DAND P5: Identify Fraud from Enron Email
A Machine Learning Approach

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

In this project, I will be investigating the Enron dataset provided by Udacity to find patterns related to the fraud. Using the dataset, I will be developing a Machine Learning system that identifies the Persons of Interest (POIs) based on the financial data as well as the email communications data. Machine learning helps us find patterns in the dataset and ultimately enables us to identify employees who were involved in the scandal.

This dataset contains two types of information: financial, and email. Basically, it talks about how much each employee is getting paid in different ways, how many messaged s/he has sent/received to/from POIs, etc.

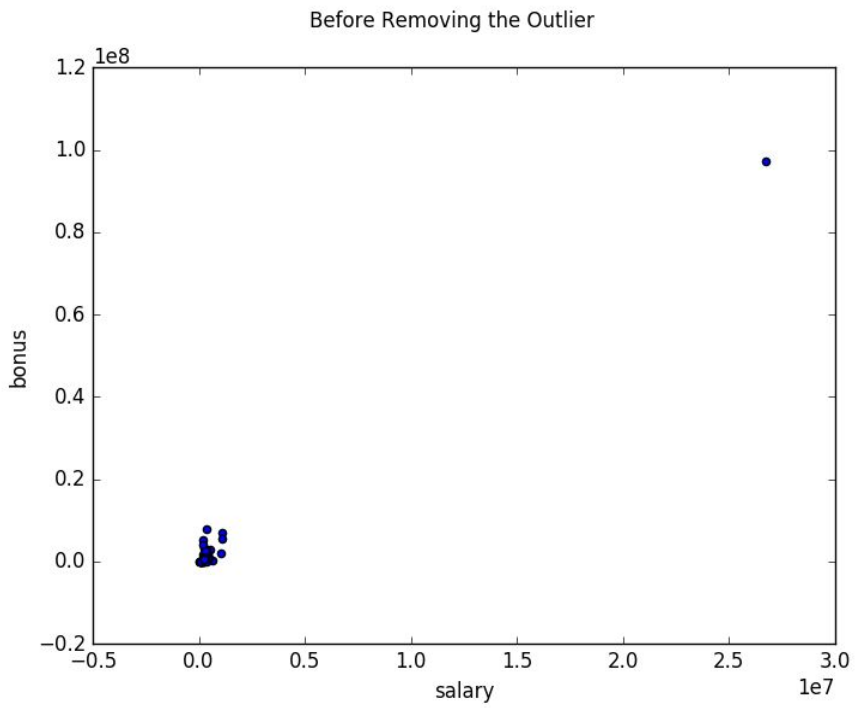
There are 146 data points, 18 of which are labeled as POI, with the rest being non-POI. Also, there are 21 features including the POI label.

It is worth noting that there are a lot of missing values in the dataset. Take director_fees for example: there are 129 data points for which this feature is marked as NaN.

I start by adding a Decision Tree classifier in order to be able to track my progress. With salary as the only feature, I get the following values:

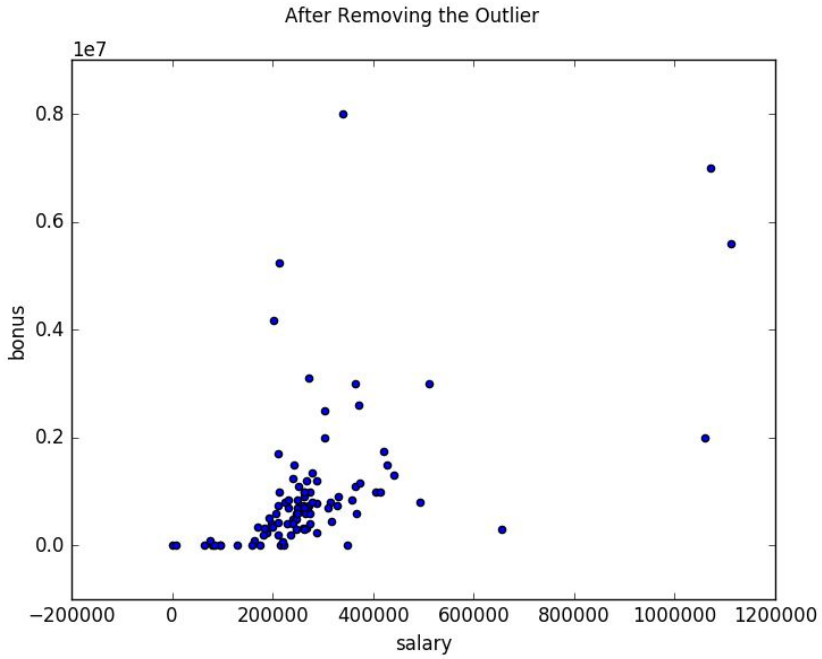
Accuracy: 0.69210 Precision: 0.23619 Recall: 0.24150 F1: 0.23881 F2: 0.24042

Plotting bonus vs. salary, I realize there's an outlier far from the rest of data points. After more investigation, it turns out the dataset includes total amounts of each parameter as a data point.



Here's the result of removing TOTAL from data_dict:

Accuracy: 0.70170 Precision: 0.25114 Recall: 0.24800 F1: 0.24956 F2: 0.24862



2. What features did you end up using in your POI identifier, and what selection process did you use to pick them?

I used SelectKBest to rank features and decide which one to use. Here's each feature with its associated score:

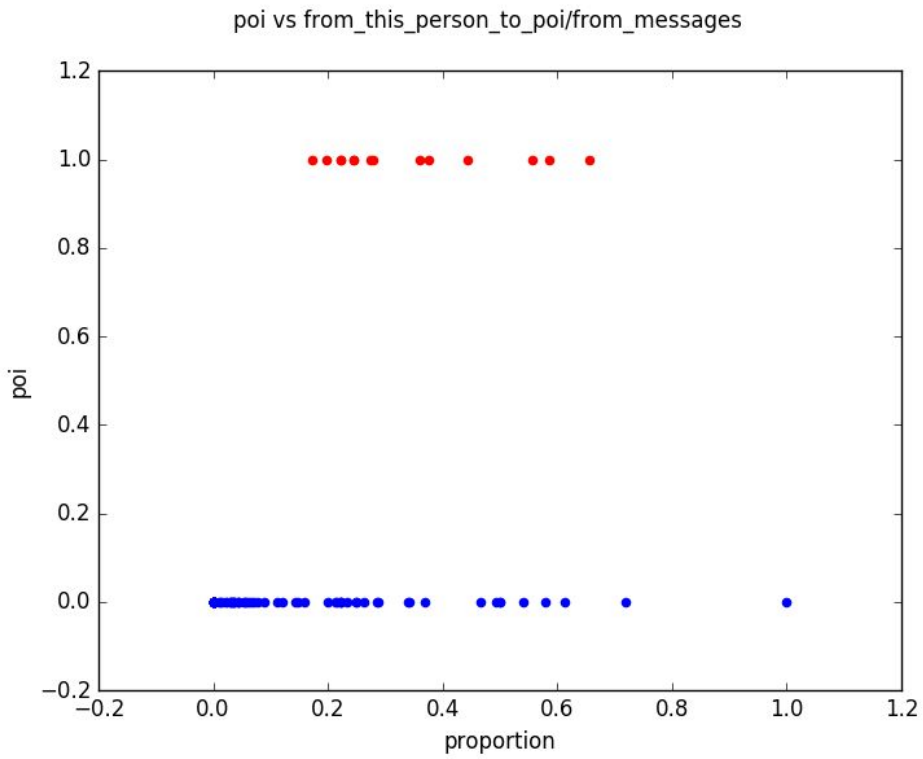
SelectKBest feature scores: [('restricted_stock_deferred', 0.06498431172371151), ('from_messages', 0.16416449823428736), ('deferral_payments', 0.2170589303395084), ('to_messages', 1.6988243485808501), ('director_fees', 2.1076559432760908), ('from_this_person_to_poi', 2.426508127242878 1), ('from_poi_to_this_person', 5.3449415231473374), ('expenses', 6.2342011405067401), ('loan_advances', 7.2427303965360181), ('shared_receipt_with_poi', 8.7464855321290802), ('total_payments', 8.8667215371077717), ('restricted_stock', 9.3467007910514877), ('long_term_incentive', 10.072454529369441), ('deferred_income', 11.595547659730601), ('salary', 18.575703268041785), ('bonus', 21.060001707536571), ('total_stock_value', 24.467654047526398), ('exercised_stock_options', 25.097541528735491)]

