1) Summary of the goal of this project and how machine learning is useful in trying to accomplish it.

The goal of the project was to optimize a machine learning algorithm to identify "persons of interest" in the Enron fraud case (the largest case of corporate fraud in US history).

Machine learning is a useful tool for detecting patterns in the data that may not be immediately recognized by humans. On a small dataset like the one for this project it is conceivable that human only analysis could provide the same/better results. But when applied to a problem of larger scale, for example fraudulent credit card transactions or identifying potential terrorists the sheer volume of data makes human only analysis impractical if not impossible. Using machine learning as a tool to sift through and classify the data, combined with human over-site can make an impossible problem possible.

That said the machine learning algorithm is only as good as the assumptions/decisions made while creating and tuning it.

The original dataset contained 146 data-points of which 18 (approx 12.3%) were persons of interest (POI's) and 128 (approx 87.6%) were not persons of interest(N-POI's).

There were 20 features and one label. IT should be noted that the features were a mix of financial and email meta-data. While email content was also available, it was not available for the majority of known POI’s and as such I elected to not use the content of emails and only use the meta-data provided in the feature-set.

Many of the features had NaN (Not a Number) values

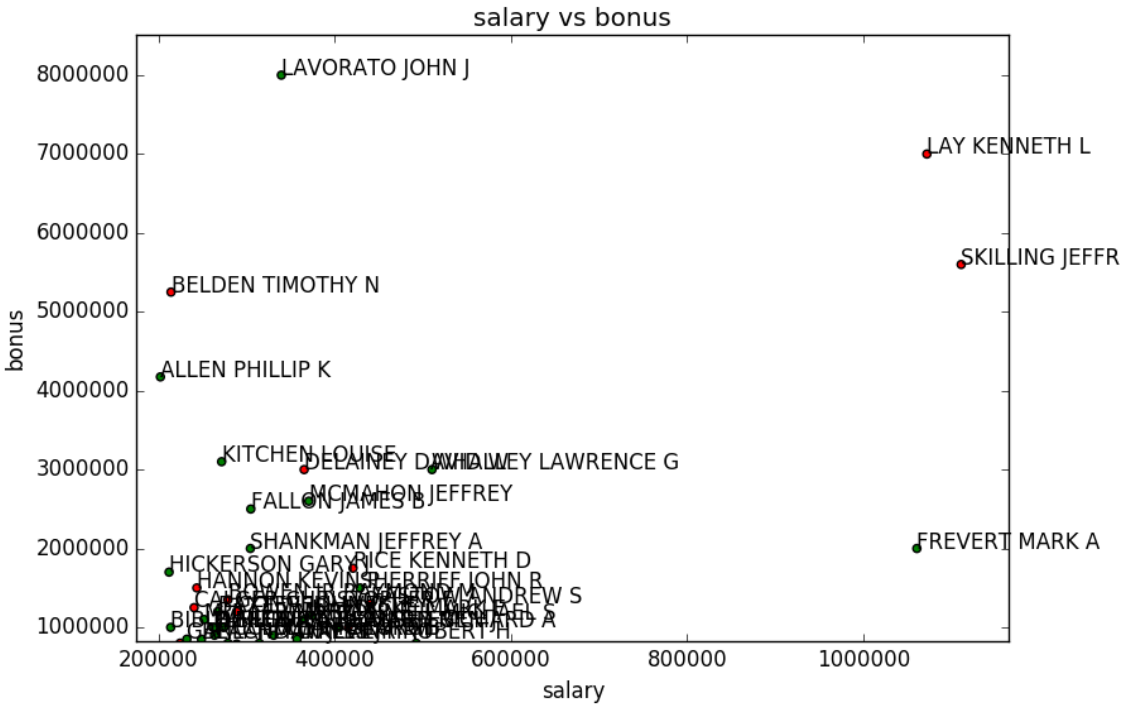
These NaN values are changed to 0.0 using the feature_scaling function found in intermediate_code.py

