes out of the box. We haven't optimized any of the parameters of the decision tree algorithm. So lets use the grid search method to optimize the decision tree algorithm on the whole data set first. The parameters that will be optimized are criterion (function to measure the quality of a split), min_samples_split (minimum number of samples required to split an internal node), max_features (the number of features to consider when looking for the best split at a node) and max_depth (the maximum depth of the tree).

```
param_grid = {"random_state": [1], "criterion":["gini","entropy"],
"min_samples_split": [10,20,30], "max_features": ["auto", "sqrt", "log2"],
"max_depth": [10,15,20]}

etc = DecisionTreeClassifier()

dtr = grid_search.GridSearchCV(etc, param_grid, scoring="precision")
dtr.fit(features_train, labels_train)

rfe = RFE(dtr.best_estimator_)
rfe.fit(features_train, labels_train)

Features used in the decision tree algorithm:
1. other
2. from_this_person_to_poi
```

```
3. long_term_incentive
4. bonus
5. director_fees
6. expenses
7. deferral_payments
8. restricted_stock_deferred
9. deferred_income
```

Let's see how the optimize decision tree algorithm with the 9 features above performs on the testing set.

```
pred = rfe.predict(features_test)
accuracy : 0.862
precision : 0.0
recall : 0.0
```

There is no improvement in the performance of our algorithm. Both precision and accuracy are still 0. Let us see if K-fold cross validation helps further improve the performance of the algorithm.

```
rfecv = RFECV(estimator=dtr.best_estimator_, step=1, cv=5, scoring="precision")

rfecv.fit(features_train, labels_train)

Optimal number of features : 6
Features used in the decision tree algorithm:
1. from_this_person_to_poi
2. long_term_incentive
3. bonus
4. expenses
5. restricted_stock_deferred
6. deferred_income
```

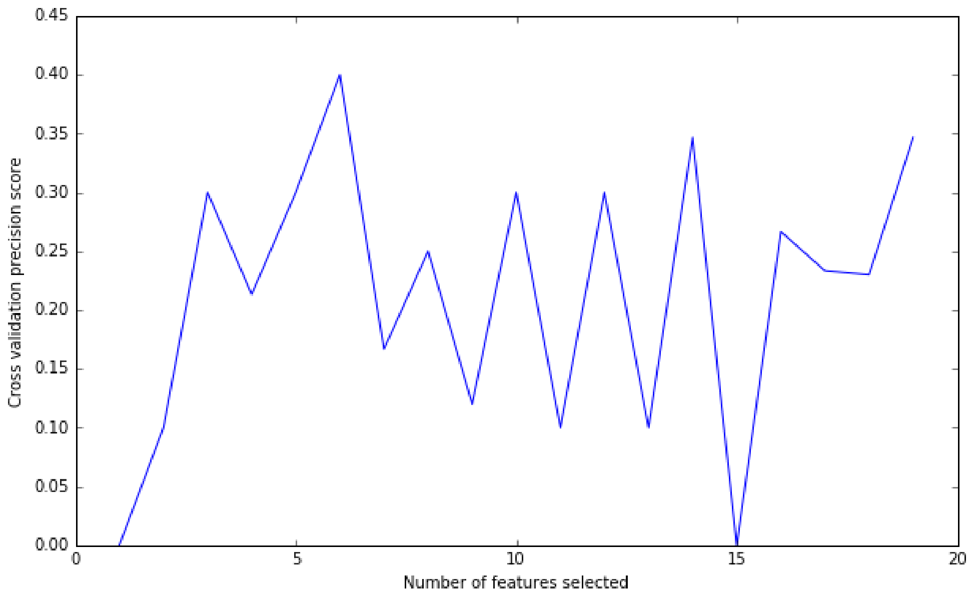

Figure 5: A plot of cross-validation precision score vs. number of features selected.

Let's see how the optimize decision tree algorithm with the 6 features above performs on the testing set.

```
pred = rfecv.predict(features_test)
accuracy : 0.862
precision : 0.0
recall : 0.0
```

Again we see no improvement in the performance of our algorithm. The precision and recall are both still 0.

