# Enron Fraud Investigation

## Summary

In 2000, Enron was one of the largest companies in the United States.

By 2002, it had collapsed into bankruptcy due to widespread corporate fraud.

In the resulting Federal investigation, there was a significant amount of typically confidential information entered into public record, including tens of thousands of emails and detailed financial data for top executives.

In this project, we play detective for fun, and put our machine learning skills to use by building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. The goal of this project is to investigate the Enron Fraud case and use machine learning to find the Persons of Interest (POIs). To do this we’ll be looking at financial and email data to see if we can predict the behaviors and indicators that would make someone a potential person of interest.

## Data Files

`dataset.pkl` - The dataset for the project, there are 146 people in the data set with 18 labeled as POI’s. From looking at the data, there are 1358 features that are missing values.

### Outliers

There are some people in the data set that don’t have real names. For our investigation purposes, we’ll remove these outliers with the name “TOTAL” and “THE TRAVEL AGENCY IN THE PARK “.

### Features

There is a total of 19 Features that we can analyze for this project:

* 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`] All units are in US dollars.
* Email Features: [`to\_messages`, `from\_poi\_to\_this\_person`, `from\_messages`, `from\_this\_person\_to\_poi`, `shared\_receipt\_with\_poi`] Units are generally number of emails messages.

### Target Variable

`poi` - Label of POI's in the dataset (boolean, represented as integer)