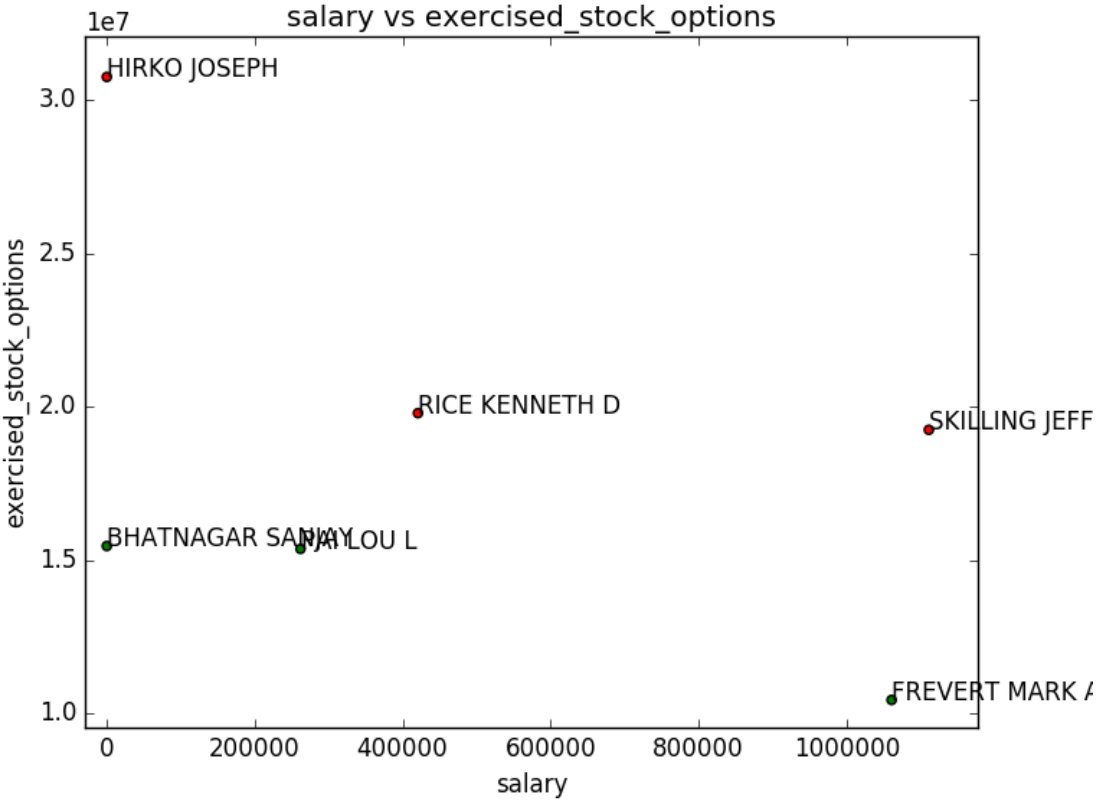
I did a feature exploration using the nanlist function found in intermediate_code.py to do an initial analysis of which features might have little value based on low occurrence rates for POI’s and non-POI’s. During this process I initially reduced the number of features from 20 to 16 simply by determining the features that had low occurrences across both POI and N-POI classes.