Now, let us try and add some features to our feature set. Reading about the background of the Enron scandal and trials it seems that prosecutors noticed that some of the Enron officials weren't big e-mail users while others used it excessively. If people weren't big e-mail users than it is likely people didn't sent them that many e-mails either. It is reasonable to normalize the amount of emails received from POI to the total number of e-mails received and similarly it is reasonable to normalize the amount of e-mails sent to POI to the total numbers of e-mails sent. Lets use 5-fold cross-validation to optimize the parameters of the decision tree algorithm on the new feature set.

```
etc = DecisionTreeClassifier()

param_grid = {"random_state": [1], "criterion":["gini","entropy"],
"min_samples_split": [10,20,30], "max_features": ["auto", "sqrt", "log2"],
"max_depth": [10,15,20]}

dtr = grid_search.GridSearchCV(etc, param_grid, scoring = "recall", cv=5)
dtr.fit(new_features_train, new_labels_train)
Optimal number of features : 9
Features used in the decision tree algorithm:
1. salary
2. loan_advances
3. director_fees
4. expenses
5. deferral_payments
6. restricted_stock_deferred
7. deferred_income
8. from_poi_ratio
9. to_poi_ratio
```

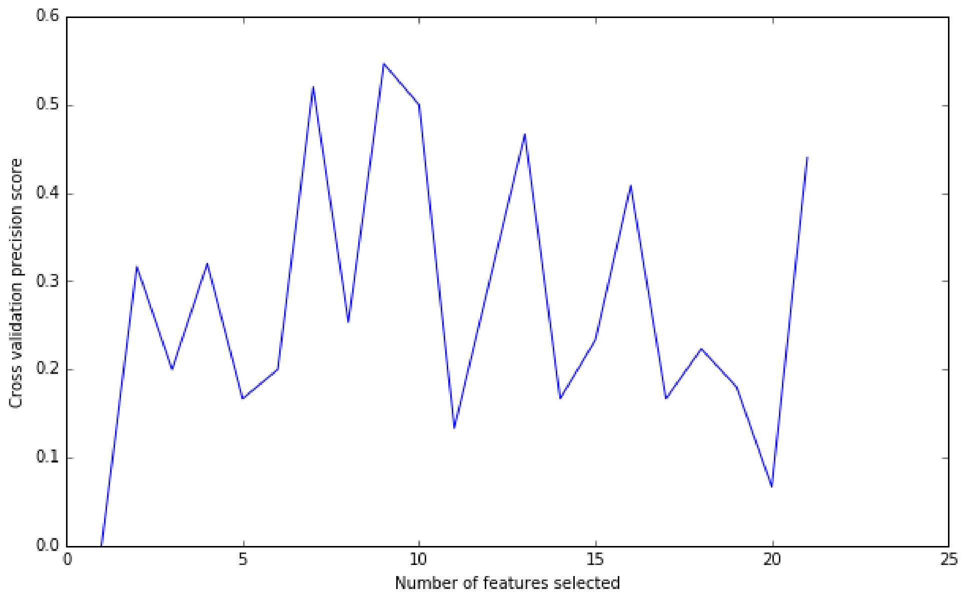

Figure 6: A plot of cross-validation precision score vs. number of features selected.

Let's see how the optimize decision tree algorithm with the 9 features above performs on the testing set.

```
accuracy : 0.793
precision : 0.0
recall : 0.0
```

With the addition of the new features into our feature set there is still no improvement in the performance of the algorithm on the test set. Let us try another algorithm to see if we can reach better results.

5 K Nearest Neighbour algorithm

Let us now move onto another machine learning algorithm, the K Nearest Neighbour (K-NN) algorithm. K-NN is a simple algorih where a point case is classified by a majority vote of its neighbours i.e. the point is assigned to the most common type of point amongst its K nearest neighbours. We will try to use 1 to 20 nearest neighbours to classify a point to see if larger or fewer neighbours help improve the algorithm. As previously we will use precision to decide what number of neighbours performs best.

