BUILDING ENRON PERSON-OF-INTEREST IDENTIFIER USING MACHINE LEARNING

by Long Nguyen

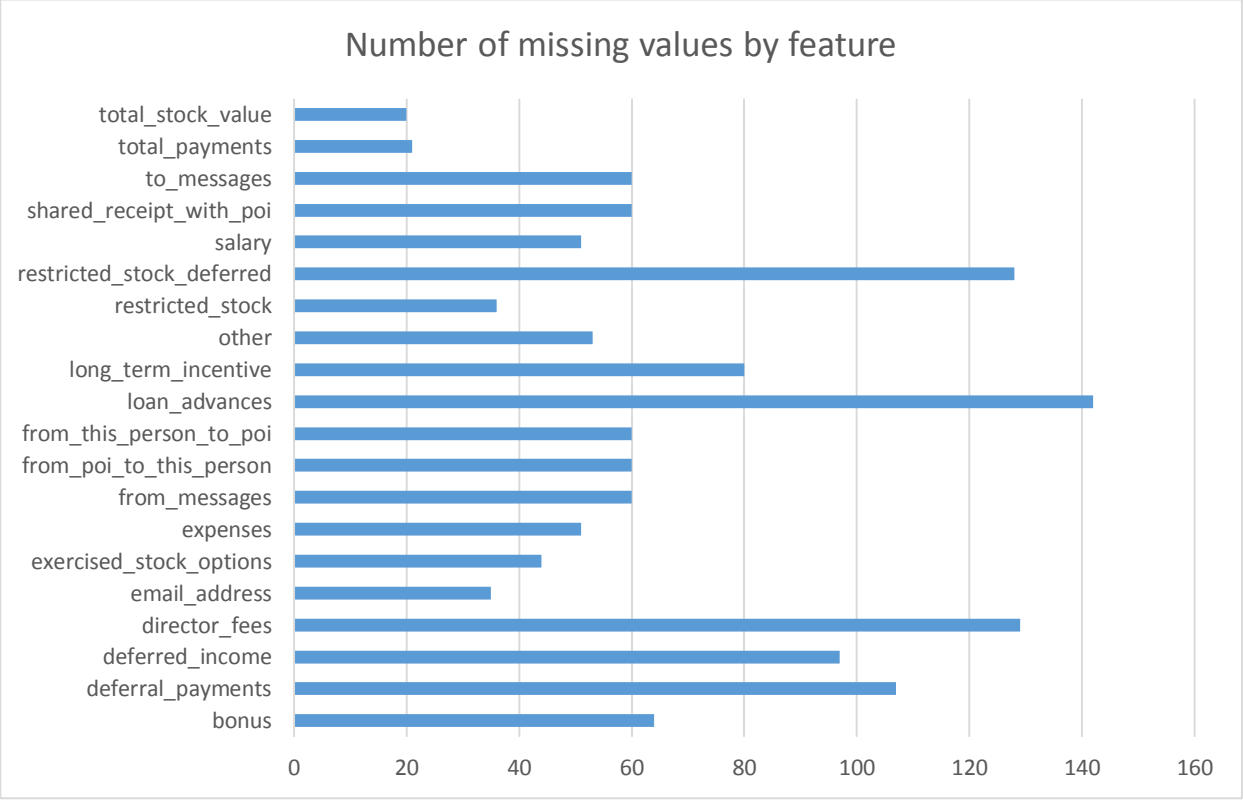
in fulfillment of Udacity Project 5

Project Mission:

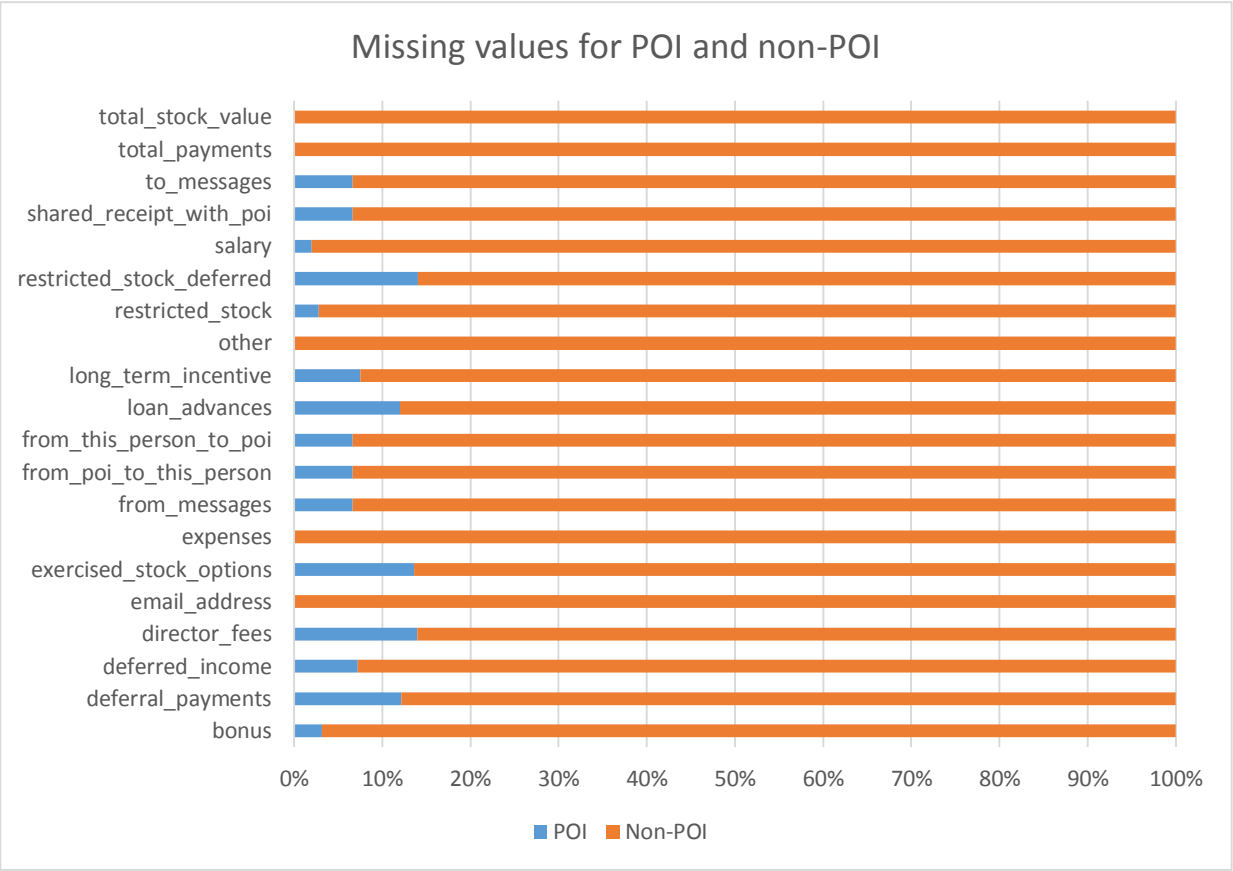
The goal of this project is to create a machine learning algorithm that can identify if an unknown person is the person of interest in the Enron scandal based on his/her publicly available emails and financial features. The emails come from the Enron email corpus, the largest email dataset available for public research as of today. It contains data from about 146 users, mostly senior management of Enron, organized into folders. The corpus contains a total of about 0.5M messages. The person of interest is defined as someone who is indicted, settled without admitting guilt, or testified In exchange for immunity. Originally, the email corpus doesn't contain a label to identify person of interest. The label is decided manually using newspaper article. There are total of 35 people of interest identified in this way. They either worked for Enron, or did business with Enron.

Data Description:

The email corpus doesn't contain all the emails of people of interest, a new data set is created which combines certain features of the email data with financial data. So now we no longer read through emails to identify word patterns, but instead extract features such as number of to and from emails, combined with financial features such as salary, bonus, etc. to form our final data set. This data has 146 data points, with each data points representing a person involved in the Enron scandal. Each person is described by 21 features, in which there are 14 financial features, 6 email features, and 1 label feature. There are total 18 people of interest in the data. Since there are only a few people of interest, this might cause unbalance in our data set and lead to un-meaningful accuracy metrics (i.e. we can achieve over 80% of accuracy just by assuming everybody is non-POI). The data also contains a lot of missing values. As we can see in the chart below, feature "restricted stock deferred", "long term incentive", and "director fees" are the ones with the most missing values.



The missing values are also not equally distributed between two classes:



From the above char, we can see that there are certain features in which all the missing values fall in to one class (i.e. only non-poi people have missing total stock value). This might cause our algorithm to determine that a future person with missing stock value data is a non-POI, while this may not be true. Therefore, such features will not be included in our model.