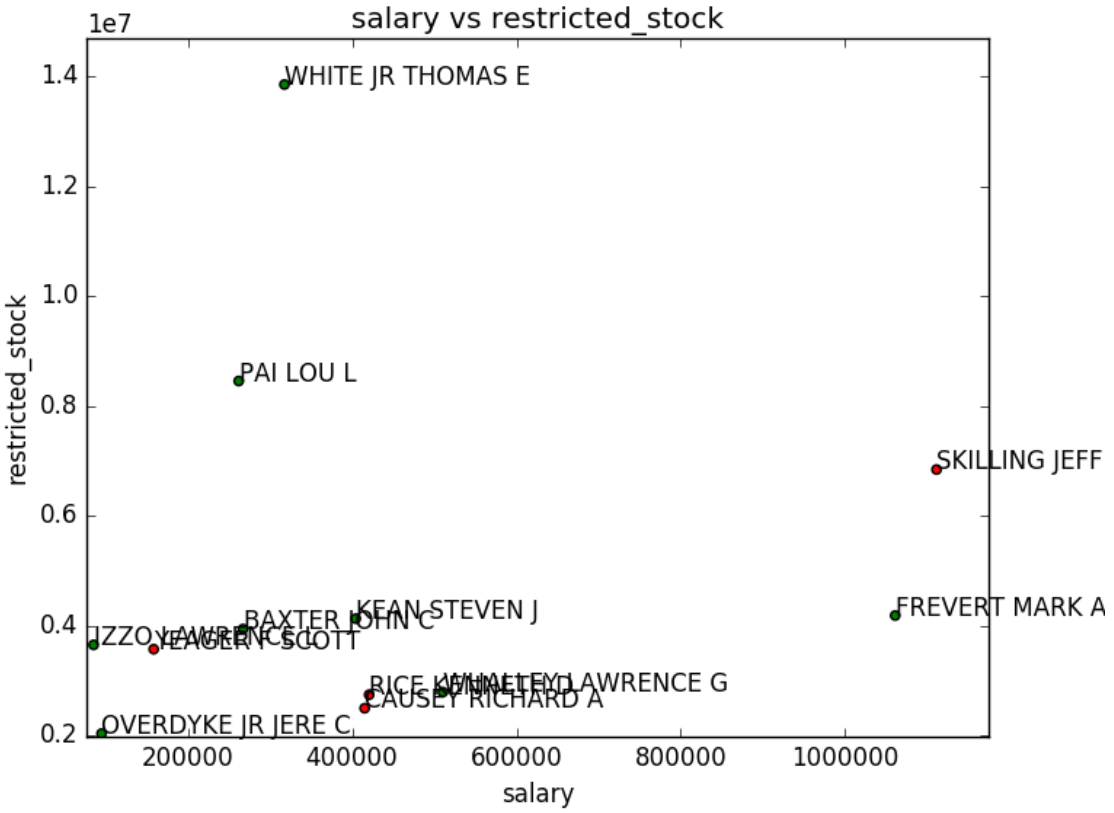
One area of importance is outliers: are they useful data points or "noise"?

In my analysis of the Enron dataset I used the outlier_examination found in intermediate_code.py to identify data points that were, in this context, noise and removed them from the dataset. These include invalid entries like "Total" or a Travel Agency and some persons known to NOT be persons of interest who nonetheless had high values for features of interest that could make them appear to be persons of interest.