UPDATE: As I was getting poor Recall score, I started playing around with features. Once I removed 'exercised_stock_options', the Recall jumped about 15%. I'm still not sure as to why SelectKBest gave is such a high value, but I'm guessing there might be some outliers involved. Anyways, I decided to use Decision Tree's feature_importances_ attribute. Here's the result:

[('salary', 0.0), ('to_messages', 0.0), ('deferral_payments', 0.0), ('exercised_stock_options', 0.0), ('shared_receipt_with_poi', 0.0), ('restricted_stock_deferred', 0.0), ('total_stock_value', 0.0), ('loan_advances', 0.0), ('director_fees', 0.0), ('long_term_incentive', 0.0), ('from_this_person_to_poi', 0.011467661195428882), ('restricted_stock', 0.0620731020005066), ('from_messages', 0.075786282682834424), ('from_poi_to_this_person', 0.075786282682834424), ('deferred_income', 0.13618859772705927), ('total_payments', 0.17978190391983495), ('bonus', 0.22042651394156018), ('expenses', 0.2384896558499412)]

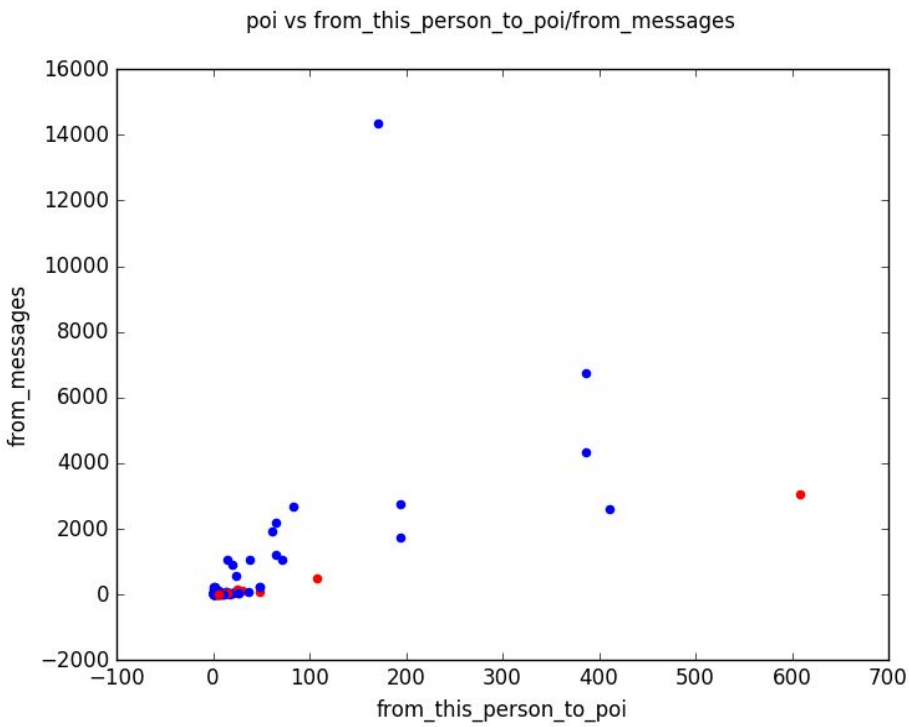
At this point, I revised the features list and based on the above results instead of those of SelectKBest. I picked features with scores better than 0.