Project Approach:

Machine learning is appropriate for this task because the size of the data set is large and there are a lot of features that describe a particular person, which deems the task inefficient to solve manually. Also, using machine learning also makes it easier for us to make adjustments and test our prediction model.

Detecting Outliers:

There are some outliers that I was able to identify by looking at each person name in the data set and counting how many useful features available for each person. In total, there are 3 outliers that can be identified in this way:

- **TOTAL**: since this is an accounting practice, not the data representing a particular person, we can exclude this outlier from the data.
- **THE TRAVEL AGENCY IN THE PARK**: this is not an actual person name, thus it's not relevant in the data.
- **LOCKHART EUGENE E**: this person's features contain all missing values; therefore, we can exclude this feature because it provides no useful information.

These outliers are eliminated from the data set. Following data-cleaning, 143 records remained.

Feature Engineering:

The feature that I ended up using in my POI identifier came from transforming the original features with randomized PCA to get the top 11 features that explained most of the variances in the data. This approach is chosen after I have experimented several feature engineering and feature extraction methods. First, I created over 40 additional features, all of which are data driven and based on my understanding of the ENRON scandal. The list if the features is as follow:

total_missing	total number of missing features in data
total_current_compensation	equals total of salary , bonus, deferral payments , director fees, exercised stock options, restricted socks, and restricted stock deferred
total_long_term_incentive	equals total of long term incentive, restricted stock, and deferred income
poi_interaction	equals emails to and from poi divided by total to and from emails
money_left_on_table	equals total long term incentive divided by total current compensation
missing_deferral_payments	equals 1 if deferral payments is missing
bonus_vs_salary	equals bonus divided by salary
missing_deferred_income	equals 1 if deferred income is missing
missing_exercised_stock_options	equals 1 if deferred income is missing
no_missing_vs_no_original_features	number of missing data divided by number of features before engineering
from_poi_larger_than_to_poi	equals 1 if there is more from poi emails than to poi email
missing_bonus_vs_salary	equals 1 if bonus vs salary division is NaN
missing_long_term_incentive	equals 1 if long term incentive is missing
missing_poi_interaction	equals 1 if poi interaction is missing
missing_shared_receipt_with_poi	equals 1 if shared receipt with poi is missing
missing_loan_advances	equals 1 if loan advances is missing
missing_bonus	equals 1 if bonus is missing
missing_money_left_on_table	equals 1 if money left on table can't be calculated
missing_director_fees	equals 1 if director fees is missing
missing_from_messages	equals 1 if from messages is missing
from_poi_equal_to_poi	equals 1 if amount of from poi emails equals that of to poi emails
missing_to_messages	equals 1 if to messages is missing
restricted_stock_equal_restricted_stock_deferred	equals 1 if restricted stocks equals restricted stocks deferred
restricted_stock_larger_than_restricted_stock_deferred	equals 1 if restricted stocks is larger than restricted stocks deferred
from_poi_smaller_than_to_poi	equals 1 if amount of from poi emails is smaller than that of to poi emails
missing_from_this_person_to_poi	equals 1 if from this person to poi is missing

missing_from_poi_to_this_person	equals 1 if from poi to this person is missing
missing_salary	equals 1 if salary is missing
missing_restricted_stock_deferred	equals 1 if restricted stock deferred is missing
missing_restricted_stock	equals 1 if restricted stock is missing

"Total current compensation" tries to capture the amount of payments that the person received, excluding any long term deferrals.

"Total long term incentive" is the amount that this person will receive in the future if he/she stay with the company.

"POI interaction" captures the percentage of emails exchanged between this person and POI.

"Money left on the table" is the comparison between the amount of future money that this person gave up to leave the company with the amount of money he/she received. Based on the ENRON scandal, several people abandon the company knowing that it's falling, thus I believe if somebody gave up a huge sum of money on the table to leave the company, that person is suspicious.

Also, since most of ENRON POIs are known to engage in aggressive and risky business, it makes sense to use bonus as a measure of the aggressiveness (since you are paid more over the salary the better you perform). Thus, "bonus vs salary" tries to compare a person bonus to his/her salary.

In the scandal, a lot of paperwork is destroyed before the investigation. Therefore, if a person is missing a lot of financial information, it seems like that person is trying to hide or destroy the documents. Several features are created to capture this characteristic of the data, such as "total missing", "missing salary", and "n/o missing vs n/o original values". Also, some binary features are created to help separate the data points. The idea is that these features will try to create more variances among data points, thus will improve classifiers' accuracy and reduce the risk of over-fitting. These features include "restricted stock larger than restricted stock deferred", "from poi smaller than to poi", etc.

