“Identifying Fraud from the Enron Dataset Report “

By Rahul Patra (25th June, 2017)

- ## Introduction:

In the year 2000, Enron was known as one of the largest energy companies in the United States, claiming nearly $111 billion in revenues. At the end of 2001, it was revealed that there was widespread corporate fraud. Essentially all of Enron's non-existent profits were created through a method called "mark-to-market" accounting, in which they would report profits even though they didn't earn a single dime. Even worse, they were solely responsible for the California electricity crisis, by which they performed large-scale blackouts to seize arbitrage opportunities.

The goal for this project is to analyse the characteristics of employees who worked at Enron during the crisis to predict if they are a person of interest ("POI"). Various resources (documentaries, news articles, etc.) were used to label people in the data set as POIs if they were previously indicted (that is, there is a 100% chance they were guilty). The application of machine learning to this project may be of assistance if we take the predictions from the Enron data model and apply them to other companies facing similar circumstances, through identifying persons of interest (we are using a naive assumption here by assuming fraud at other companies is similar to that of Enron, which may or may not be the case).

- ## Citation:

  This dataset consists from ENRON emails and financial data publicly available for research. The ENRON Email dataset was collected and prepared by the CALO Project (A Cognitive Assistant that Learns and Organizes), details are described in https://www.cs.cmu.edu/~./enron/.

- ## Data Exploration:

The features in the data fall into three major types, namely financial features, email features and POI labels.

- There are 146 samples with 20 features and a binary classification ("poi"), 2774 data points.
- Among 146 samples, there are 18 POI and 128 non-POI.
- Among 2774, there are 1358 (48.96%) data points with NaN values.

- Understanding the Enron Data:

The first step of this project is examining the data for any outliers or obvious mistakes. There was one significant outlier that arose from a spreadsheet summary line that got transcribed into the dataset; this outlier was removed by hand before proceeding.

