“Ask Why.”

---A Machine Learning Study of Enron Data

Guo Yading
2016/4/26

This document contains answers to what, how and, more imprtant, why. It is always important to “ask why”, the famous slogan of Enron Corporation. Dataset itself is numbers without further annotation. I believe that these numbers are from facts. It is often a lot easier to deduce numbers from facts than vice versa. Machine learning provides tools and framework to learning rules and building models out of numbers. However, it is user’s obligation to understand the data, choose tools for building models and justify the result. So the reasons behind each step of machine learning does matter.
Three main sections, exploration, analyzation and evaluation, are concentrated on what I have done with the Enron data and why I did so.

Exploration

This main section is divided into three subsections, introduction, background and summary of data.

Introduction

The data contains features, described by numbers, of people. The goal of this task is to build a model which can classify these people into two groups by these features. The persons in the dataset are real persons from former Enron Corporation. The two groups to which these people belong are “Person of Interest (POI)” and “non POI”. These persons’ features provided by the dataset include two kinds. One is email related such as numbers of emails the person sent and recieved. One is finicial information such as salary and bonus. So the problem is, given a person's features, how can we systematically decide whether the person is POI by only using these features. The role of machine learning in this task is to build a predictive model by using data provided and verify the model by true data.

Background

The reason why these people were divided into two groups (POI and non POI) lies behind the astonishing story of Enron. By 2001, the fifteenth year of Enron, the Enron Corporation grew into seventh-largest company in US. There are over 21000 people in 40 countries working for Enron. And all it begin with were just two local natural gas suppliers. However, a monsterous scandal of Enron is revealed in

October 2001. In less than two months, Enron filed the largest bankruptcy in American history at that time. Enron's audit company, Arthur Andersen which was one of the five largest audit and accountancy partnerships in the world, cllapsed due to the scandal.
The consequence is catastropic. Enron's stock price fell from $90(August 23, 2000) to $0.12(January 11, 2002). Shareholders lost nearly $11 billion. Nearly 62% of 15,000 employees' savings plans relied on Enron stock that was purchased at $83 in early 2001 and was practically worthless. Thousands of them lost most of their pensions. And, here I quote words from Senator Byron Dorgan who is Chairman of Congressional committee investigating Enron's downfall, “In the Titanic, the captain went down with the ship. And Enron looks to me like the captain first gave himself and his friends a bonus, then lowered himself and the top folks down the lifeboat and then hollered up and said, 'By the way, everything is going to be just fine.' ” And this is the reason why we need to find Persons of Interest (POIs).
POIs are basically people who are responsible for scandal of Enron. They used extreme complex financial reports to confuse shareholders and analysts. They also use illusional future to interpret earnings and modify the balance sheet to misguide investors and, even, their own employees. The majority of these wrongdoings were carried out by Kenneth Lay (founder of the company), Jeffrey Skilling (Chief Exective Officer), Andrew Fastow (Chief Finicial Officer) and other executives. These people are called Person of Interest in this task. Our goal in here is to find out these executives among all the executives from Enron by the methods of Machine Learning on the dataset provided.
This dataset is extract from Enron email dataset and public available finicial dataset regarding Enron case. The Enron email dataset was originally made public, and posted to the web , by the Federal Energy Regulatory Commission during its investigation.

Summary

The dataset contains features of 146 items (145 persons and 1 total calculation). 18 of them are POIs and 127 non POIs. Each person have 21 features. Some of email related features are the number of email recieved and sent, the number of emails other POIs sent to the subject and so on. Other features are about finicial information such as salary, bonus and total stock value. I also notice that a lot of value is missing (represented by ‘NaN’ in the dataset). For example, for the bonus of these people, 64 values of 145 are missing. It is also worth noting that almost all the POIs lack direct information of emails. The features of POIs’ email mostly came from other managers' mailboxes.