Features "other", "expenses", "total_payments", "total_stock_value", and "email_address" are excluded, because none of the POIs miss these features, while several non-POIs have these features missing. Thus, this can be picked up by classifiers as anybody who have these features missing will automatically be non-POI.

Using classification tree ensemble, I take a first look at the importance of the engineered features:

total_current_compensation	0.114
bonus	0.094
long_term_incentive	0.092
shared_receipt_with_poi	0.078
exercised_stock_options	0.057
restricted_stock	0.054
missing_deferred_income	0.052
from_this_person_to_poi	0.052
deferred_income	0.049
missing_exercised_stock_options	0.048
from_messages	0.045
bonus_vs_salary	0.044
salary	0.040
from_poi_to_this_person	0.037
total_missing	0.028

total_long_term_incentive	0.025
money_left_on_table	0.017
no_missing_vs_no_original_features	0.016
deferral_payments	0.010
total_missing_vs_all_features	0.008
originally_missing	0.008
poi_interaction	0.008
missing_long_term_incentive	0.006
to_messages	0.005
from_poi_smaller_than_to_poi	0.004
from_poi_larger_than_to_poi	0.004
missing_deferral_payments	0.003
director_fees	0.001
restricted_stock_deferred	-
loan_advances	-
missing_bonus	-
missing_money_left_on_table	-
missing_director_fees	-
missing_from_messages	-
from_poi_equal_to_poi	-
missing_to_messages	-
restricted_stock_equal_restricted_stock_deferred	-
missing_loan_advances	-
missing_bonus_vs_salary	-
restricted_stock_larger_than_restricted_stock_deferred	-
missing_poi_interaction	-
missing_from_this_person_to_poi	-
missing_from_poi_to_this_person	-
missing_salary	-
missing_restricted_stock_deferred	-
missing_restricted_stock	-
missing_shared_receipt_with_poi	-

It looks like some of the engineered features are doing quite well. “Total current compensation” got the highest score, 0.114, while surprisingly “poi interaction” is not doing quite well. Also, most of the best performing features are financial features and missing-indicating features.

Numeric features are then scaled using the Min Max Scaler to reduce the bias of classifier towards large scale features. This is very important because when projected on to large scale dimensions, data points seem to be further away from each other, thus classifiers will tend to be biased towards these features while better features with smaller scale tend to be ignored. Feature scaling would ensure that every features will be weighted evenly.

Then I used Select K Best to automatically identify 6 bet performing features. The number 6 is chosen because it gives the best performance after several manual tuning. The module selected these 6 features:

total_long_term_incentive	17.53
bonus	20.79
deferred_income	11.46

exercised_stock_options	24.82
salary	18.29
total_current_compensation	25.51

Interestingly, 2 of these features are engineered. These are also the features that Random Forest deemed as important (as in the table above). I tried to run Gaussian Naïve Bayes classifier to see how well these features perform (using Stratified Shuffle Split with 1000 folds):

> Accuracy: 0.85887 Precision: 0.46144 Recall: 0.35000 F1: 0.39807 F2: 0.36776
>
> Total predictions: 15000 True positives: 700 False positives: 817 False negatives: 1300 True negatives: 12183

Compare to 6 features without any feature engineering:

> Accuracy: 0.85727 Precision: 0.45860 Recall: 0.39050 F1: 0.42182 F2: 0.40245
>
> Total predictions: 15000 True positives: 781 False positives: 922 False negatives: 1219 True negatives: 12078

As we can see, using engineered features slightly increases the accuracy and precision, but it also reduces recall, which means while there is a high percentage of what it thinks as POIs are actually POIs, it's not as good as identifying all of the POIs out there.

However, we can still squeeze more information from the features. Instead of using Select K Best to select the 6 best features, I used PCA to try to capture the main variances in the data. My idea is first to use manual feature engineering to create original features that increases the variances between data points (i.e. features that help separating the data better), then use PCA to capture most of these variances. Since PCA depends on original features to get the shape of the data in order to determine the components that explain maximum amount of variance, it makes sense to first engineer some features that can better separate the data.

Then I go one step further and use Select Percentile to choose best features among these new features created by PCA. The number of components for PCA and the percentile for Select Percentile is optimized using exhaustive grid search.