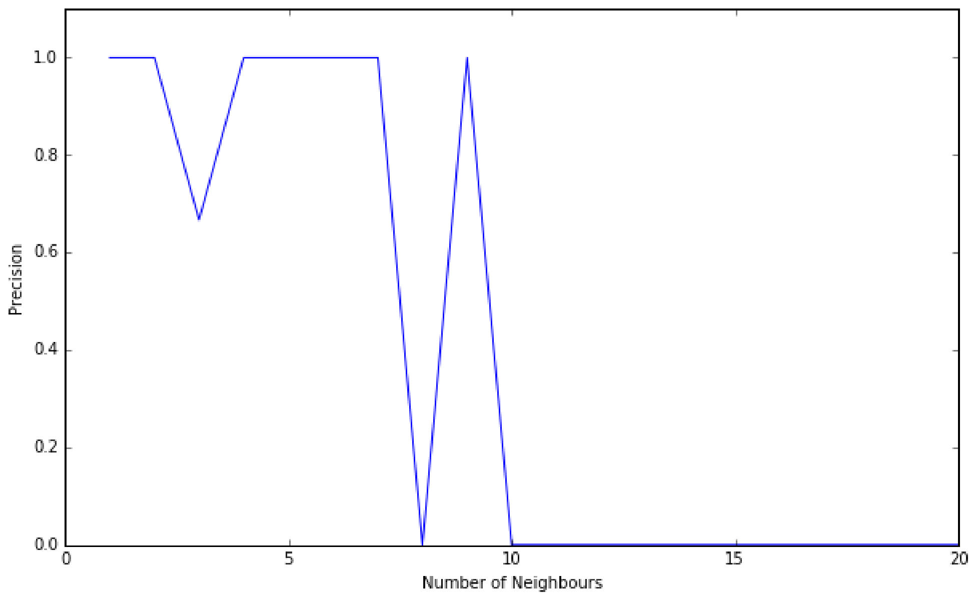

Figure 7: A plot of precision score vs. number of nearest neighbours used for classification. Algorithm run on the original feature set.

The maximum precision occurs at 1, 2, 4, 5, 6, 7 and 9. There is a bias-variance trade-off involved here, small values of K will produce a model with low bias and high variance while large values of K will produce a model with high bias and low variance. So we will go towards the middle and pick the best K as 6.

```
neighb = KNeighborsClassifier(n_neighbors = 6)
neighb.fit(features_train, labels_train)
pred = neighb.predict(features_test)
accuracy : 0.897
precision : 0.0
recall : 0.0
```

As seen from the results above the model doesn't generalize over to the test set. Let us now try the 5-fold cross-validation method to find the best number of neighbours for classification.