Analyzation

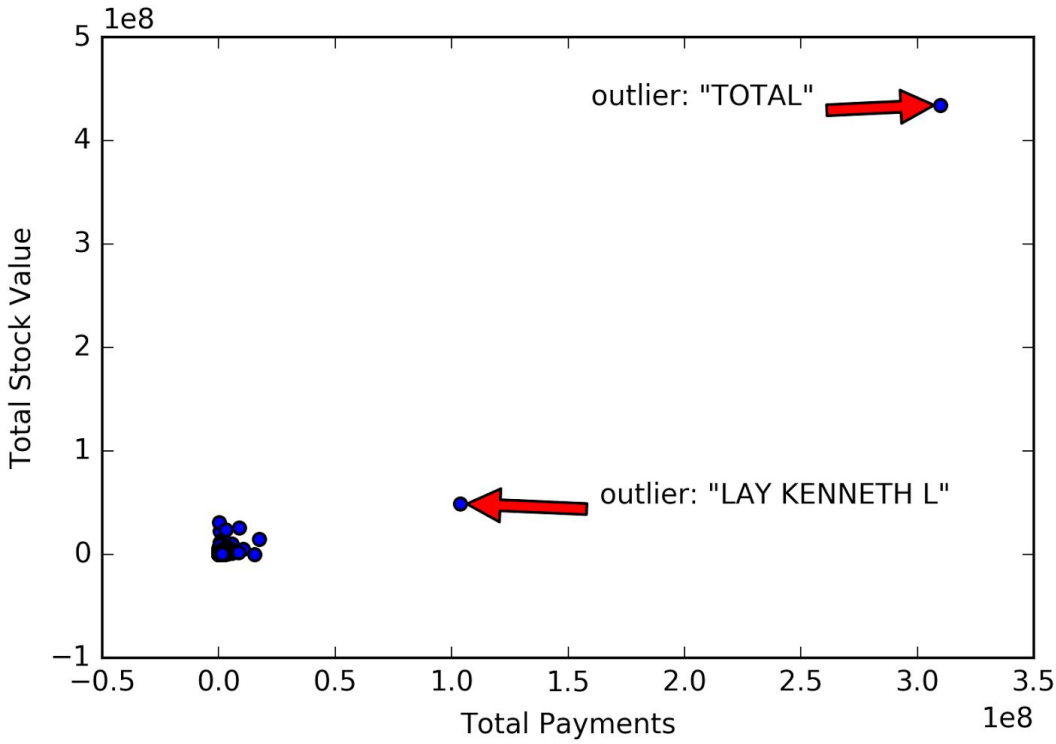
This main section is divided into three subsections, preprocessing, features manipulation and algorithm tuning.

Preprocessing

Four procedures are conducted to preprocess the data. They are how to deal with outliers, missing values, unbalanced classes and feature scaling.

Outliers

Outliers need to be examined and processed accordingly. One of most useful ways of identifying outlier is by plot. Following plot is scatterplot of two features, total payments and total stock value. I use first and second places of total payments to trackdown outliers.



Obviously, there are two “outstanding” outliers in this plot. And they are “TOTAL” and “LAY KENNETH L” . I simply remove “TOTAL” and keep “LAY KENNETH L”. The reason to remove “TOATL” is apparent. “TOTAL” is merely a statistical value that can not represent any information regarding POIs and non POI. Mathematical speaking, “TOTAL” is a linear combination of other rows and can always be removed as a redundant row without reducing rank of the matrix formed by our data. “LAY KENNETH L” is Ken Lay who is the founder and Chairman of Enron. First, he is not only a valuable data point in 145 points but also a precious POI in 18 POIs. “LAY

KENNETH L” amounts to information the whole data set provided. Second, I also think he serves as a guide light of searching POI. The model which will be built would distinguish people like Ken Lay from other people. So I believe the first step of a successful algorithm of this task is whether the algorithm can identify Ken Lay as a POI. So I keep Ken Lay in the dataset.

Missing Values

‘NaN’ values are replaced by median of all the numbers in the feature. Replacing missing numberic values with zeros is always the first option. However, I don't think it is a proper method dealing with missing values in this dataset. Replacing ‘NaN’ with most probable values is the essence of dealing with missing values without lossing any item in the dataset. In this dataset, I take the feature 'salary' to explain the idea. If someone's salary is missing and it is my obligation to guess a number. It is unreasonable to have a thought that he or she earns nothing or most in his or her company. But it is true that no one can blame me on making an assumption of a median of salaries. I believe that the medians of the feature is more probable than other values to replace the 'NaN's. I also conduct the comparison of performances between three different kinds of dealing with missing values. They are replacing by zeros, by medians and by results from a complex algorithm. The results show that the medians perform as well as the complex algorithm and better than zeros.

Unbalanced Classes

It is worth noting that the numbers of instances of each class are far from balance. I randomly select 'POI' instances from data set and add them into the dataset until the numbers of instances from two classes are equal. The number of 'non POI's is 7 times of that of 'POI's. The highly unbalanced numbers of two classes may cause problems in nearly every algorithm of classification. Most classifiers don't have a good performance on unbalanced data. The reasons is as follows: 1. these algorithm take accuracy as metric which can't be influenced by minority class. 2. these algorithm assume that both classes have the same prior probability. The problem can be overcomed by many methods such as under sampling or over sampling. And I choose to use over sampling which randomly add instances of minority class into data set.