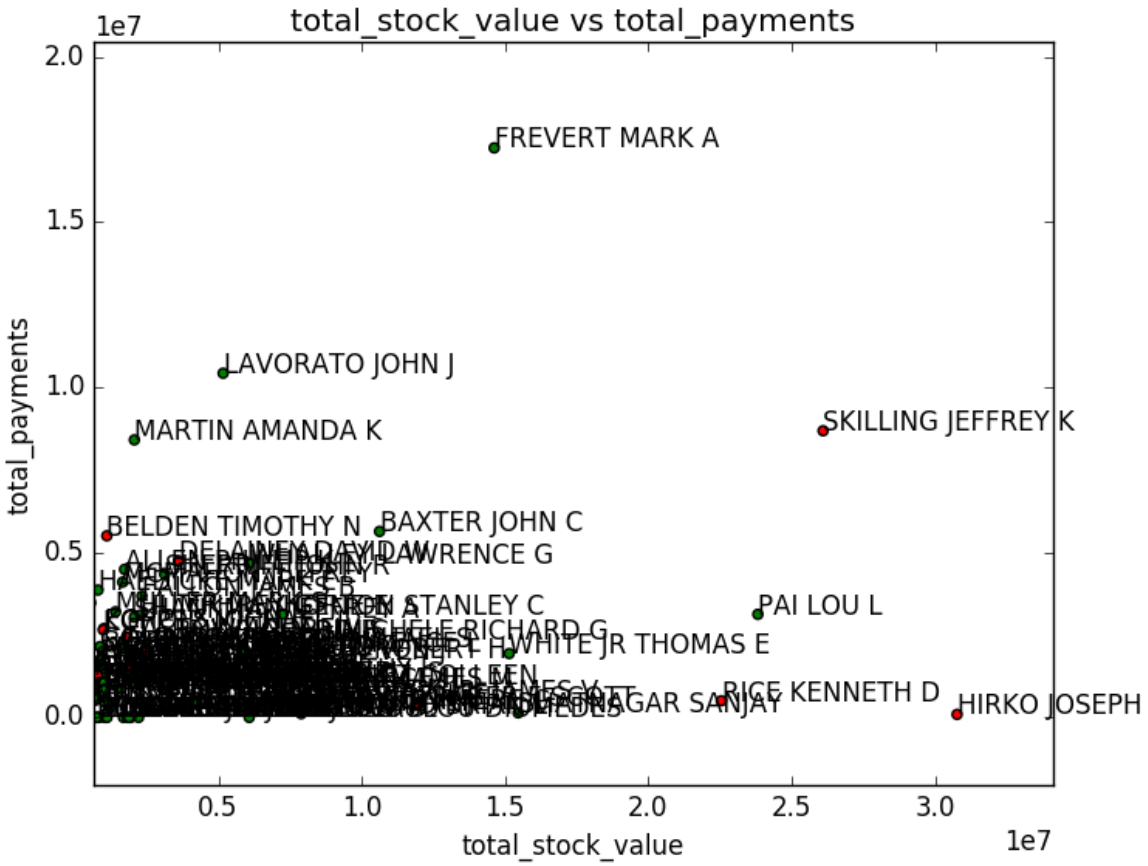
The graphs below show the results of the examination (green points are non poi’s, red are poi’s):


salary vs bonus
LAVORATO JOHN J
LAY KENNETH L
SKILLING JEFFR
BELDEN TIMOTHY N
ALLEN PHILLIP K
KITCHEN LOUISE
DELANEY DAVID W
WHALEY LAWRENCE G
MCMAHON JEFFREY
FALLON JAMES B
SHANKMAN JEFFREY A
FREVERT MARK A
HICKERSON GARY
RICE KENNETH D
HANNON KEVIN P
BHERRIEF JOHN B
MANDREW S
bonus
salary
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
200000
400000
600000
800000
1000000


salary vs exercised_stock_options
1e7
HIRKO JOSEPH
RICE KENNETH D
SKILLING JEFF
BHATNAGAR SANJAY
FREVERT MARK A
exercised_stock_options
salary
3.0
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000


salary vs restricted_stock
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
restricted_stock
200000
400000
600000
800000
1000000
salary
WHITE JR THOMAS E
PAI LOU L
SKILLING JEFF
FREVERT MARK A
KEAN STEVEN J
BAXTER JOHN C
LAWRENCE G
CAUSEY RICHARD A
OVERDYKE JR JERE C


shared_receipt_with_poi vs percent_to_poi
percent_to_poi
shared_receipt_with_poi
1.0
0.8
0.6
0.4
0.2
1000
2000
3000
4000
5000
HUMPHREY GENE E
SHERRICK JEFFREY B
HANNON KEVIN P
GARLAND C KEVIN
TILNEY ELIZABETH A
RIEKER PAULA H
BOWEN JR RAYMOND M
MCMAHON JEFFREY
CARTER REBECCA C
LAY KENNETH L
SHELBY REX
JACKSON CHARLENE R
SUNDE MARTIN
OLSON CINDY K
COLWELL WESLEY
SKILLING JEFFREY K
FREVERT MARK A
DODSON KEITH
DIETRICH JANET R
DELAINEY DAVID W
BELDEN TIMOTHY N
CALGER CHRISTOPHER F
LAVORATO JOHN J
HUGHES JAMES A
BLACHMAN JEREMY M
COX DAVID
KITCHEN LOUISE
BECK SALLY W
MCDONALD REBECCA
KEAN STEVEN J
SHAPIRO RICHARD S
SHARP VICTORIA T
WHALLEY LAWRENCE G



The outliers removed were:

- TOTAL which as the name implies was the total of all values for each feature and not a person
- THE TRAVEL AGENCY IN THE PARK which was a company half owned by Ken Lay's sister and Enron was their largest client accounting for 50-80% of their annual revenue. Enron and Enron employees purchased over $100 million USD a year in travel through the agency. It is possible that this was used to launder some of the money from Enron, but that question can not be answered easily through the data available to me.
- LOCKHART EUGENE E who had NaN values for all features
- LAVORATO JOHN J, and non POI who had unusually high salary, bonus and shared_receipt_with_poi for a non_poi
- FREVERT MARK A a non poi who had unusually high salary and total_stock_value for a non poi
- PAI LOU L a non poi who had a high total_stock_value
- HUMPHREY GENE E a non poi who had an extremely high percent_to_poi
- KITCHEN LOUISE, KEAN STEVEN J, WHALLEY LAWERENCE G and SHAPIRO RICHARD S: all non poi's who all had a high shared_receipt_with_poi

Note that in many cases the outliers removal relied on knowing which persons were or were not POI's and as such could also result in peek ahead/ data leakage and therefore not all of the outliers removed

may necessarily be considered actual outliers in a real world context as opposed to the historical context of this project (using historical data).

As part of the process I also corrected two entries in the data set that had incorrect values. They were for BELFER ROBERT and BHATNAGAR SANJAY both non-poi's. I checked their values using the totals_match function found in intermediate_code.py and checked their values manually against the enron61702insiderpay.pdf file.

2) What features were ultimately used in the POI identifier, and what selection process was used to pick them? Was scaling used and why? Were any engineered features used, why and how was it tested?

I created (engineered) 2 new features "percent_from_poi" and "percent_to_poi" using the percentage_comms_poi function found in intermediate_code.py. These features measured the ratio of communications between individuals in the dataset with known person's of interest. The rational behind creating these features was that persons of interest would be more likely to communicate with each other in part because the "conspiratorial" nature of corporate fraud requires communication between those who are complicit.

I did some examination of the new features using the poi_freq_sanity_check function found in intermediate_code.py to see what the frequency of communications looked like at various thresholds.

After this examination I was uncertain if these features would have any predictive value, but if they did percent_to_poi was the most likely.

Note that these features can lead to peek ahead so their use may not be valid in all contexts. That said the pre-existing shared_receipt_with_poi, from_this_person_to_poi and from_poi_to_this_person features all suffer from this same issue.

I used the feature_scaling function found in intermediate_code.py to scale features that had values possible below 0 or above 1. This function also converted NaN values to 0.0. This function used the x=x-xmin/xmax-xmin formula in order to give equal weight to each retained feature by scaling the values between 0 and 1.

I then used SelectKBest to determine which features had the highest determination potential on all features, pre-existing or engineered, with the exception of the email_address feature which offered no determination value.

The k values are as follows:

** = high importance. * = medium importance

salary = 1.58583956e+01 **
to_messages = 2.61613809e+00
total_payments = 8.96141166e+00 *
deferral_payments = 1.10210083e-02
exercised_stock_options = 9.68026043e+00 *
bonus = 3.07287974e+01 **
restricted_stock = 8.05496353e+00 *
shared_receipt_with_poi = 1.07226847e+01 **

restricted_stock_deferred = 1.94304716e+00
total_stock_value = 1.06327580e+01 **
expenses = 4.17386908e+00
loan_advances = 7.03793093e+00
from_messages = 4.35307622e-01
other = 3.20455696e+00
from_this_person_to_poi = 1.11182142e-01
director_fees = 1.81808008e+00
deferred_income = 1.48018515e+00
long_term_incentive = 7.55509239e+00
from_poi_to_this_person = 4.95872548e+00
percent_to_poi = 1.58380708e+01 **
percent_from_poi = 5.19324793e-01

Surprisingly two email features including one engineered email feature had high k values: shared_receipt_with_poi and percent_to_poi.
.
I did various tests using Gaussian-NB as a classifier to using the highest scoring features and combining them with the medium scoring features to determine which of the features would give me the best results.

