P5: Identify Fraud from Enron Email

By Rhalimi Mouna

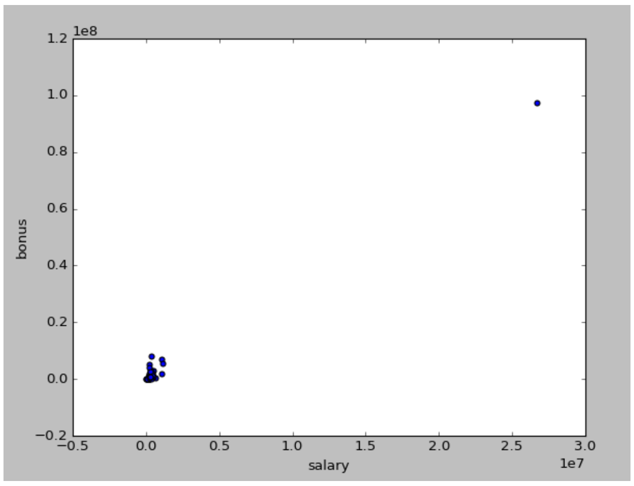
Project overview:

Enron Corp was one of the largest companies in the US, that collapsed in 2002 due to fraud. During the investigation, thousands of email and financial data went public.
The goal of this project is to create a model capable of predicting people of interests based on their emails and financial information. We use emails of 146 executives at Enron to identify the persons of interest in the fraud case.
We will start by investigating and cleaning our data, then comparing two algorithms and validating the one that provided more significance and precision.

Some statistics:
The dataset has 18 person of interest (Poi), which leaves 128 non Poi person.
These features are the features with the higher missing values:
deferral_payments
number of missing values: 107
restricted_stock_deferred
number of missing values: 128
loan_advances
number of missing values: 142
director_fees
number of missing values: 129
I used 20 of the features, then worked with few automatically selected ones, we will see that later.

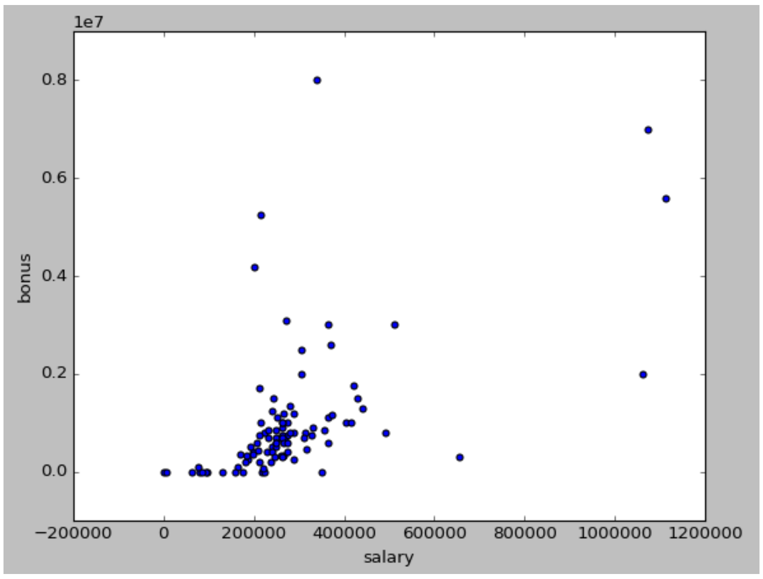
Enron data:

We plot the data to check for outliers, and to get a view about how it is distributed, from the plot 1, we notice one clear outlier, when we check in the data, we see that it is the total of salaries and bonuses, that's one outlier to remove.



Plot 1: before removing total

Now the data looks much better , of course there still other outliers, but those are information of person that we will want to keep and investigate.