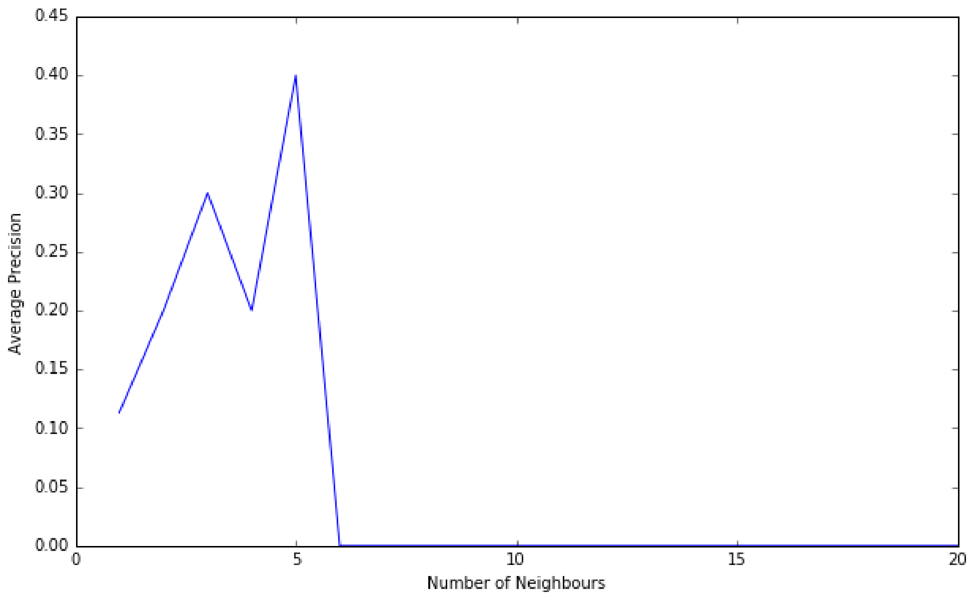

Figure 8: A plot of precision score vs. number of nearest neighbours used for classification. Algorithm run on the original feature set with 5-fold cross-validation.

According to the cross-validation method, 5 is the best nearest neighbour parameter for the K-NN algorithm. Let us try the K-NN algorithm with 5 nearest neighbour on the test set.

```
neighb = KNeighborsClassifier(n_neighbors = 5)
neighb.fit(features_train, labels_train)
pred = neighb.predict(features_test)
accuracy : 0.862
precision : 0.0
recall : 0.0
```

As seen from the results above the model doesn't generalize over to the test set. Lets try the K-NN algorithm on the new features set.

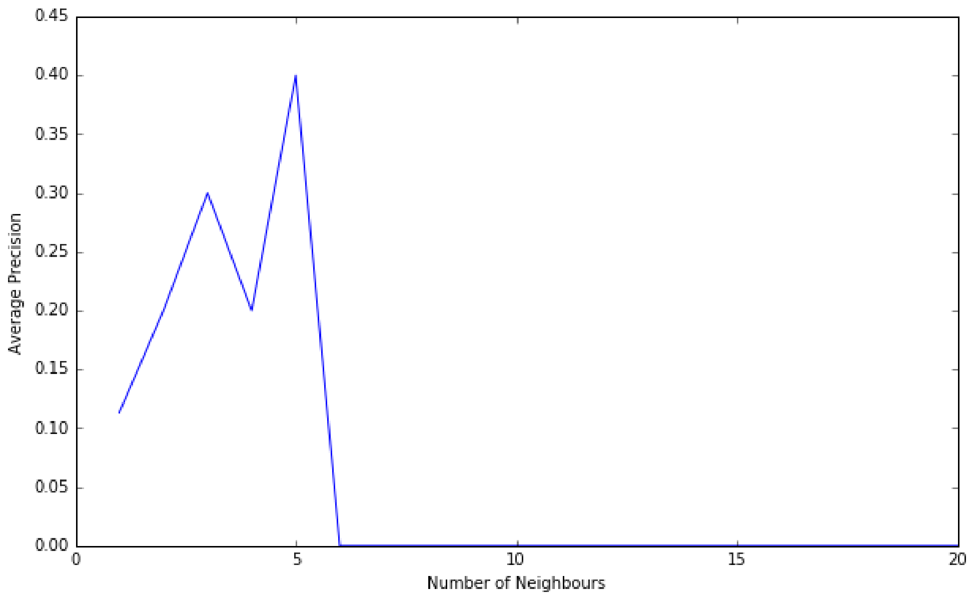

Figure 9: A plot of precision score vs. number of nearest neighbours used for classification. Algorithm run on the modified feature set with 5-fold cross-validation.

```
accuracy : 0.862
precision : 0.0
recall : 0.0
```

The algorithm still doesn't generalize from the training set to the testing set. It should be noted that we haven't scaled the data. In K-NN it is important to scale the data because you want the distances to be meaningful. In our case the number of e-mail received can be between 0 and 15000 while stock values are between 0 and 31 million. For the distances between the points evaluated in the K-NN algorithm to be meaningful it is important to scale the features. We will use scale function from sklearn to scale all the data point to between -10 and 10.

```
from sklearn import preprocessing

X_train = new_features_train
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(-10, 10))
X_train_minmax = min_max_scaler.fit_transform(X_train)
```

Let us now try the 5-fold cross-validation method on the scaled data set to find the best number of neighbours for classification.