Features Manipulation

This section is divided into three sub-sections, two new features, feature scaling and feature selection.

Two New feature

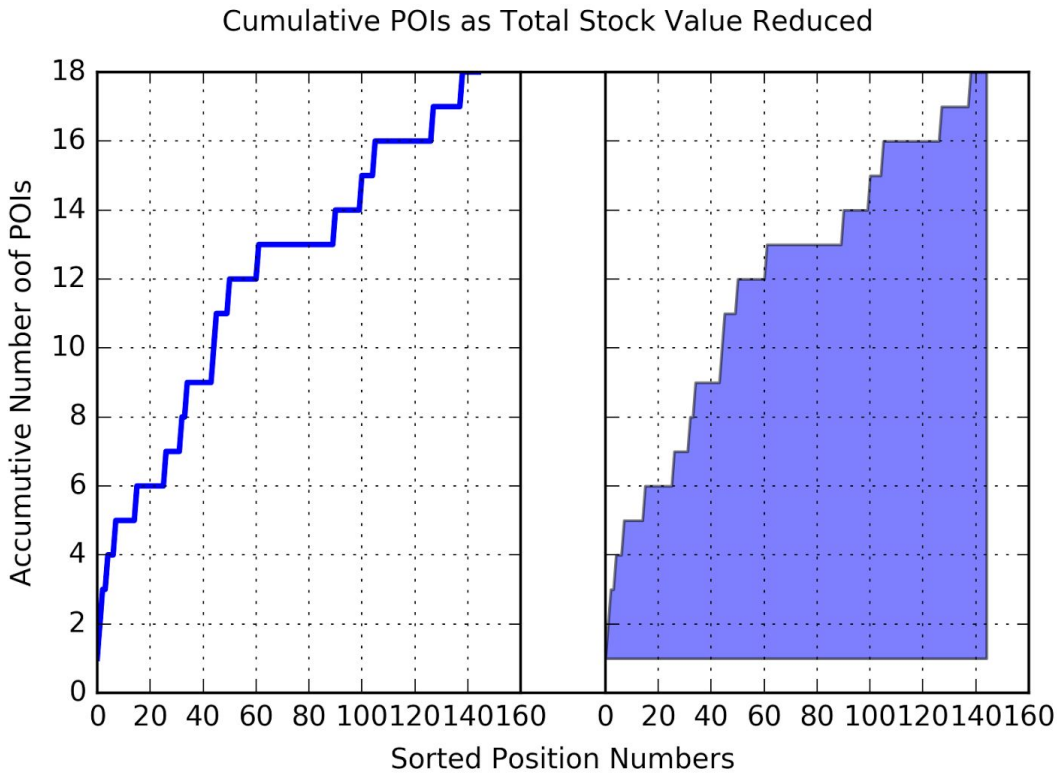
The first new feature is number of 'NaN's and second is sort score.
The first new feature is just number of 'NaN's of a instance. The reason for calculating this feature is simple. People usually gather more information about what

they are interersting than not. 'POI's in this dataset tend to have less 'NaN's than 'non POI's. And a statistical test ($t = -4.53, pvalue = 6.23e-6$) verify that number of 'NaN's in 'POI's is statistically less than that of 'non POI's.
The second new feature is sort score. Before the sort score is proposed. It is very useful to examine the characteristics of a POI by the dataset and the story of Enron. Two characteristics I have learned is as following.
1. POIs are people in high ranks of Enron. The scandal of Enron is not individual crime or an occasional event. It is organized in level of entire Enron Corp and last for almost 5 years long. People who managed to commit the crime in such a scale and length must have power and resource needed to accomplish it. And people who have enormous power and resources must be at top ranks of Enron.
2. POIs is rare. Only 18 of 145 persons are identified as POI. Based on above two characteristics of POIs, combining the hierarchical nature of Enron, I think the problem of identifying POIs became finding a few people on the top of company pyramid. A new feature is created according to this conclusion. The new feature should reflect above two characteristics of a POI. Firstly, the new feature contains information of ranks in Enron. Secondly, the new feature can create distance minority of top players from most of executives.
The first quality of the new feature is by using sorted position numbers instead of real numbers in dollars or counting of emails. I think actual numbers give us less information of ranks than the sorted position number. Taking CEO of Enron, Jeffrey Skilling, for example, salary of 1,111,258 dollars does not directly give his rank to me than number 1 of salary in the dataset do.
The second quality of the new feature is guaranteed by transforming the position number.
The function I have chosen is,