Plot 2: After removing total

Features processing:

We first added new features called to_person and from_person, and append them to the dataset. Our main interest is the messages from and to poi, having a rate is much better than having numbers.
For Example, a person can send 200 from 10000 of his email to poi, while another may send 200 to poi out of 300, my intuition is that the second one is a poi , since 2/3 of his emails are sent to poi, while the first sent only 1/5 of total of his sent emails to poi.

to_person=from_poi_to_person/to_messages
from_person=from_person_to_poi/from_messages

While applying from_person only on the classifier we picked, we got:
Accuracy: 0.80933 Precision: 0.34191 Recall: 0.46500

when applying to_person:
Accuracy: 0.80040 Precision: 0.28706 Recall: 0.33500

when taking both features:
Accuracy: 0.79653 Precision: 0.28192 Recall: 0.34000
this shows that ‘to_person ’ improve the precision and recall of the model.

The model selected uses gridsearchCv and selectBest as a parameter, after running on all possibilities, we found out that 16 gives the best result, so we keep it and go back to select the best 16 features:
Results with k=16 for Decision tree algorithm:
Accuracy: 0.81387 Precision: 0.35484 Recall: 0.48400

The 16 best features and their score:

Feature	**Score**
exercised_stock_options	25.09
total_stock_value	24.46
bonus	21.06
salary	18.57
from_person	16.64
deferred_income	11.59
long_term_incentive	10.07
restricted_stock	9.34
total_payments	8.86
shared_receipt_with_poi	8.74
loan_advances	7.24
expenses	6.23
from_poi_to_this_person	5.34
other	4.20
to_person	3.21
from_this_person_to_poi	2.42

So we fix number of features to k=16 , and change pca values in parameters before applying gridsearchCv again, we put all number of features, the best recall and precision values are found when the n_component is equal to 2, we get :

Accuracy: 0.82167 Precision: 0.35307 Recall: 0.40550
We then fix n_components to 2.

Tunning an algorithm is a very important phase to get an optimal model parameters that gives our model a good accuracy and most importantly, good recall and precision values.
I tuned features selections and parameters of the Decision tree, the best feature numbers were 16, while 2 was the optimal number of pca composant.
For the rest of parameters, we just try several things till we get good models and keep them.

“Algorithm Tuning where discovering the best models is treated like a search problem through model parameter space.” scikit-learn.org

Choice of Algorithm:

At first we split the data to 30% for testing and the rest for training, then we used naive bayes algorithm. The results were not good, we actually got 0.74720 for accuracy, 0.23585 precision and 0.400 recall.
The other algorithm is Decision tree, while working on the same list of 16 features selected by Kbest. We will scale it using MinMaxscaler, select components with PCA and split it into training and testing data. We tune the parameters with GridSearchcv one parameter at time.
It seems that working with 2 components and 5 tree split gave an accuracy of 0.80580, Precision of

0.35099 and a Recall of 0.42400

Validation of the model:

A classic mistake would be to only relay on accuracy instead of recall and precision, we have small number of poi compared to the population size.
Also to not split the data before using it means that we will train the model and test it on the same data, this will causes over-fitting, a model that will repeats the labels of the data it has just seen will have perfect scores but won’t be useful to predict new data.
Here comes model validation technique that helps generalize statistical analysis to independent dataset.
Validation helps avoiding over-fitting, by splitting data into training data and test data, then train the model on train data to select the best model, and test it to see if it can be generalized on training data.
Since the dataset is small, we would want to split our data while keeping the same percentage of Poi and non Poi, for that we use StratifiedShuffleSplit , which is a technique that returns stratified randomized folds while preserving the percentage of samples for each class .
This is another classic mistake, making wrong validation by splitting the sorted data without taking into account the structure of our dataset.

Precision is True Positive /(True Positive + False Positive)
With precision, we want to see how much people who were detected as Poi are actually Poi.
I had around 0.35 in this evaluation metric, it is the probability that a person who is identified as Poi is actually a Poi .

Recall is True Positive /(True Positive + False Negative)
With recall, we want to detect as much as possible how much Poi there is in our data.
Recall means that the model will catch a Poi in the data in 42 time out of 100 time.