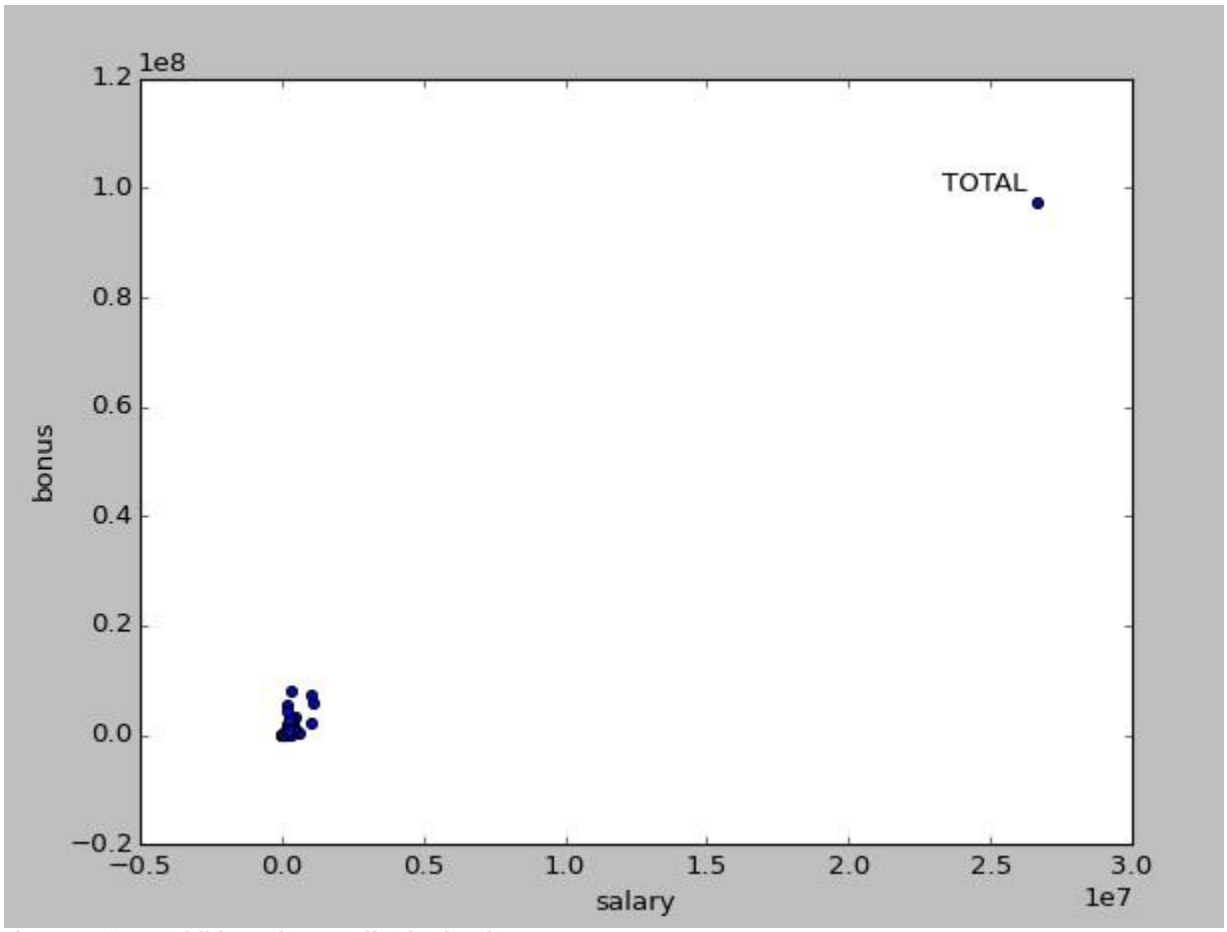

Figure 1. "TOTAL" is a clear outlier in the dataset

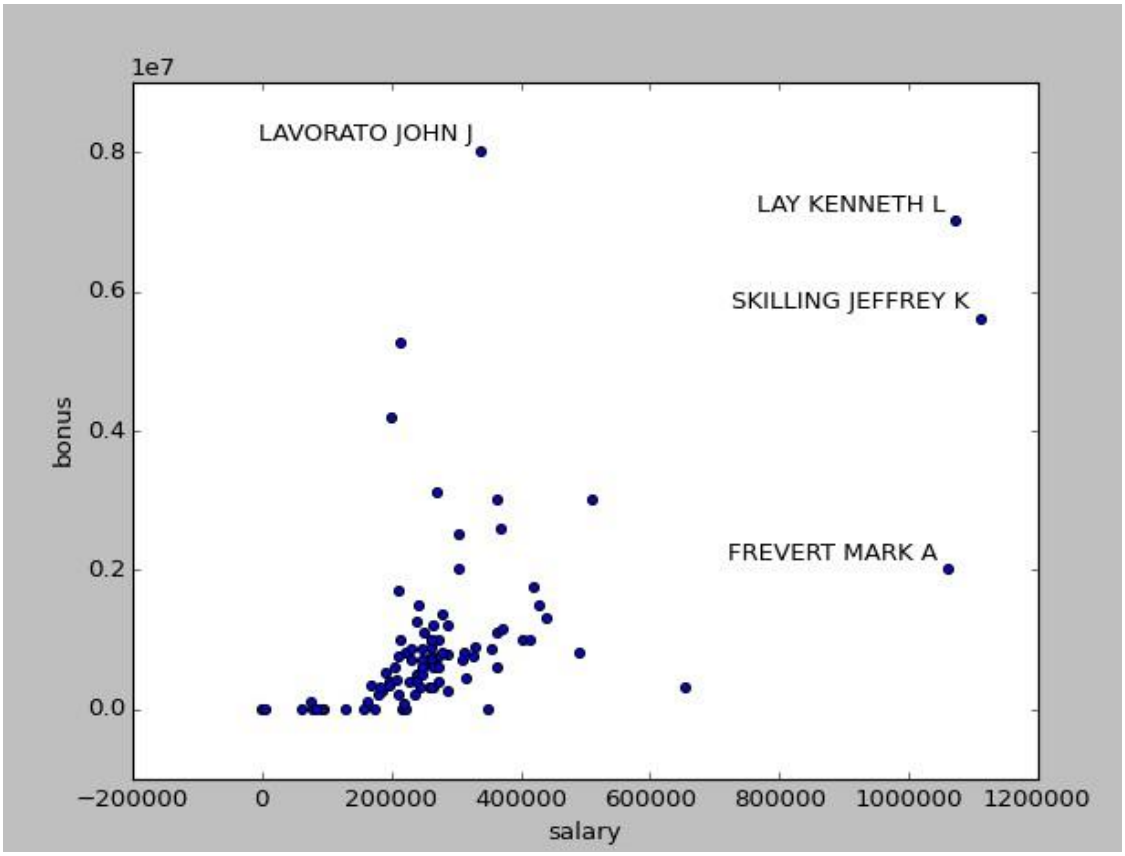

Figure 2. After "TOTAL" has being removed, this is the dataset distribution. Labels for each point are included when salary is higher than 800000 or bonus higher than 6000000.

Two more financial outliers (Ken Lay and Jeff Skilling) were left in because they are clearly real Enron characters, and the fact that they made so much money off the company is not noise or a mistake--it's tightly entangled with the corporate fraud.

Apart of this, is also necessary to remark for each person in the dataset there are unknown fields: "NaN" that latter are transformed to 0 when executing the function feature format, so that in the final dataset used to train and test the algorithms 0 and "NaN" are somehow equivalents. In order to reduce this issue, the function feature format is executed with "remove_all_zeroes=True" and when creating the final dataset, "my_dataset", those person that meets this statement are not included:

len(item) - len([x for x in item if x == 0])

<=2 being item a row of the selected

features.

Max Number of zeros per person in best feature list	Precision	Recall
1	0.28	0.30
2	0.41	0.31
3	0.38	0.34
4	0.31	0.29

Table 1. Simulations conducted with DecisionTreeClassifier(compute_importances=None, criterion='entropy',max_depth=None, max_features=None, max_leaf_nodes=None, min_density=None, min_samples_leaf=1, min_samples_split=5,random_state=None, splitter='best')

- ## Feature Processing:

Once the data was cleaned of outliers, the next step was selecting features to use. This was an iterative process, where a handful of features were selected for a first pass based on visualizations of the data that suggested that they might have some discriminatory power. I represented the different features to select those that bring some useful information to the current problem:

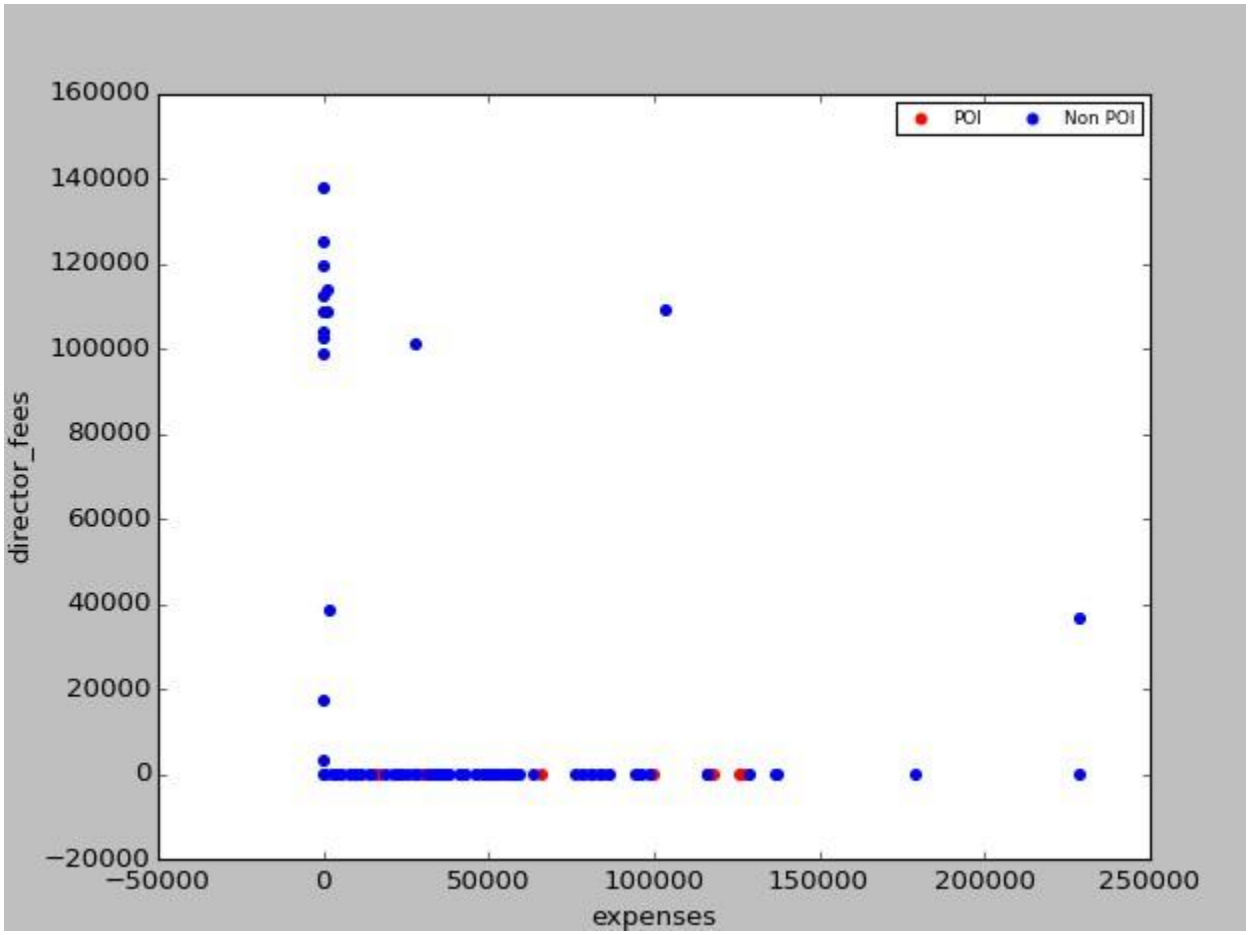

Figure 3. In the graph above, 0 is used when no information is available, because there are a lot of "nan" points that is why points are distributed in two perpendicular lines. Most of "nan" points are for the feature director_fees.

In Figure 3 it is represented director_fees vs expenses, 0 value points are when no information is available ("NaN"). As shown in the graph, there is not a single POI in the y-axis, which means director_fees is not a good feature to be used in our current problem.

Next graph includes in the y-axis the feature: total_payments:

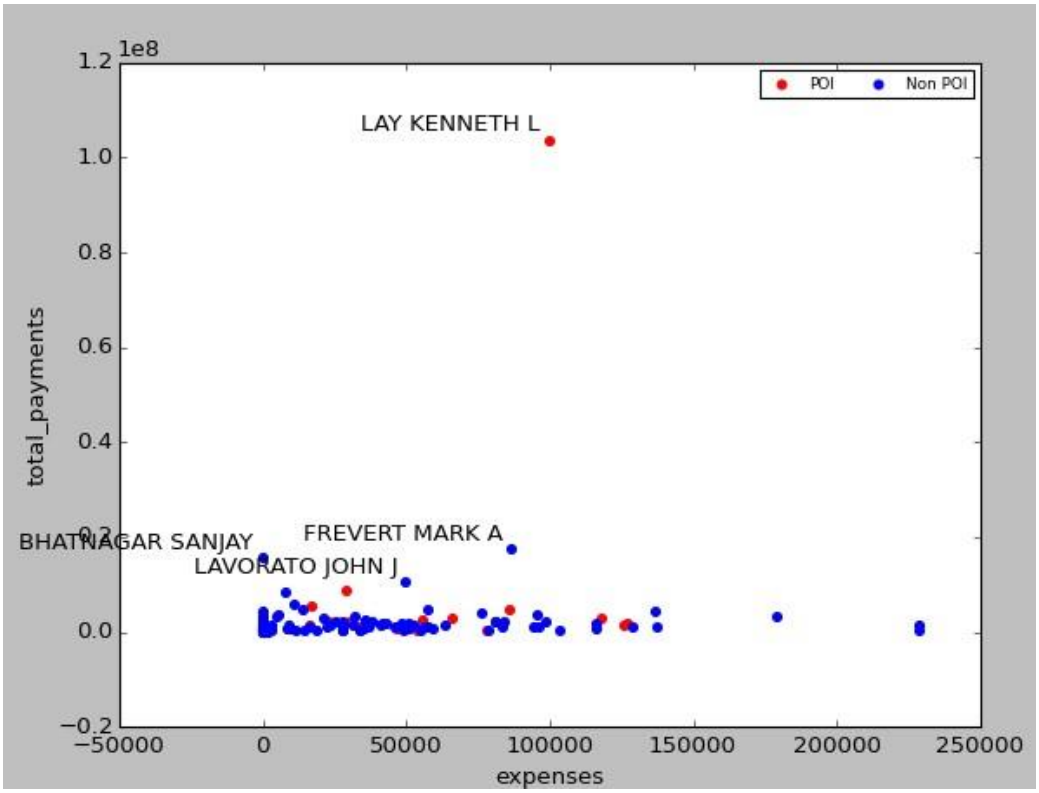

Figure 4. Lay Kenneth is not an outlier, is just a gangster.

To get a better representation, I get rid of Lay in the next graph:

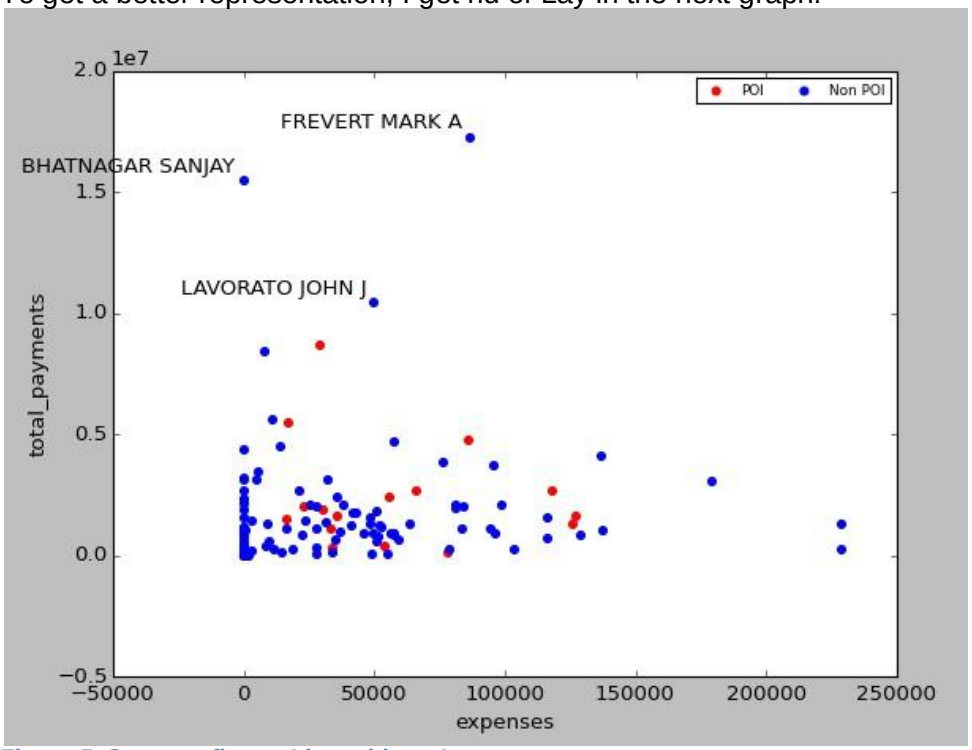

Figure 5. Same as figure 4 but without Lay.