I started with a feature set consisting of the highest scoring features (those indicated above with two asterisks) including: 'salary', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'percent_to_poi' and a label of poi and after removing non poi outliers associated with these features scored a Precision of 0.63402 and a Recall of 0.29450

I then tested further, still using Gaussian-NB as a classifier, by adding in each single asterisk feature alone and combined (again removing any associated non-poi outliers for those features) with the following results:

- with exercised_stock_options feature added Precision: 0.52012 Recall: 0.44600, a significant increase in both
- with exercised_stock_options and total_payments feature added and total_payments associated outliers removed, Precision: 0.43048 Recall: 0.35450: a decrease vs just adding exercised_stock_options
- with additional restricted_stock feature added in addition to exercised_stock_options and total_payments and associated outliers for restricted_stock removed, Precision: 0.46037 Recall: 0.39500 again not as good as just adding exercised_stock_options
- with only additional total_payments feature added and associated outliers removed, Precision: 0.44058 Recall: 0.34850 : not as good as with exercised_stock_options
- with only additional restricted_stock feature added and associated outliers removed, Precision: 0.49558 Recall: 0.39250 : again, not as good as with exercised_stock_options.

After testing the various options my final feature list which produced the best results with the Gaussian-NB classifier are the 6 features 'salary', 'bonus', 'exercised_stock_options', 'shared_receipt_with_poi', 'total_stock_value', 'percent_to_poi' and the label ‘poi’.

With all outliers releveant to the feature list removed my final dataset consisted of 18 POI's and 118 non-POI's out of a total dataset of 136 people.

3) What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]

I tested various algorithms including AdaBoosted Decision trees, Random Forest and GaussianNB with the following results:

- GaussianNB - Precision: 0.52012 Recall: 0.44600
- AdaBoost with Decision tree as base estimator - Precision: 0.43481 Recall: 0.34850
- Random Forest - Precision: 0.46176 Recall: 0.23850
- AdaBoost with Decision tree as base estimator, recall score focus - Precision: 0.41320 Recall: 0.32250
- Random forest with recall scoring focus - Precision: 0.43108 Recall: 0.25800

My final algorithm is GaussianNB. I used GridsearchCV to tune it for the best score and tested it with a variety of performance metrics (accuracy, recall, precision and f1). I feel the best results were obtained using a focus on recall as opposed to accuracy, precision or f1 scores.

My tuning tests results are as follows:

- GaussianNB parameter: prior - Precision: 0.48041 Recall: 0.44750
  - priors:[[0.1,0.9],[0.2,0.8],[0.3,0.7],[0.4,0.6],[0.5,0.5],[0.6,0.4],[0.7,0.3],[0.8,0.2],[0.9,0.1], [0.1324,0.8676],[0.8676,0.1324]]
- GaussianNB, no parameters but with scoring using recall - Precision: 0.52012 Recall: 0.44600
- GaussianNB parameter: prior, plus gridsearchCV parameter scoring using recall - Precision: 0.37742 Recall: 0.56650
  - priors:[[0.1,0.9],[0.2,0.8],[0.3,0.7],[0.4,0.6],[0.5,0.5],[0.6,0.4],[0.7,0.3],[0.8,0.2],[0.9,0.1], [0.1324,0.8676],[0.8676,0.1324]]

Of the 3 tunings the recall scoring focus with priors[0.1, 0.9] gave the best results for recall while still maintaining fair precision:

- Precision: 0.37742 Recall: 0.56650 and Accuracy: 0.78954

My reasoning for focusing on recall as a scoring metric rather than precision or accuracy is that recall minimized the false negatives which is more important in fraud situations or other situations of high risk.

Accuracy is almost useless in this type of binary classification problem with a unbalanced dataset: one where the labels are heavily skewed to label over another. A high accuracy may seem to perform well by simply guessing every data point is the most common label (in this case non-poi) while completely failing to identify a single instance of the other label (poi).

Precision minimizes the false positives but often at the expense of reduced recall which means getting more false negatives. In a negative skewed dataset like this one false positives can be manually investigated more easily than false negatives as there are far fewer of them overall.