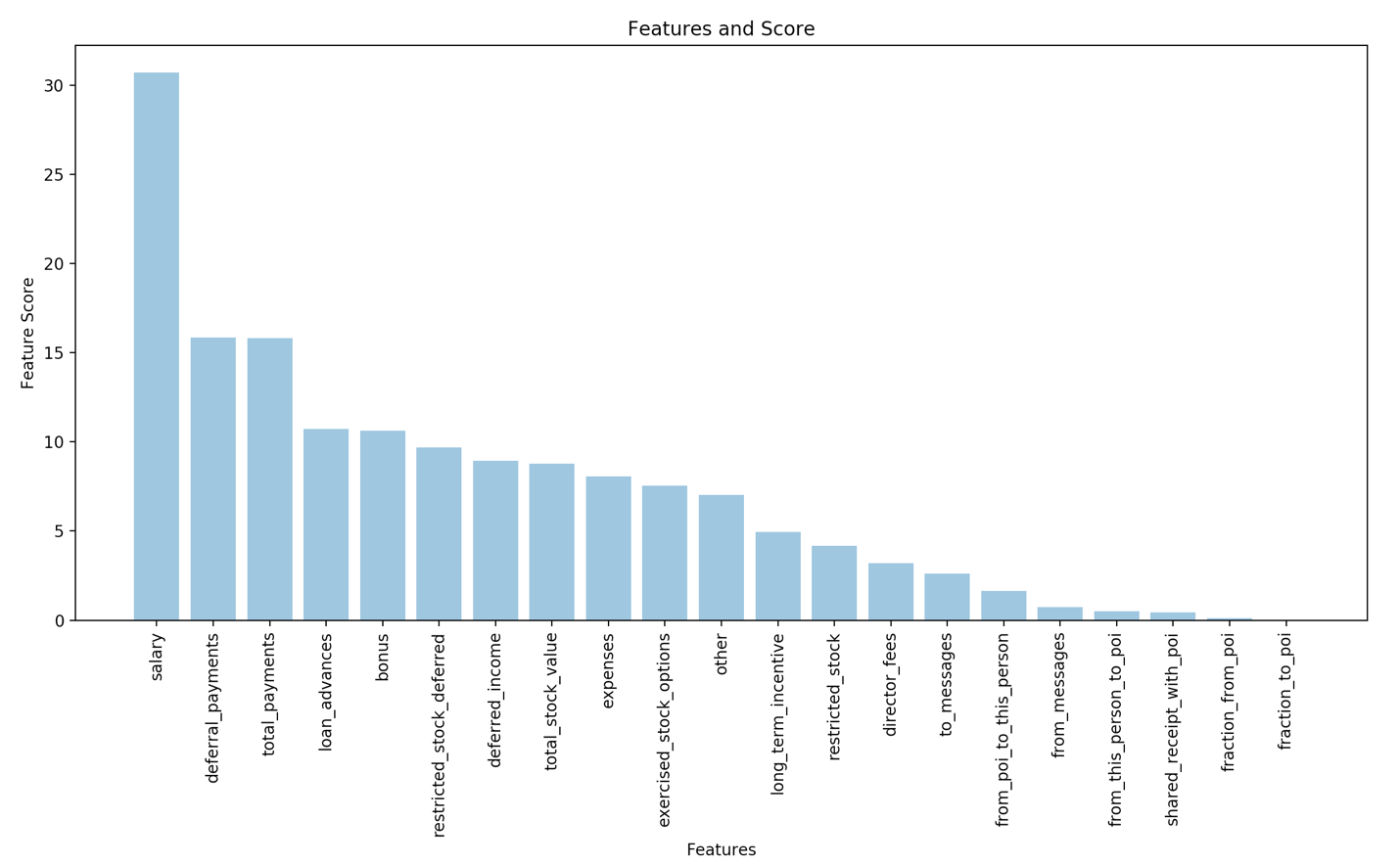
## Feature Selection

To begin the analysis, we’ll start with feature selection.

We’re going to add two new features, fraction\_to\_POI, and fraction\_from\_POI which gives the ratio of messages sent to and from POI’s. The thinking behind this metric is that a fellow POI will be sending a lot of messages to other POI’s.

Next, in order to prevent certain features for overshadowing others because of their numeric range, we added in a MinMax Feature Scaling layer to make sure each feature was scale respective to between 0 and 1. Now large numbers won’t skew our results.

To get the best features we’ll use a Select K Best feature selection to get features that are going to help the model be most accurate. I created a plot that shows each feature and their scores:



As we can see there are quite a few features with pretty high scores and there are some that contribute very little.

Using the cutoff technique, we’re going to include features that have a Feature score of 5 or higher. This is because we see a drop from around ~7.5 to 5 in score and that gives us 11 features. We’ll now use Select K Best with a k value = 11 to select the best features.

As a note, the two features we created fraction\_to\_poi and fraction\_from\_poi are going to be cut out, as their score is below 5. This is a good example of features which we thought would be significant to our model, turning out not significant.

## Algorithms

For this project, I tried multiple algorithms:

* Decision Trees
* K Means Clustering
* Naïve Bayes
* SVC
* Logistic Regression

## Parameters

An important concept in Machine Learning is tuning. Tuning is where one changes the parameters in their algorithms to get more accurate results.

To tune the parameters of the algorithms, I used Grid Search CV.

GridSearch CV is an sklearn function that automatically tunes the parameters for you, and was used on the Logistic Regression Algorithm with Principal Component Analysis to tune.

## Validation

The last step we need in our analysis is validating the accuracy of our Machine Learning algorithms to help us tune our parameters as well as see how close we are to an accurate prediction.

In validation, two important concepts are overfitting and under fitting. Overfitting is making the model fit the training data exactly, and therefore the model does a great job predicting the training data but performs poorly when we use testing data.

The opposite is true with under fitting. In this scenario, the model is too simple and doesn’t do a great job fitting either the training data nor the testing data.

To make sure we’re not overfitting or under fitting, it’s important to note that earlier in our code we used train\_test\_split to break up our data into a training set and a testing set. We trained our machine learning algorithm on the training data and now we’re going to use the testing data to see how our accurate our models are.

To validate the model that best predicts POI, we’ll look at the accuracy score, the precision, and the recall.

Here’s quick definitions of those 3-accuracy metrics:

Score - A Measure of a test's accuracy

Precision = (True Positive)/ (True Positive + False Negative). This gives us a score of how well our model does of identifying a POI when they are a POI. A true positive is classifying a POI correctly and a false negative is identifying a POI as not a POI when they’re really a POI.

Recall = (True Positive)/ (True Positive + False Positive). This gives us a score for how well our model does of identifying a POI when they may not be a POI. For example, a True Positive is identifying a POI when they’re a POI and a False Positive is calling a Non-POI a POI when they’re not.

The end goal for this program is finding a model that has greater then .3 recall and precision.

Here were the results:

Decision Tree:

* Score: 0.860465116279
* Recall: 0.4
* Precision: 0.4

Kmeans:

* Score: -25.4062141797
* Recall: 0.277777777778
* Precision: 0.625

Naive Bayes:

* Score: 0.860465116279
* Recall: 0.4
* Precision: 0.4

SVC:

* Score: 0.883720930233
* Recall: 0.0
* Precision: 0.0

Logistic Regression:

* Score: 0.883720930233
* Recall: 0.0
* Precision: 0.0

The Gaussian Naïve Bayes model did the best job in optimizing for both recall and precision with the following values:

* Score = .86
* Recall = .4
* Precision = .4