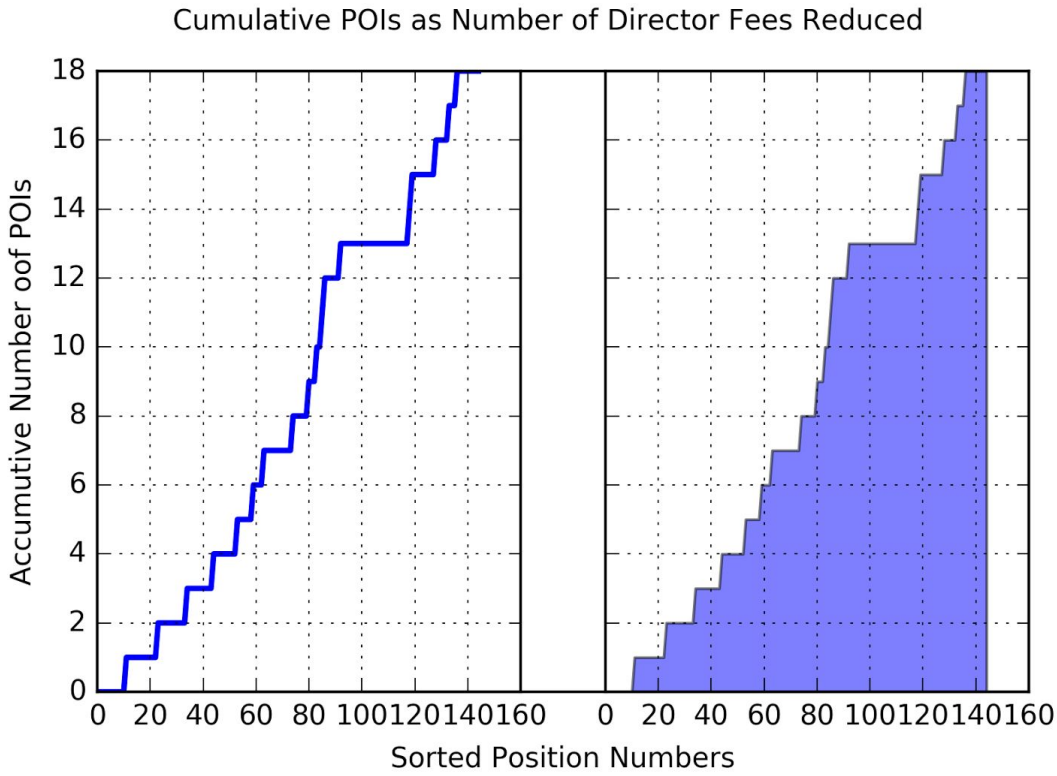
$$\frac{1}{(position\ number)^{p}}$$

p is parameter in this tranform. After this transformation of position number, first (position number 1) become biggest. Most important, this function can drops slowly or dramatically in beginning positions by choosing proper value of p. Thus distances will be created between the top ranks and the rest. For example, if $p = 2$, the transformation of 1 is still 1 and number 2 is only 0.25. Top ranks are more easily to be isolated by this transformation.
More than one features are needed to construct the new feature. The transformations of values of these features will be summed into a new value which is the new feature. There are two reasons for using sum of several original features. First, I noticed that too many ‘NaN’ appears in the dataset. Using several features mitigate this problem. Second, I noticed that nearly every POI have one or two leading features but not all of features. For example, Jeffrey Skilling had highest

salary, Kenneth Lay had highest total stock value and Christopher Calger had top rank bonus. Adding the values means that if a person excel at one or two these features then the person may be a POI.
In conclusion the new feature (sort_score in the program) is created in following algorithm:
1. Finding k features;
2. Sorting each features and replacing the actual value of feature by the sorted position (1, 2 ,3,....n);
3. Transforming the position number using above equation;
4. Adding k transformations of features of a person up to a new value for each one;
And the new value represents the new feature which I called sort score.
One of the problems of the section above left is the first step of construction the new feature, finding k features. I use integrals of cumulative of POIs to represent importance of a feature. And the integrals are on sorted position numbers of that feature. I will take two features to illustrate why the integrals represent the importance of feature. See following two plots of cumulative of POIs from two features. The first is “total_stock_value”.