Without Lay in the representation, we get a better insight of the data distribution. As we can see POI are somewhere in the middle of the distribution for both features (except Lay of course), maybe they wanted to remain hide?

Next picture represents the features: restricted_stocks and total_stock_values:

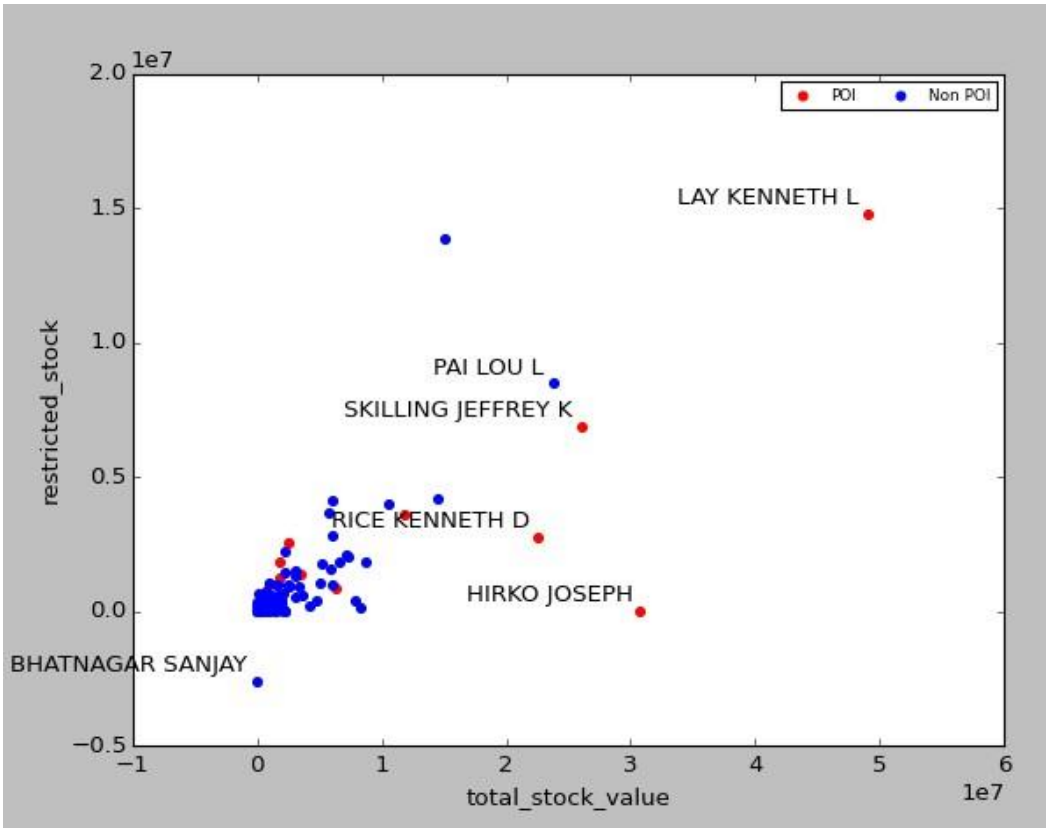

Figure 6. Representation of the features restricted_stocks and total_stock_value. There is a wrong value for Sanjay Bhatnagar.

Again no comments with Lay, but Sanjay Bhatnagar restricted_stocks value is clearly wrong (as far as I know stocks only can be a positive number), so I will correct this value in the dataset for a “NaN”.
Apart of this, there seem to be several POI points when total_stock_value is over 2M.

Following it is represented loan_advances vs deferral_payments:

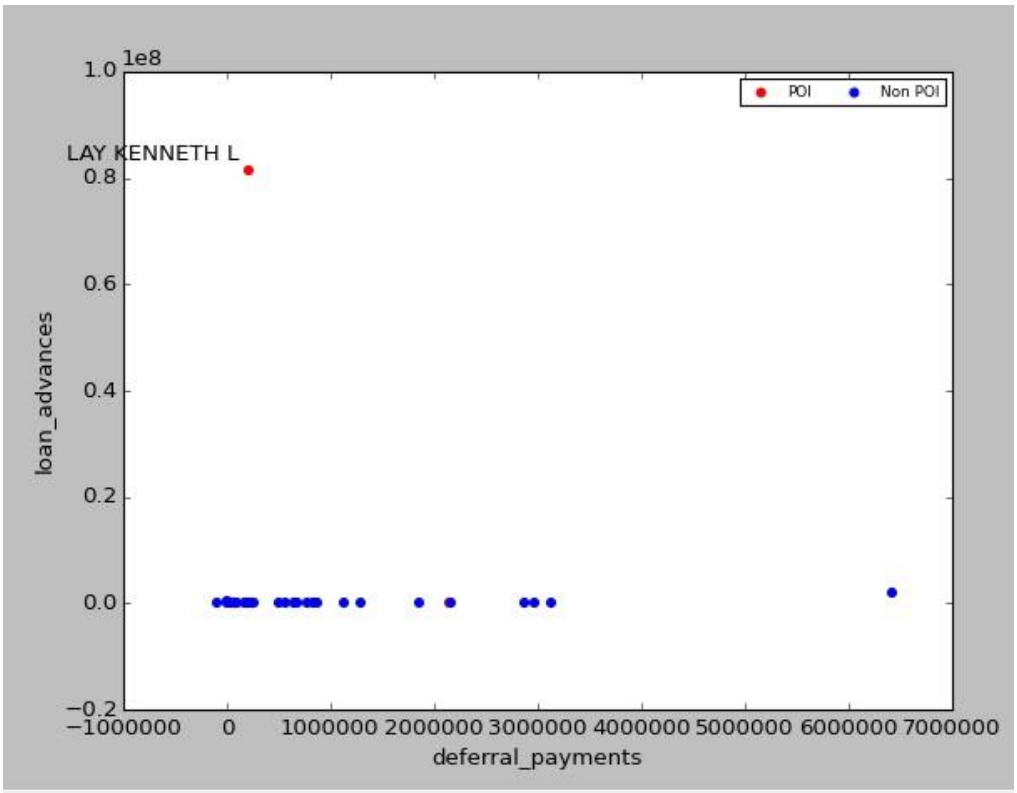

Figure 7. Representation of loan_advances vs deferral_payments.

