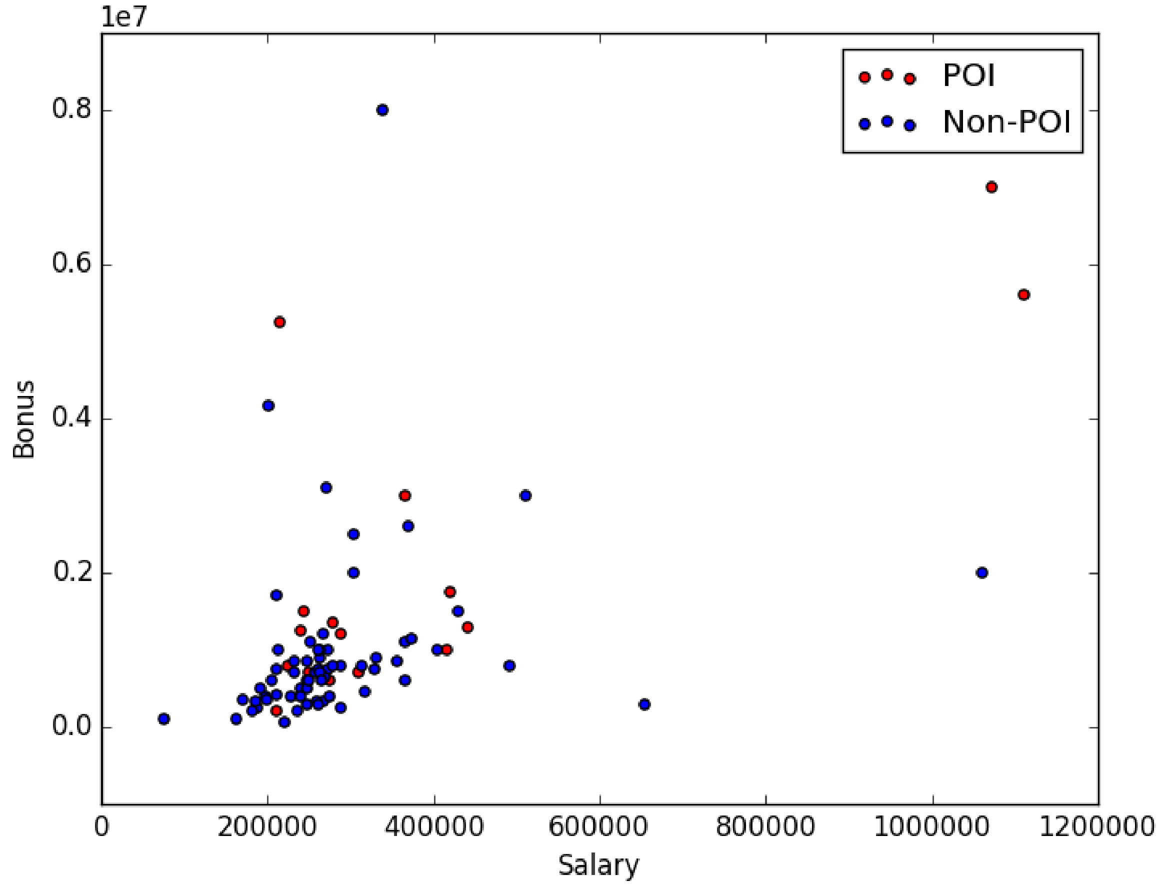
Data

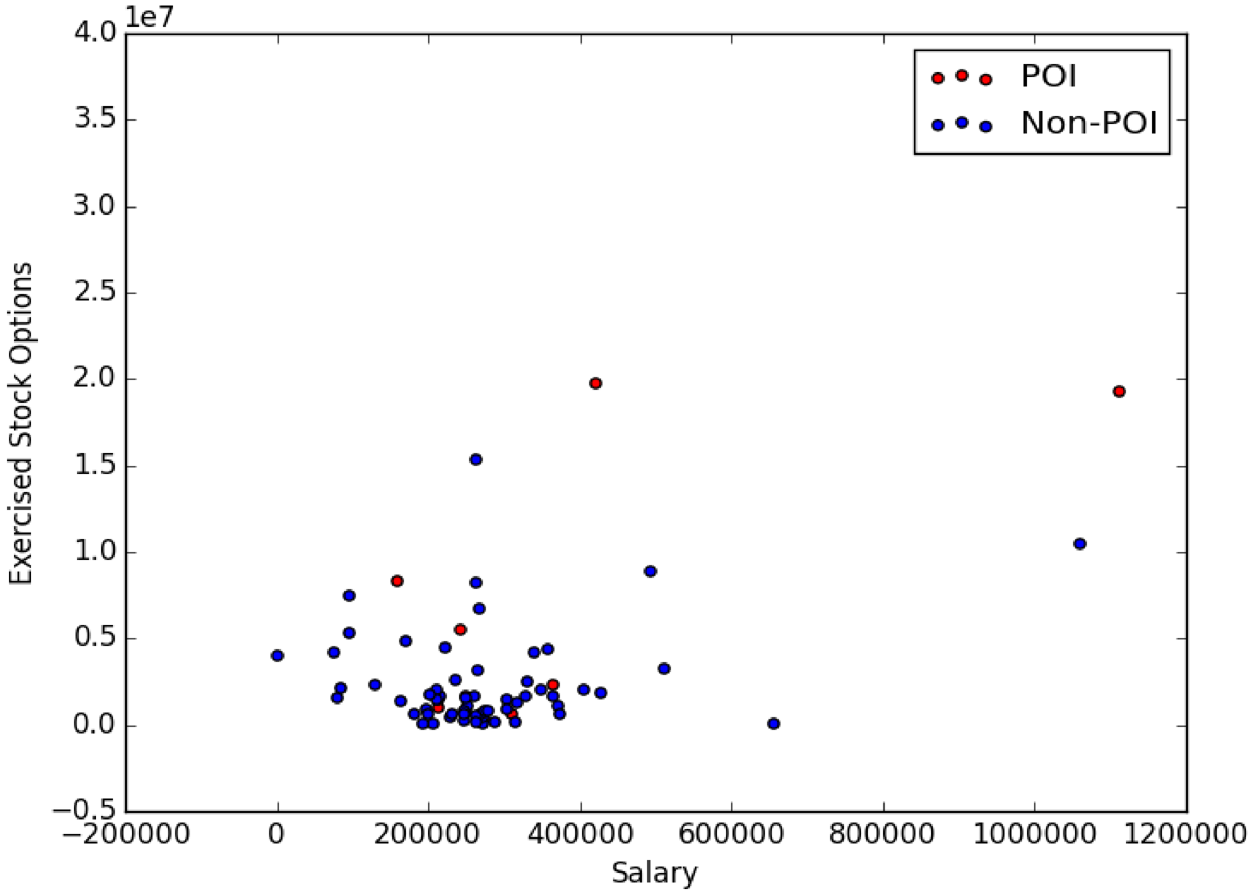
The goal of this project is to be able to identify persons of interest “POI” in the Enron case. Machine learning is helpful in these cases because normally POIs will share some characteristics that machine learning can identify or calculate to be able to predict, to a certain extent, whether a person is a POI or not.

Our dataset contains 145 data points, out of which 18 people are POIs and 127 are not.
Each data point in our data set has 21 attributes. One of these attribues is the “poi” which is the target in this case, it is set to 1 if the person is a POI and 0 if not. The rest of the attributes are the features that we're going to analyze, and feed into our machine learning algorithm to try to predict the target.

Outliers

By looking at our data and the names in it, we can directly notice that we have 2 outliers
the first is the data point with key “TOTAL” which is the row that contains the sum of each of the features we have, the other outlier that we notice is “THE TRAVEL AGENCY IN THE PARK” which doesn't seem to be a person, so it is not of interest to us. Looking further into the data by plotting some of our features we couldn't really identify any other outliers that aren't persons, or that are some data entry mistakes. What we do with outliers is that we remove them from our dataset either manually, or inside our code, before we start feeding data into our machine learning algorithm. I did that inside the code. Below are some scatter plots, the points we see that are far from the others aren't outliers, but are people that might be interesting to us, such as “Lay Kenneth” and “Jeffrey Skilling”.