Recall can be thought of as the ratio of a number of data-points the algorithm can correctly label vs the total number of all correct labels. If there are 100 labels and the algorithm correctly identifies all 100 then the recall is 1.0 or 100%. If the algorithm correctly identifies only 61 than the ratio is .61 or 61%

Precision can be thought of as the ratio of a number of data-points the algorithm can correctly recall vs. the total number of all data-points. In other-words the precision of the recall.

F1 scoring is a tradeoff (harmonic mean) between recall and precision, but in a balanced way, giving each equal weighting. A case could be made that this may be the best metric, but I personally feel that recall is better. Minimizing false negatives is more important in cases of fraud or other high risk scenarios where marking a positive as a negative could have dangerous financial or even life threatening repercussions (for example credit card fraud, terrorist detection, disease detection etc.).

4) What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: "tune the algorithm"]

Tuning the parameters of an algorithm allow you to adjust how it performs by adjusting various "settings". If this is not performed well you can end up with an algorithm that over or under fits the data and performs poorly at classifying new data points it has not seen before.

I chose the GaussianNB algorithm to tune and tuned it using the "priors" parameter which allows the algorithm to account for skewed data by pre-assigning values for occurrence frequencies of labels. In this case a priors setting of 0.1, 0.9 worked the best at optimizing for recall. The classify function found in intermediate_code.py makes use of the test_classifier function from tester.py to provide accuracy, recall, precision, F1 and F2 scores.

Optimizing for recall with priors set to 01. and 0.9 gave me a score of: Precision: 0.37742 Recall: 0.56650 and Accuracy: 0.78954

5) What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is a method of training an algorithm by splitting the data into sets of testing, training and validation data. Training is performed on the training data only, while validation is done using validation data. The test data is only used once the algorithm is tuned and using the validation process. If validation is done wrong you can end up over-fitting the data so it is vital that you never train on test data. It's also important that the data be split into training, validation and tests sets randomly to lessen the likelihood that you end up with all of one label in one set and all of the other in the other.

The validation strategy used was Stratified Cross Validation (as provided by the tester script) which takes a dataset, randomly splits in into a larger training set and a smaller test set. It then trains by randomly sampling the training data several times to increase the amount of training data available for training. More importantly as there is a class imbalance in the data (many more of one label than the

other) Stratified Cross Validation handles this better than a standard train_test_split can by ensuring there are similar ratios of each class in both training and test sets.

6) Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]

I used 5 different evaluation metrics as provided by the test_classifier function found in tester.py and used in the classify function found in intermediate_code.py: accuracy, precision, recall, f1 anf f2.

Of the 5 I found recall to be the most important for this specific problem as it minimizes false negatives. It does so at the expense of precision, increasing false positives.

Given the data for this problem is heavily skewed to negatives (not a poi) accuracy alone won't give a good result as simply guessing every data point was negative (not a poi) would give an accuracy of 87% while still not correctly identifying a single person of interest.

F1 score tries to strike a balance between recall and precision and a good argument could be made that it may be the best metric because of this.

But I feel that when dealing with situations where a false negative can have massive negative repercussions while false positives simply mean human oversight and manual investigation is required a higher false positive ratio is preferred over a higher false negative ratio.

As such I focused on recall as my main performance metric with precision as secondary. The goal being to get the recall as high as possible (as few false negatives as possible) while still keeping precision above 0.3

GaussianNB achieved the best result of any algorithm I tried with a priors setting of 0.1, 0.9 and a recall scoring focus resulting in: Precision: 0.37742 Recall: 0.56650 and Accuracy: 0.78954

Further steps that could be taken to improve the scores include:

- Analyzing the text content of the emails, although as most of the POI's in the data-set did not have email content and only email meta data was available this is if questionable value and computationally expensive for this specific case. In cases where more email data is available the content of emails will likely prove very useful in identifying fraud.
- PCA as a feature selection method was not tested, it would be interesting to see how it performs
- Testing more algorithms/parameter combinations could result in a tuned algorithm that scores getter.