To get a better view of the data distribution, Lay is removed in the following graph:

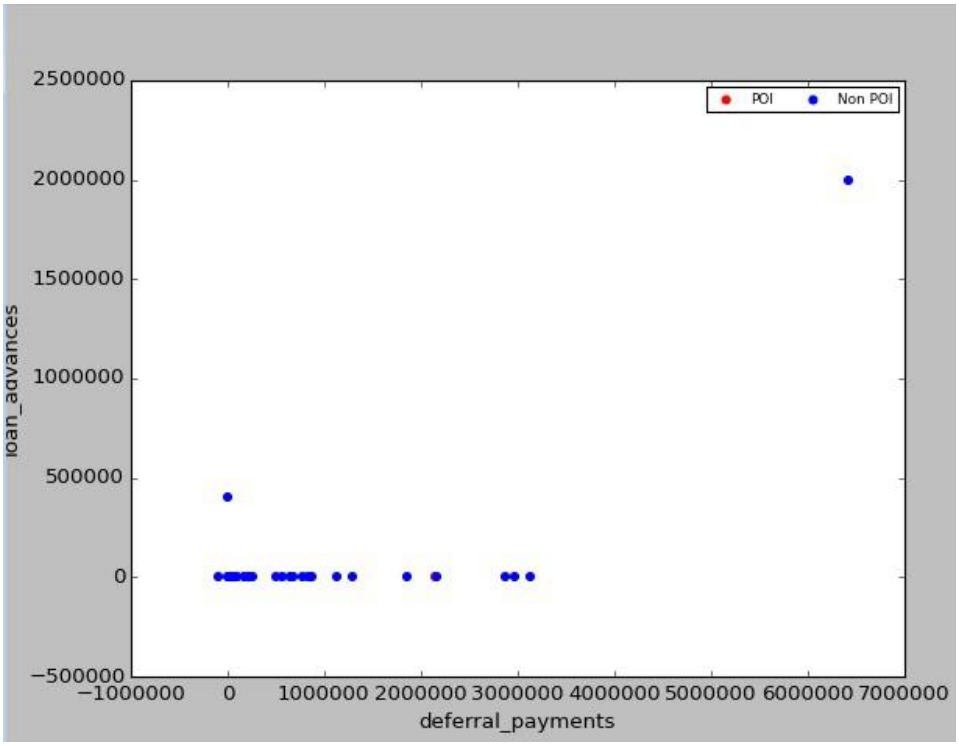

Figure 8. Representation of loan_advances vs deferral_payments without Lay

As this graph shows, these two features don't have much useful information. Next graph represents exercited_stock_options vs deferred_income:

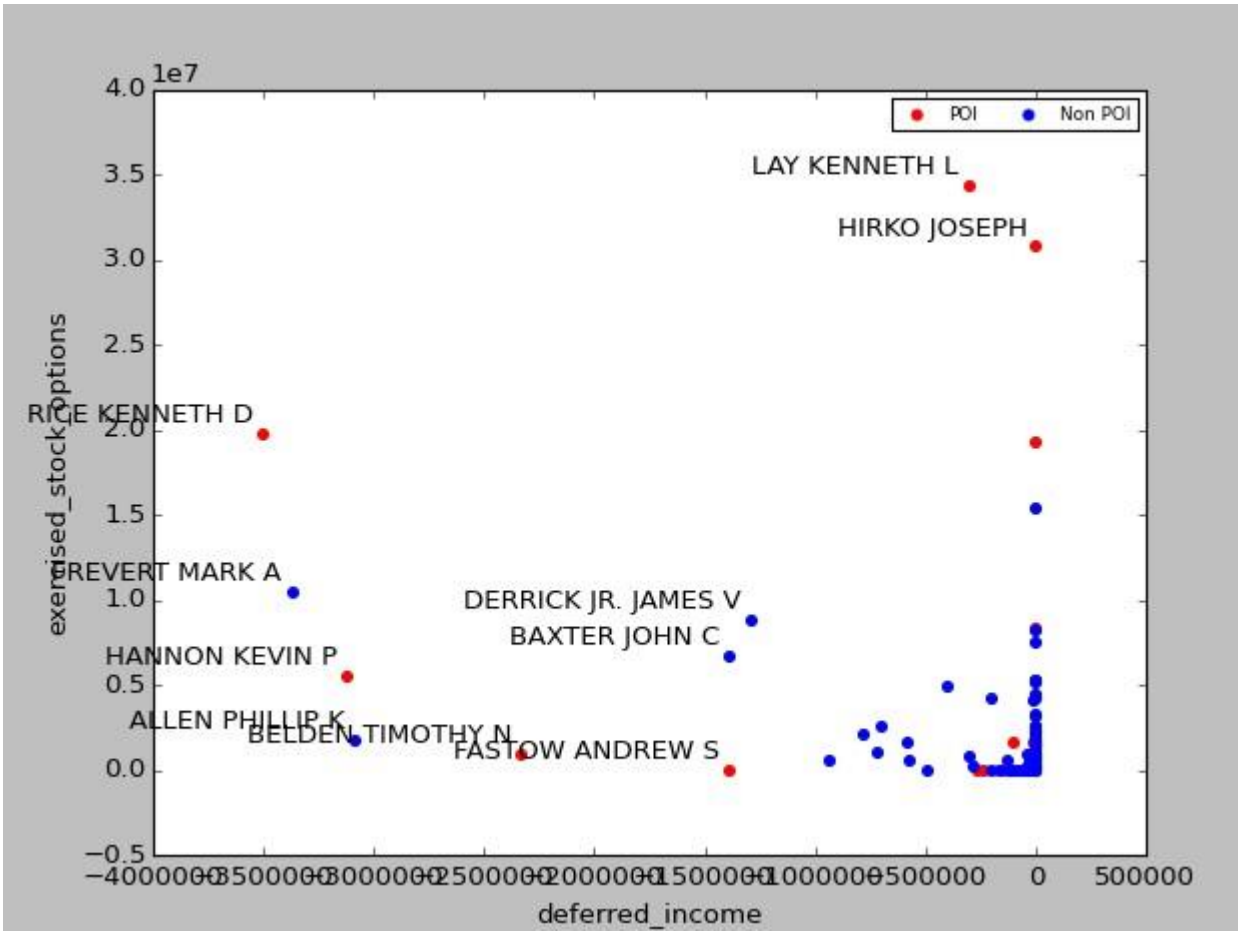

Figure 9. Representation of exercited_stock_options vs deferred_income.