Next, I created a new feature based on the hypothesis that emails sent to a POIs proportional to total emails received is higher for a POI compared to a non POI. This is basically normalizing the number of emails sent to a POI using total number of emails. It's worth noting that I preferred these variables to to_messages (or similar variables) based on their high feature_importances scores.



I tested my hypothesis using some visualizations. The first plot is labeled as poi vs from_this_person_to_poi/from_messages. It illustrates that the proportion for a POI is actually not necessarily higher than a non-POI but as we can see, POIs proportion mostly vary between 0.2 to 0.6 while the proportion for non POIs is more spread out. Similarly, we can see in the second plot that most of the data-points associated with POIs are concentrated in one area whereas data-points for non-POIs are more spread out. Therefore, adding this parameter (which I called *poi_to_total_proportion*) may result in less false negatives and thus, an improved Recall. Here's the result:

Accuracy: 0.81473 Precision: 0.29293 Recall: 0.27550 F1: 0.28395 F2: 0.27882



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”]

Algorithms I tried include KNN, Naive Bayes, SVM, and Decision Tree. SVM threw an error saying that there are no true positive predictions so it can’t calculate the precision and the recall. The best performance belongs to Naive Bayes. Here I compare it to Decision Tree and KNN.

Naive Bayes:
Accuracy: 0.85793 Precision: 0.45166 Recall: 0.30600 F1: 0.36483 F2: 0.32710

Decision Tree:
Accuracy: 0.81473 Precision: 0.29293 Recall: 0.27550 F1: 0.28395 F2: 0.27882

KNN:
Accuracy: 0.84573 Precision: 0.11330 Recall: 0.02300 F1: 0.03824 F2: 0.02736

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?

Parameter tuning stage in machine learning process refers to picking optimal values for the parameters of an algorithm so it meets the performance requirements. In the case of the project, I tried to pick values that result in a precision and a recall of higher than .3. I

used GridSearchCV to tune parameters for KNN and Decision Tree. After implementing the changes to parameters, here are the results:

KNN:
Accuracy: 0.87153 Precision: 0.55282 Recall: 0.19100 F1: 0.28391 F2: 0.21977

Decision Tree:
Accuracy: 0.84013 Precision: 0.41596 Recall: 0.49250 F1: 0.45101 F2: 0.47502

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]
A classic way to overfit an algorithm is by using too many features and not a lot of training data.

I used train_test_split to to make a test set of size 0.3. At each stage throughout the project, I used the test set to validate the algorithm performance. I also used tester.py since it utilizes StratifiedShuffleSplit and provides 15000 prediction events.

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.

A classic mistake is to rely upon accuracy scores. In the case of project, since the data is skewed and we're going to have a lot of true negatives, the overall score is high. But that doesn't mean our model is doing well. In order to justify the performance of our model we need to use other metrics. Precision and recall are good metrics as they change with number of false negatives and false positives.
I also used F1 And F2.

Here's some more human understandable statistics:
Total predictions: 15000 True positives: 985 False positives: 1383 False negatives: 1015 True negatives: 11617

My main takeaway from this is that the algorithm fails to detect 1015 pois out of 15000 examples.

Finally, I used PCA to explain variances associated to each feature. The result shows that the efforts made to create the new feature was really worth it:

explained variances: [('from_poi_to_this_person', 2.046757965744877e-17), ('expenses', 5.2478608134629678e-12), ('total_payments', 1.1399372828469683e-05), ('restricted_stock', 0.0016127525964120836), ('deferred_income', 0.004434089919443546),

('bonus', 0.00634001821970383), ('total_stock_value', 0.089841668507648367), ('poi_to_total_proportion', 0.8977576760479169)]