Then I tried a number of classifiers to see which one gives the best performance: Gaussian Naïve Bayes, Support Vector Machine, and K Neighbors Classifier. The result from each classifier is as follow:

Support Vector Machine:

> Accuracy: 0.52667 Precision: 0.18125 Recall: 0.72500 F1: 0.29000 F2: 0.45312
>
> Total predictions: 300 True positives: 29 False positives: 131 False negatives: 11 True negatives: 129

Gaussian Naive Bayes:

Accuracy: 0.87333 Precision: 0.58065 Recall: 0.18000 F1: 0.27481 F2: 0.20882

Total predictions: 750 True positives: 18 False positives: 13 False negatives: 82 True negatives: 637

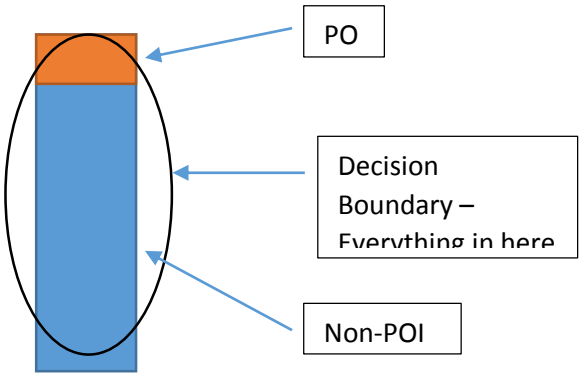
K Neighbors:

Accuracy: 0.86400 Precision: 0.48333 Recall: 0.29000 F1: 0.36250 F2: 0.31522

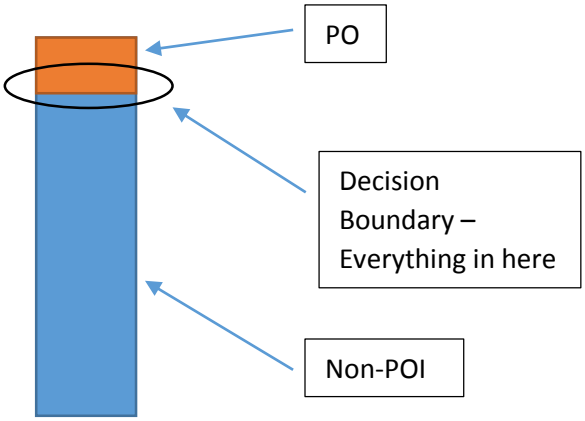
Total predictions: 750 True positives: 29 False positives: 31 False negatives: 71 True negatives: 619

Problems with each model:

Using grid search, each classifier is tuned according to the training data they receive from Stratified Shuffle Split, then the tuned model will be used to predict the test data and the average performance of these models is reflected through precision, recall, f1 and f2 score. With SVM, the decision boundary is larger, which results in many data points being classified as POI, which accidentally increases recall, but very few of the data points chosen are actually true POI (low precision). Here is the illustration:



K Neighbors and Gaussian NB, on the other hand, relies strictly on the small amount of POI in training data to determine if the new person is a POI, thus it only classifies a few number of test data as actual POI. This increases precision but also decreases recall, since it can't identify all the POIs.



In order to find the classifier whose decision boundary is of the appropriate type, I look for the classifier which doesn't try to estimate feature parameters, but rather capturing the data image and draw a simple decision boundary around each data points. I decided to use Ada Boost Classifier. With appropriate number of estimators, this classifier ensures that there is a decision boundary drawn around every data points. This is done by applying ever increasing weight for data points that are hard to predict in the training data. Moreover, Ada Boost uses Classification Tree, which learns the data by using hyper-plane cut in high-dimensions, thus it results in box-like decision boundary, which fits my goal of a small and simple decision boundary to maximize precision. Ada Boost also doesn't have a lot of tuning parameters, which greatly reduces the computational load of grid-search. The metrics is much better using either K Neighbors or SVM:

Ada Boost:

Accuracy: 0.85333 Precision: 0.43478 Recall: 0.33333 F1: 0.37736 F2: 0.34965

Total predictions: 450 True positives: 20 False positives: 26 False negatives: 40 True negatives: 364

To find the appropriate number of estimators for Ada Boost, I again used exhaustive grid-search. This is particularly important because grid search introduces automation, optimization and efficiency. Each time Stratified Shuffle Split create new training data sets, the data structure might be something completely different. For example, there might be more easy-to-predict data points in this set than in previous set; thus, more or less estimators are needed. Using exhaustive grid-search, we can introduce a range of number of estimators that we deem appropriate for most cases, and let the machine do the model optimization automatically. In the future, when new data is added to the system and the whole data structure changes, this process can be done again automatically to find the best parameters for the new training data.