These two seems to be useful features (although there are several "NaN" values in both of them), in case of exercited_stock_options, when its value is higher than 20M then the person is POI and in case of deferred_income when value is lower than – 1M the person has a high probability to be a POI.

To end with the financial data, it is represented deferral_payments vs long_term_incentive:

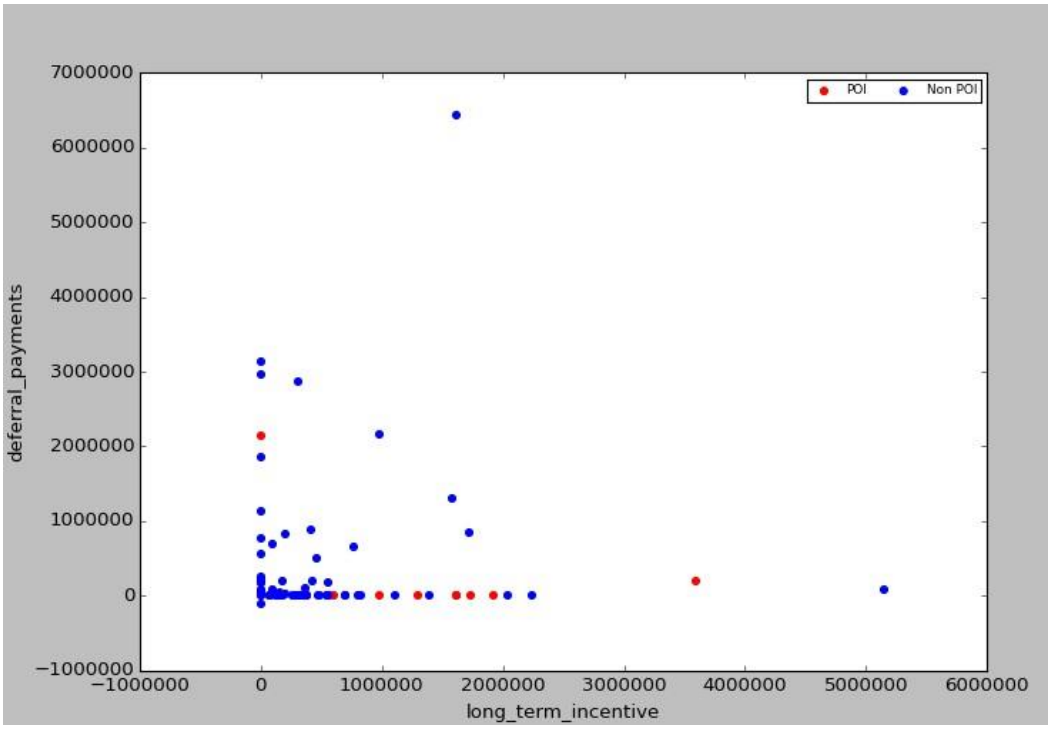

Figure 10. deferral_payments vs long_term_incentive.

Although there are a lot of "NaN" values on these features, it is interesting to see there are high chances to be POI if your long_term_incentive is higher than 0.7M.

Apart of the financial data, emails are also available. These features will be also included to try to identify the POI relations:

Next graph shows the relation between the two features: from_poi_to_this_person vs from_this_person_to_poi:

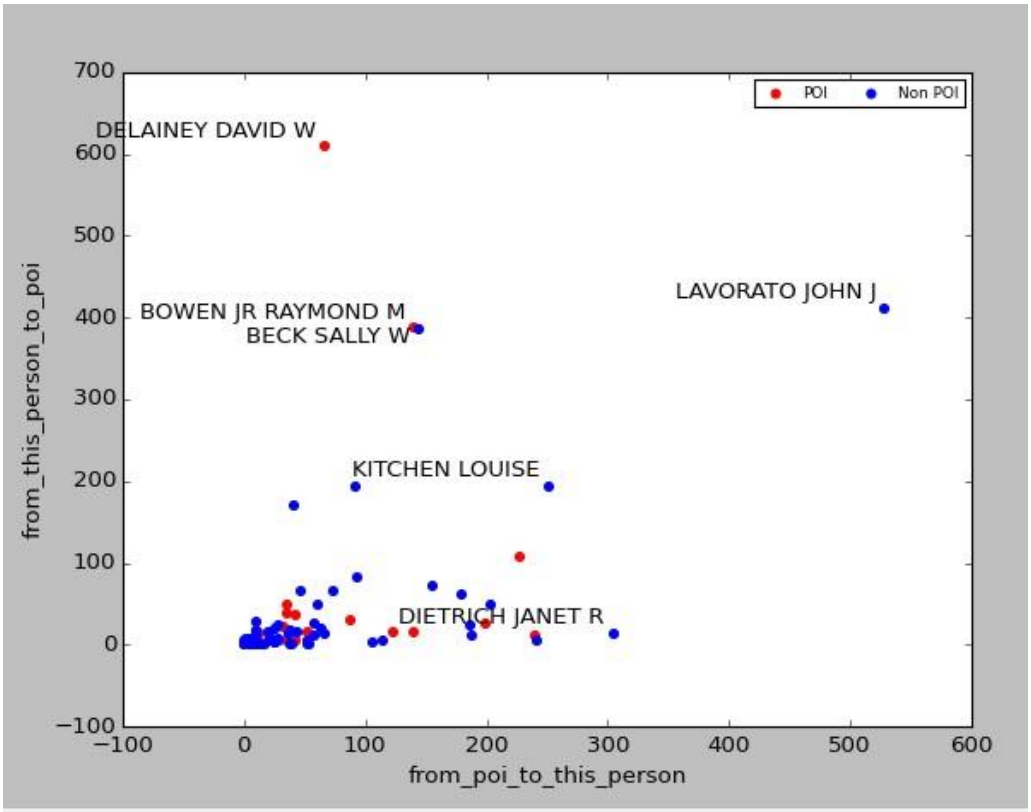

Figure 11. Representation of from_poi_to_this_person vs from_this_person_to_poi.