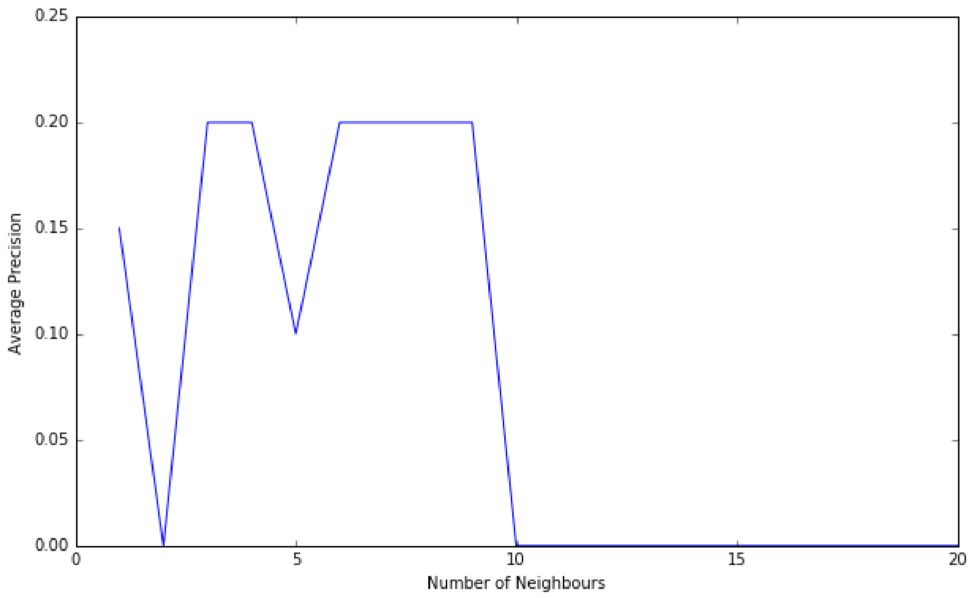

Figure 10: A plot of precision score vs. number of nearest neighbours used for classification. Algorithm run on the modified feature set and scaled features with 5-fold cross-validation.

```
accuracy : 0.862
precision : 0.0
recall : 0.0
```

Let's do some variance threshold feature selection to remove features that do not have a lot of variation. We will try a couple of variance threshold to see which one gives the best performance in terms of precision on the training set.

The variance of the scaled features varies between 2.88 and 21.9. Let's do some feature selection.

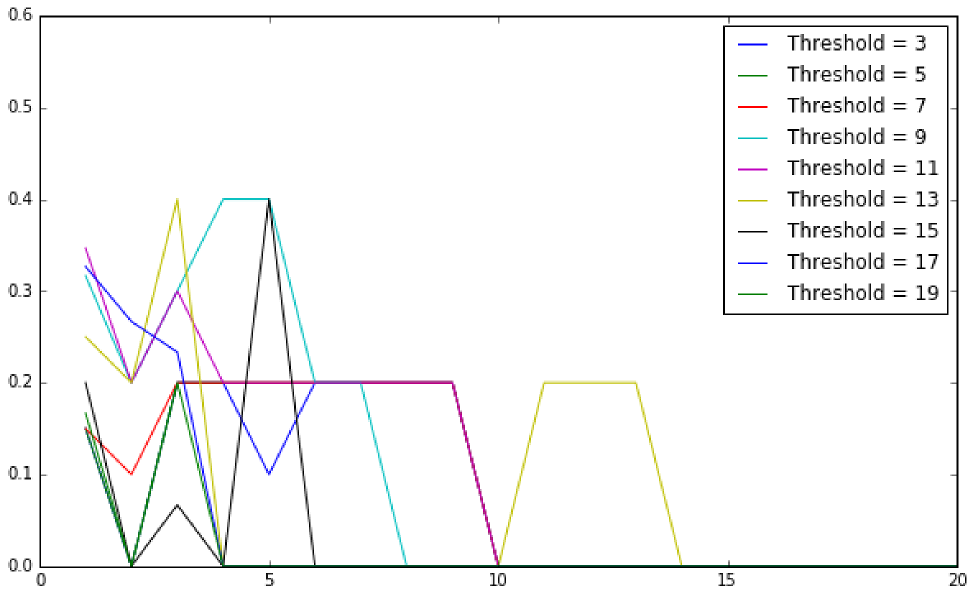

Figure 11: A plot of precision score vs. number of nearest neighbours with a variety of thresholds used to remove low variance features. Algorithm run on the modified feature set and scaled features with 5-fold cross-validation.