The second is “director_fees”.



X axis is the sorted position number range from 1 to 145. Y axis is cumulative of POIs. For example, point (10, 3) means that first 10 ranks of that feature contains 3 POIs. My new feature use ranks to distinguish POIs from non POI. What makes a good features that can be used to produce the new feature is that top ranks contains more POIs than lower ranks do. As shown by above plots, the feature “total_stock_value” contains more POIs in its high ranks than “director_fees” do. So the integral which is indicated by blue region of the plot of “total_stock_value” is bigger than that of “director_fees”. So integral of cumulative POIs is a measure of importance of features. Integrals of all the number features are calculated and sorted from big to small as following chart. And I select first 6 of them to construct new feature. These five features are 'nan_number'(the first new feature), 'shared_receipt_with_poi', 'from_poi_to_this_person', 'from_this_person_to_poi', 'bonus' and 'total_stock_value'.

Features	**Integrals**
nan_number	1874
shared_receipt_with_poi	1848
from_poi_to_this_person	1802
from_this_person_to_poi	1771
bonus	1755
total_stock_value	1729

total_payments	1709
exercised_stock_options	1707
restricted_stock	1704
to_messages	1610
long_term_incentive	1591
salary	1566
expenses	1481
deferred_income	1353
from_messages	1310
restricted_stock_deferred	1178
director_fees	1176
loan_advances	1080
other	1078
deferral_payments	980

The other problem is how to choose parameter p in construction of new features. I solve this problem by simple searching. I just run tests on different values of p and see which p has the best F1 score. In this task I use Gaussian Naive Bayes and only the new feature for testing(I used provided code tester.py to run test). Following is the performance of different p value.

Value of p	Precision	Recall	F1
0.4	0.4144486692	0.218	0.2857142857
0.2	0.3880994671	0.2185	0.279590531
1	0.4507422402	0.167	0.2437066764
5	0.4466292135	0.159	0.2345132743
10	0.4466292135	0.159	0.2345132743
2	0.4442877292	0.1575	0.2325581395
3	0.4442877292	0.1575	0.2325581395

And the result is that p value for highest F1 score is 0.4.

Feature Scaling

I simply scale all the features by standardizing. I will use several algorithms to build classifiers. And some of them need to be scaled, such as SVM. Some of them don't need to be scaled, such as decision tree. There are two reasons why I used scaled features in all these algorithms. First, the numbers in all the features are relatively important. The absolute value is not important in this data set. And which make the

value meaningful is its comparisons to other values. Since scaling doesn't change relative order. The information of the features is retained. Second, scaled features perform as well as original feaures when algorithms which don't need scaling is implemented.

Feature Selection

I use a simple sorting procedure to select features. I build a Gaussian Naive Bayes model on each feature. Then I test these models on provided code tester.py. Then I select features corresponding to first n highest scores of F1 and precision. (Why not recall? I guess it has to do with some principles of convicting suspects.) n is chosen by comparasion of SVM models on each n (1, 2, 3, ...20). The preformance is as following.

Number of Features	Precision	Recall	F1
1	0.8791058394	0.2964615385	0.4433962264
2	0.8616053776	0.5028461538	0.6350609608
3	0.8681912022	0.4205384615	0.5666165725
4	0.7098222638	0.8755384615	0.7840192871
5	0.723718561	0.8743076923	0.7919177844
6	0.7782566694	0.8056153846	0.7916997392
7	0.7940651495	0.8213076923	0.8074567042
8	0.8110787576	0.8999230769	0.8531942824
9	0.8111650485	0.8997692308	0.8531728665
10	0.8033379694	0.8886153846	0.8438276114
11	0.8157173959	0.8934615385	0.8528213224
12	0.8157173959	0.8934615385	0.8528213224
13	0.829820902	0.8874615385	0.8576738654
14	0.829820902	0.8874615385	0.8576738654
15	0.8245138889	0.9133076923	0.8666423358
16	0.8245138889	0.9133076923	0.8666423358
17	0.8245138889	0.9133076923	0.8666423358
18	0.8245138889	0.9133076923	0.8666423358
19	0.8245138889	0.9133076923	0.8666423358
20	0.8245138889	0.9133076923	0.8666423358

As we can see, for n>8, the f1 score is increasing in a small step. So I choose n to be 8. There are total 15 features for use. And they are as following.