It seems David W informs POI on their operations? He significantly sent emails to POI. However this graph improves if instead of representing the total number of emails to/from POI we create two new features, as suggested in the example project, and represent the fractions of emails:

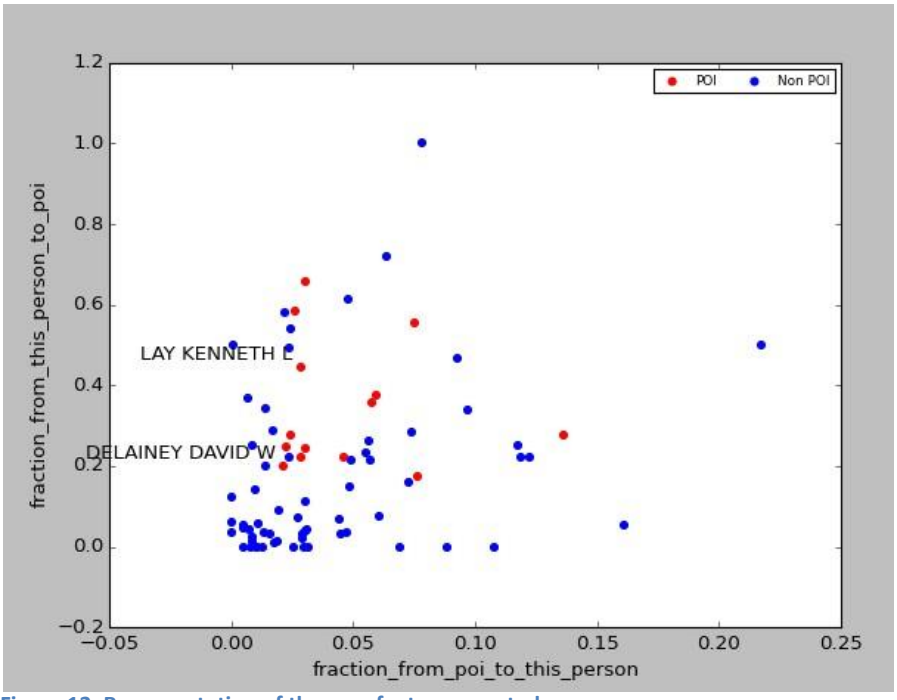

Figure 12. Representation of the new features created.

According to the previous graph, all POIs are in the interval: fraction_from_this_person_to_poi = [0.2,
0.8].

To end, this last graph shows the features shared_receipt_with_poi vs from_poi_to_this_person:

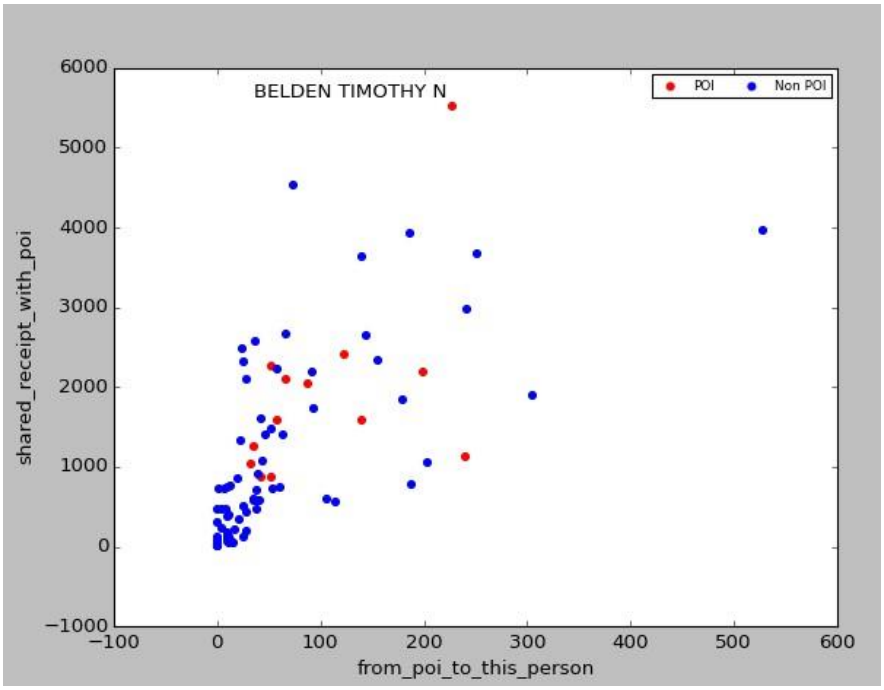

Figure 13. Representation of shared_receipt_from_poi vs from_poi_to_this_person

Except for TIMOTHY N BELDEN, POIs are in the interval between [800, 2500].

Although the previous representations help to get an insight of the data and those features that are more relevant than others, in order to find the most effective features for classification, univariate feature selection was deployed to rank the features, to do that it was performed feature scaling over the features as puts the features on a more equal footing so that they can be compared in terms of relative impact.
Feature selection is performed to have the minimum number of features necessary to capture the trend, patterns on data, this is done following the philosophy of making things as simple as possible, but not simpler (Albert Einstein), or in other words: having the minimum number of features that holds the maximum information. The machine learning algorithm performance is going to be 100% dependent on the features used.

The top 5 features were then selected, included in my_dataset (in the scaled form) and used in the final classifier; trying both smaller and larger numbers of features negatively impacted the precision and recall. These features were:

```
['salary', 'bonus', 'fraction_from_this_person_to_poi', 'exercised_stock_options',
'long_term_incentive']
```

As can be seen from the list of features used in the classifier, one of the new fractional email features were used in the final version because it did help the precision and recall.