1e7
0.8
0.6
0.4
0.2
0.0
Bonus
0
200000
400000
600000
800000
1000000
1200000
Salary
POI
Non-POI


1e7
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Exercised Stock Options
−200000
0
200000
400000
600000
800000
1000000
1200000
Salary
POI
Non-POI

Features

The features I ended up using are the following:

- exercised_stock_options
- total_stock_value
- stocks_money
- bonus
- salary
- other_bonus
- di_lti
- deferred_income
- long_term_incentive
- restricted_stock
- total_payments
- shared_receipt_with_poi
- exp_la
- loan_advances
- expenses

stocks_money is a calculated feature, that I calculated by multiplying total_stock_value by exercised_stock_options. I thought that the way to know how much a person made from his stock, is by multiplying what he used of his stock options by the total stock value.

sal_exp_df: I got it by summing up salary, expenses, director_fees, and total_payments. This is sort of about how much the person was making per month, which I considered important. This feature wasn't used because it was lowering my results.

other_bonus: calculated feature that I got by adding up other feature and bonus. I considered this as additional money that the person was getting, I think that the higher this is, the more he could be a POI. This feature wasn't used too, it lowered the score.

di_lti: calculated feature, deferred_income multiplied by long_term_incentive. I think these 2 are negatively correlated to our target.

exp_la: calculated feature, expenses * loan_advances. I think the more person uses as expenses and loan advances, the more this person is living higher than his income, that's a reason why a person might use fraud plans to get more money.

Feature	SelectKBest score	LogisticRegression Coeff.
salary	18.575703268041785	5.53969298104
total_payments	8.8738352555162319	-107.678503479
exercised_stock_options	25.097541528735491	-12.202491865
bonus	21.060001707536571	2.60447344204
total_stock_value	24.467654047526398	3.74881600013
expenses	6.2342011405067401	2.96159319691
loan_advances	7.2427303965360181	30.6800814586
from_messages	0.16416449823428736	-154.356825441
other	4.2461535406760671	5.33597919053
from_this_person_to_poi	2.4265081272428781	42.8987162327
director_fees	2.1076559432760908	-18.5969796279
deferred_income	11.595547659730601	-5.85719852693
long_term_incentive	10.072454529369441	6.34105619093
from_poi_to_this_person	5.3449415231473374	4.49659726799
stocks_money	24.267815589244496	28.5190280029
exp_la	7.2575177855837749	44.6287699661
di_lti	12.064714337361639	2.3313367337

Calculated feature	Precision - Recall before	Precision / Recall after
sal_exp_df	0.46769 - 0.31850	0.42868 – 0.27050
exp_la	0.46769 - 0.31850	0.46822 - 0.32050
stocks_money	0.35739 - 0.21050	0.46822 - 0.32050
di_lti	0.46354 - 0.32100	0.46822 - 0.32050
other_bonus	0.46822 - 0.32050	0.46822 - 0.32050

One feature, I was trying to calculate, or generate was by getting some words that were used in emails between POIs. I thought that if I can get a list of words that the POIs used between each others in emails, I could compare this to emails sent by any person and that would help figure out who's a POI and who's not. I faced two issues with this idea, the first is that we don't have email addresses, and texts of all POIs and the second was that even though I ran the script and tried to modify the document frequency of the words, thinking that those words would appear in a small number of emails, this strategy didn't help, I didn't get words that are really interesting.

Choosing an algorithm

I first started by testing some algorithms on the already existing features, then I started creating some features and testing the algorithms. Then I used selectKBest features to choose the most important features. Then I manually tested the number of features that scored the best, I noticed that after selectKBest increasing the number of features by 1 is sometimes increasing my scores, and sometimes decreasing it, so I had to remove some features manually and ended up using 17 features. I scaled the features using MinMaxScaler, which makes all values between 0 and 1, this scaler only affects the logisticRegression algorithm, since logisticRegression multiplies theta of each feature by the feature then sums all this up to get the predicted target, if there is a big variation in a feature, this will affect theta not in a good way.