We get the best precision when threshold is 15 and the value of newarest neighbour parameter is 5. Let's use these parameters to see how well the algorithm performs on the test set.

```
sel = VarianceThreshold(threshold=15)
X_train_var = sel.fit_transform(X_train_minmax)
X_test_var = sel.transform(X_test_minmax)

neighb_var = KNeighborsClassifier(n_neighbors = 5)

pred = neighb_var.predict(X_test_var)
Features used in the K-NN algorithm:
1.  shared_receipt_with_poi
2.  salary
3.  director_fees
4.  expenses

accuracy : 0.897
precision : 0.0
recall : 0.0
```

The performance of the algorithm is not improved at all. It seems feature selection reduces the number of features and over-fits the data which leads to high variance. Therefore the algorithm does not perform well on the test set.

6 Manual Feature Selection

Both threshold feature selection and recursive feature selection have failed to improve the performances of either the K-NN or decision tree algorithms. It is now time to attempt some manual feature selection. The algorithms attempted in the previous sections provide a general idea of what the most important features are to classify POI's. After a bit of trial and error I have decided on the following 8 features, not including POI.

```
1. other
2. long_term_incentive
3. expenses
4. deferral_payments
5. restricted_stock_deferred
6. deferred_income
7. from_poi_ratio
8. to_poi_ratio
```

We will optimize the decision tree algorithm on the manually selected features and see how they perform on the training set:

```
accuracy : 0.953
precision : 0.875
recall : 0.636
```

Now lets see how the algorithm performs on the test set:

```
accuracy : 0.852
precision : 0.6
recall : 0.6
```

With some manual feature selection the decision tree algorithm performs reasonable well on the test set. A precision and recall of 0.6 means that if our algorithm correctly identifies 10 POI's correctly then it would also falsely identify 6-7 people falsely as POI's and similarly 6-7 actual POI's would be classified as non-POI's.

7 Conclusion

Two algorithms were tested on the data to see which algorithm performs better. Decision tree performed incredibly well right out-of-the-box on the training set but it's success didn't translate over into the testing set. Decision trees very much over fit the data and even recursive feature selection and K-fold cross-validation were not able to improve its performance. K-NN don't perform as well on the training set. Recursive feature elimination does not work with the K-NN algorithm because it is not possible to assign weights to features in the K-NN algorithm. Even with variance threshold feature selection and cross validation the performance of the K-NN algorithm did not improve. It seems that the data set is to sparse for the application of K-NN algorithm. The variance of features varies quite considerably, K-NN is not suited for this kind of data.

It was a bit of a surprise that the recursive feature selection and variance threshold feature selection were not able to pick the best features that generalized well over to the test set. Manual feature selection with optimized decision tree parameters was able to give much better results in the end.

The performance of the algorithm can still be improved by digging into the contents of the e-mails of the people present in the data set. The e-mails from POI to POI might have certain types of vocabulary which would set them apart from e-mails between POI and non-POI, and non-POI and non-POI.

8 References

Decision Tree Learning
K Nearest Neighbors - Classification
Model evaluation: quantifying the quality of predictions
sklearn.grid_search.GridSearchCV
Training a machine learning model with scikit-learn
Selecting the best model in scikit-learn using cross-validation
Enron Fast Facts
The Immortal Life of the Enron E-mails
Grid Search: Searching for estimator parameters
sklearn.feature_selection.RFECV
Cross-validation: evaluating estimator performance
Decision Trees
sklearn.neighbors.KNeighborsClassifier
sklearn.feature_selection.VarianceThreshold