Number of features Selected	Precision	Recall	Features
4	0.25	0.29	'bonus', 'fraction_from_this_person_to_poi', 'exercised_stock_options', 'total_stock_value'
5	0.38	0.33	'salary', 'bonus', 'fraction_from_this_person_to_poi', 'exercised_stock_options', 'total_stock_value'
6	0.31	0.28	'salary', 'bonus', 'fraction_from_this_person_to_poi', 'exercised_stock_options', 'total_stock_value', 'total_payments'

Table 2. Simulations conducted with DecisionTreeClassifier(compute_importances=None, criterion='entropy',max_depth=None, max_features=None, max_leaf_nodes=None, min_density=None, min_samples_leaf=1, min_samples_split=5,random_state=None, splitter='best')

Algorithm Selection and Tuning

This link is used as an external reference:
http://scikit-learn.org/stable/auto_examples/grid_search_digits.html#example-grid-search-digits-py

Since several algorithms (Decision tree, GaussianNB, SVM) were tested in order to find the one with best performance over "my_dataset", this was built with several remarks:

Feature Selection: The best 5 features that contain most information are included, as mentioned in the previous point, I try to keep things as simple as possible but not simpler. In general, feature selection selects the most relevant features and gets rid of redundant or irrelevant features that don't provide any useful information.

Feature Scaling: Since I need to test the performance of several algorithms, some of them needs feature scaling to be performed on the dataset like SVM. For those algorithms that don't require feature scaling like Decision Tree, this technique is not going to hurt its performance.

Finally, a decision tree classifier was selected as the algorithm for the POI identifier; Naive Bayes was also attempted (an SVM was found to be extremely slow to train, and abandoned). The decision tree was selected over Naive Bayes because an examination of the Naive Bayes predictions was worst. Ensemble algorithms like AdaBoost was also used but it didn't improve the results obtained with the Decision Trees, so that this ensemble algorithm was disregarded.

Classifier	Precision	Recall
`AdaBoostClassifier(DecisionTreeClassifier(criterion='entropy', min_samples_split=5),algorithm="SAMME",n_estimators=100)`	0.38	0.27
`GaussianNB()`	0.51	0.22
`DecisionTreeClassifier(criterion='entropy', in_samples_split=5)`	0.38	0.34

After the algorithm and features were selected, now it is time to do the final step: algorithm's tuning. Since Machine learning algorithms are parameterized, their behavior can be tuned in order to optimize its scores, in this case Precision and Recall scores. Algorithms can have many parameters and finding the best combination of parameters can be treated as a search problem. In case this step is not completed, I might not get the best performance possible for the final selected algorithm.

I played with the different algorithm's parameters: min_samples_split, max_depth and criterion to improve results. Although Scikit-Learn provide methods for automatic tuning, I preferred to use a "manual" version, finally for the selected parameters:

criterion='*entropy*',
min_samples_split=5

Max Depth	Precision	Recall
100	0.38	0.34
200	0.39	0.34
300	0.39	0.34
400	0.38	0.34
None	0.38	0.34

Table 3. Simulations conducted with DecisionTreeClassifier(compute_importances=None, criterion='entropy',max_depth=Max_Depth, max_features=None, max_leaf_nodes=None, min_density=None, min_samples_leaf=1, min_samples_split=5,random_state=None, splitter='best')

Min Samples Split	Precision	Recall
2	0.34	0.33
3	0.37	0.33
4	0.38	0.33
5	0.39	0.34
6	0.38	0.34
10	0.35	0.30
20	0.35	0.24

Table 4. Simulations conducted with DecisionTreeClassifier(compute_importances=None, criterion='entropy',max_depth=None, max_features=None, max_leaf_nodes=None, min_density=None, min_samples_leaf=1, min_samples_split=Min_Samples_Split,random_state=None, splitter='best')

Analysis Validation and Performance

In order to estimate how good, the model is, we need to firstly train it (the entire set is split in training data, used for learning, and test data (in this case 10% of the entire set) for validation) and then run validation tools to estimate the model's performance over the test set.

This validation process used cross_validation.cross_val_score with cv=StratifiedShuffleSplit(labels,n_iter = 500). Basically this StratifiedShuffleSplit creates 500 randomized folds where for each fold the entire dataset is split into training data used for learning (90% of the entire set) and test data (10% of the entire set) used for validation.

It is important to note randomized folds are important in order to avoid errors due to potential data patterns in the dataset, if folds were not created randomized, the validation process may fail estimating algorithm's performance.

This Validation process is particularly important since we need to address the Bias-Variance tradeoff (http://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff). The different performances of the algorithm for the training/test sets is going to give us an idea on the bias/variance that the algorithm presents. Although it is out of the scope of this project the analysis of the bias/variance, I'd like to note just as a reference:

BIAS: Indicates the adaptation capacity of the model to new data. A biased algorithm presents similar results for training and test sets, which means the algorithm ignores the data.

Variance: Indicates the adaptation capacity of the model to new data. Algorithms with high variance has good scores for the training sets but not for the testing sets, that is, the algorithm is not able to reproduce/forecast new situations, it is not flexible.

Since this project is about building an algorithm that meets the requirements of Precision and Recall over 0.3, I'll focus on these two metrics to quantify algorithm's performance.

For the algorithm I created, the average precision was **0.40** and the average recall was **0.37**.

Discussion and Conclusions

The precision can be interpreted as the likelihood that a person who is identified as a POI is actually a true POI; the fact that this is 0.40 means that using this identifier to flag POI's would result in 60% of the positive flags being false alarms. Recall measures how likely it is that, given that there's a POI in the test set, this identifier would flag him or her; 37% of the time it would catch that person, and 63% of the time it wouldn't.

With these numbers it would still be challenging to rely exclusively on this POI identifier alone to flag POIs. One thing that made this work challenging was that there were only 18 examples of POIs in the dataset. There were 35 people who were POIs in "real life", but for various reasons, half of those are not present in this dataset--by the "more data is better than a finely-tuned algorithm" maxim of machine learning, having a larger dataset is likely the single biggest way to improve the performance of this analysis. Another possible path to improvement is digging in to the emails data more. The email features in the starter dataset were aggregated over all the messages for a given person. By digging into the text of each individual's messages, it's possible that more detailed patterns (say, messages to/from a specific address, rather than just messages to/from any POI address, or the usage of specific vocabulary terms) might emerge. Since we live in a world in which more POI finance data might not be easy to find, the next realistic thing to try might be to extract more data from the emails.