With / Without Scaling	**Accuracy**	**Precision**	**Recall**
Without	0.26780	0.11436	0.66600
With	0.86087	0.46822	0.32050

Feature scaling doesn't affect algorithms such as ExtraTreesClassifier and decision trees, because the work by drawing limits on where to separate decisions, in our case this will draw a line of the limit between POIs and non-POIs based on the value. So if you imagine we have a plot where POIs are on the far right and the non-POIs is on the far left, the following algorithm will draw a limit in the middle of these 2 keeping the distance from each to the line about the same. So whether we scale, or we don't the plot will stay the same, just the x-axis and y-axis values will change, hence the decision will not be affected.

The algorithm I ended up using is logisticRegression. I figured that logistic regression was a good algorithm for such features, that are all kind of mathematical features, and was scoring well. As for other algorithms I tried, I tried ExtraTreesClassifier and RandomForestClassifier. These 2 were actually getting good scores, but not as high as logistricRegression, what's also interesting is that these 2 needed about the same time to run, which is much higher than the time needed by the chosen algorithm.

Algorithm Tuning

Every, or most of the, machine learning algorithm have their parameters to be tuned based on what we're trying to predict, to get the most out of the algorithm. Some make the algorithm more complex, causing it to take more time to run but gives better results, and some make it run faster but gives lower results (this is not always the case, sometimes a parameter makes an algorithm run slower but at the same time gives bad worse results), that's why we have to tune algorithms' parameters, deciding what matters more, speed, accuracy or both etc.... For my particular algorithm (Logistic Regression), I kept all the parameters to defaults, except   for C, I've set it to 1.4 manually trying many values for C I found that 11500 gives the results the closest to what I want, for example setting C to .1 increases accuracy to 0.86700, increases precision to 0.51 but gets recall down under 0.3 to 0.05400. Another parameter I modified is fit_intercept, I've set it to False, because we don't want any intercept in our case.

For the other algorithms I tried, they both take about the same parameters, I've set n_jobs to -1 to tell the algorithm to use all the cores of the CPU to run, and I have set n_estimators to 500, normally the n_estimators the better results, but the the algorithms run slower. At last I've set min_samples_split to 1 telling the algorithm to split even when there is 1 sample in a branch after this split.

Validation

After deciding on everything for our algorithm, we have to validate it to make sure we're not overlooking something and we're over-fitting that's why we're seeing acceptable results. We have to make sure our model generalizes well for any test case (will our model performs the same thing if we get a totally new dataset). So a classic mistake that might happen if we don't validate well our model, is over-fitting or under-fitting. That's why we use a validation method. The validation method that was used here is StratifiedShuffleSplit, where our data is split into training / testing, then the algorithm is trained on the training part and then tested on the test part, the difference here, is that we split the data multiple times and test multiple times, because sometimes the algorithm works well on one split but not on the other, so splitting and testing multiple times differently proves to us that these results are true.

Evaluation Metrics

LR: LinearRegression (Time to run: 1.42 seconds)
GNB: GaussianNB (Time to run: 0.71 seconds)
ET: ExtraTreesClassifier score (Time to run: 754.87 seconds)
RF: RandomForestClassifier score (Time to run: 1680.86 seconds)

Metric	LR	GNB	ET	RF	Definition
Accuracy	0.86	0.841	0.849	0.0.867	how accurate our algorithm in general, so how much correct predictions it got overall
Precision	0.468	0.372	0.334	0.505	how much is our algorithm precise, so if our algorithm says this person is a POI what is the percentage that this person is really a POI.
Recall	0.32	0.276	0.133	0.267	how much POIs we got with respect to how much POIs there were overall. So this is the percentage the 32.7% is that if there were 100 POIs we could have identified ~33 of them.

Resources

http://en.wikipedia.org/wiki/Enron_scandal
http://www.investopedia.com/articles/stocks/09/enron-collapse.asp
http://usatoday30.usatoday.com/tech/news/2002/02/19/detectives.htm
http://www.fbi.gov/news/pressrel/press-releases/former-enron-chairman-and-chief-executive-officer-kenneth-l.-lay-charged-with-conspiracy-fraud-false-statements