I have also tried several different feature extraction methods, one of which is PCA alone. Using PCA, I reduced the data set from 15 dimensions down to 8 dimensions, without any additional feature engineering. Using Gaussian Naïve Bayes and Stratified Shuffle Split, the result is as follow:

Accuracy: 0.84247 Precision: 0.39138 Recall: 0.32700 F1: 0.35631 F2: 0.33812

Total predictions: 15000 True positives: 654 False positives: 1017 False negatives: 1346 True negatives: 11983

The performance is not as good as the combination technique. Some other combinations of methods have also been tested, such as just performing PCA on engineered features, or using K means clustering to squeeze additional information from PCA features, etc. All of which didn't yield result as good as manual feature engineering with PCA and Select Percentile.

Validation is to split your data into two independent data sets: training and testing, so that you can train the model and test it on different data sets. This is important because it helps you determine if your model is over-fit (very high accuracy score on training data but low on testing data). One of the classic mistake that you can make while doing validation is forgetting to shuffle your data. If your data is arranged so that most of POIs are in the top half of the data, then splitting the data without shuffling might give you data sets that either have all POIs or have no POIs at all, which is not appropriate to perform any training. Thus, in order to validate my analysis, I used Stratified Shuffle Split. This validation

function first shuffles the data within each class, then performs splitting within classes to preserve the class distribution in training and testing data. However, to go one step further, I also applied Stratified K Fold, to see what effects does shuffling have on my model. The result is as follow:

Ada Boost using Stratified Shuffle Split:

Accuracy: 0.85333 Precision: 0.43478 Recall: 0.33333 F1: 0.37736 F2: 0.34965
Total predictions: 450 True positives: 20 False positives: 26 False negatives: 40 True negatives: 364

Ada Boost using Stratified K-fold:

Accuracy: 0.86014 Precision: 0.43750 Recall: 0.38889 F1: 0.41176 F2: 0.39773
Total predictions: 143 True positives: 7 False positives: 9 False negatives: 11 True negatives: 116

The model performs very well using both validation methods. I used precision and recall metrics to check my model from different perspectives. Precision measures how many percent of the group that my model think is POI are actually POIs (the accuracy of the result); recall measures how many percent of the total POIs in the data are correctly identified by my model (. the completeness of the result). Accuracy is clearly not suitable for evaluation, because in our data set, the number of POI accounts for less than 15% of the entire data. If we were to classify everybody as non-POI, we can easily get accuracy of 85%. Thus, to correctly validate the accuracy of our model, we need a different measure, which is independent of class distribution in the data: precision and recall. Moreover, the f1 and f2 are the harmonic means of the precision and recall metrics. F1 averages precision and recall, and assign equal weight to both of them, while f2 assigns recall twice the weight it gives precision. Thus, higher f2 tends to mean that the model gives higher recall metrics.

Conclusion:

The most challenging part of the project was to engineer various feature from the raw data. It's because even techniques like PCA depends a lot on original features. If we have features that can better separate the data points, then PCA can identify principal components better and these components would be useful in downstream analysis. Also, picking out an appropriate classifier is also a difficult task, due to the trade-off when using different classifiers. K Neighbors and Gaussian NB are very sample specific, which means it strictly depend on the each data points and their specific feature without using any function to extend the decision boundary. This often result in high precision but very low recall. SVM, on the other hand, can extend decision boundary with radial basis function, yet is very computationally intensive and depends a great deal on tuning, which results in high recall but low precision. In my opinion, the model would be more useful if it can help identify potential POI for further investigation; thus, it is best if the model has very high recall. I would suggest using SVM with radial basis kernel for this task, since SVM boundary effectively capture most of POIs:

Support Vector Machine:

Accuracy: 0.52667 Precision: 0.18125 Recall: 0.72500 F1: 0.29000 F2: 0.45312 Total predictions: 300 True positives: 29 False positives: 131 False negatives: 11 True negatives: 129

What can be done in the future is to explore as many original features as possible. Some suggestion can be the job title, or whether that employee left the company after or before it went bankrupt. I believe the model can greatly benefits from these new original features.

Work Cited:

Introduction to Machine Learning (Udacity)

Scikit-learn documentation

Kernel methods (Wikipedia)

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.