salary'

sort_score'
deferral_payments'
long_term_incentive'
exercised_stock_options'
bonus'
restricted_stock_deferred'
total_stock_value'
expenses'
loan_advances'
from_messages'
to_messages'
deferred_income'
nan_number'
director_fees'

Algorithm Tuning

Three algorithms are used to construct models. They are Gaussian Naive Bayes, Desision Tree and Support Vector Machine. And some parameters of Support Vector Machine are tuned by using GridSearchCV.
Performances(which is measured using tester.py) of three algorithm is as follows:

Algorithm	**Precision**	**Recall**	**F1**
Gaussian Naive Bayes	0.5729032258	0.9904615385	0.7259196617
Decision Tree	0.8669841696	0.9942307692	0.9262577039
Tuned Support Vector Machine	0.8675655831	0.9997692308	0.9289875273

As we can see the tuned Support Vector Machine perform better than other two algorithm. Support vector machine are finally used for this task.
Gaussian Naive Bayes and Desision Tree are build using default parameters in sklearn package. And two parameters, 'kernel' and 'C', of Support Vector Machine are tuned using GridSearchCV. Parameter tuning is a essential step for building any models. Different kinds of models represent various tools of machine learning. Parameters of the models are used to specify a particular tool to solve a certain problem. I think that parameters are like hardware configuration of physical world. The parameters define the structure, capacity, complexity, speed, robustness and so on of a model. Like hardware configuration, there are good and bad ones. Different problems reflect different nature of the world. And the first step of solving various kinds of problem is to find a proper tool. This is also true for machine learning.

Tuning parameters is building proper tool for the particular problems. The result of tuning parameters, 'kernel' and 'C', in this task in as following:

Parameter C	Parameter kernel	Score
0.1	linear	0.797752809
0.1	rbf	0.7640449438
0.1	poly	0.6292134831
0.1	sigmoid	0.5168539326
1	linear	0.8258426966
1	rbf	0.8258426966
1	poly	0.6460674157
1	sigmoid	0.5168539326
10	linear	0.8370786517
10	rbf	0.8595505618
10	poly	0.7471910112
10	sigmoid	0.5168539326
100	linear	0.8483146067
100	rbf	0.893258427
100	poly	0.893258427
100	sigmoid	0.5168539326
1000	linear	0.8539325843
1000	rbf	0.893258427
1000	poly	0.9101123596
1000	sigmoid	0.5168539326
10000	linear	0.8483146067
10000	rbf	0.893258427
10000	poly	0.9157303371
10000	sigmoid	0.5168539326
30000	linear	0.8483146067
30000	rbf	0.893258427
30000	poly	0.9157303371
30000	sigmoid	0.5168539326

The parameters of best performance are 10000 for C and ‘poly’ for kernel.

Evaluation

Many procedures in this project involves evaluations. A fair validation of the performance is crucial in evaluation. Validation is the process justifying the performance of the model. The procedures of validation must be proven to have the ability to predict the outputs of unknown instances. The most common mistake of evaluating models in machine learning is take scores of the model on the training data as the performance of the model. Because model is built upon training data.It may reflect some noise of the training data. To see how the model perform, the best way, of course, is testing it on actual working conditions. However, this is not always a feasible way. In a pratical way, validation is done by testing models on some data that the model have never seen. I evaluate the model by the cross-validation of 1000 folds.(The same as tester.py) I randomly split the dataset for 1000 rounds. In every round, the dataset is split into training and test set by 0.9/0.1 standard. The model is analyzed on training data and validated on test data. The results of 1000 rounds are accumulated to evaluate the performance of the model.

Three metrics are calculated to represent the performance of a model.They are precision, recall and f1 scores. For the tuned SVM model in this project, the precision is 0.868, recall is 0.999 and f1 score is 0.929. The precision describe the accuracy of the model. In this dataset, 0.868 precision means that if a person was predicted by this model to be a POI, then with 86.8% possibility the person would truely be a POI. The recall describe the coverage of the model. In this dataset, 0.999 means that if a person was a POI, then there will be 99.9% possibility for the model to predict the person to be a POI. And f1 score is used to combined the precision and recall. It is always easy to achieve only high precision or only high recall. High precision indicate that the model is very confident to predict some instance belonging to certain class. If the model only predicted the most obvious instance, high precision can be obtained. However, high precision can't guarantee that the model can find out all the instances in one class. The recall is used to compensate the weakness of precision. High recal means that the model can predict high percentage of the class. Thus 100% recall is achieved by predicting every instance as one class. The conclusion here is that, in most circumstance, precision and recall must both provided to evaluate performance of a model.