Machine Learning Project -- Enron Data

Long Wan, June 23th, 2016

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

Enron was one of the largest companies in the US, however, it bankrupted in 2002. The goal of this project is to identify persons of interest, who were believed to be responsible for company fraud, based on the dataset given. In this process, I will make some data cleaning and extract key features, and then use machine learning to train the features. Finally I will test the model to see its performance.

The enron dataset contains 146 samples and 21 variables. There are 18 POIs and 128 non-POIs. Variable list is showed as below,

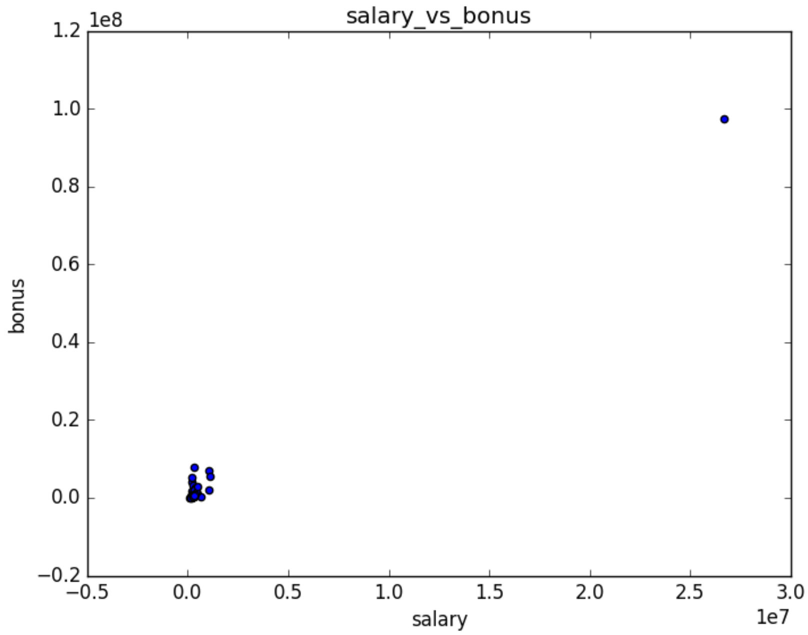
```
['salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercise
d_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi',
'restricted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advanc
es', 'from_messages', 'other', 'from_this_person_to_poi', 'poi', 'director
_fees', 'deferred_income', 'long_term_incentive', 'email_address', 'from_p
oi_to_this_person']
```

Among these variables, “poi” is boolean type, and “email_address” is string. All others are numbers. The following table shows the number of missing values each numeric variable has.

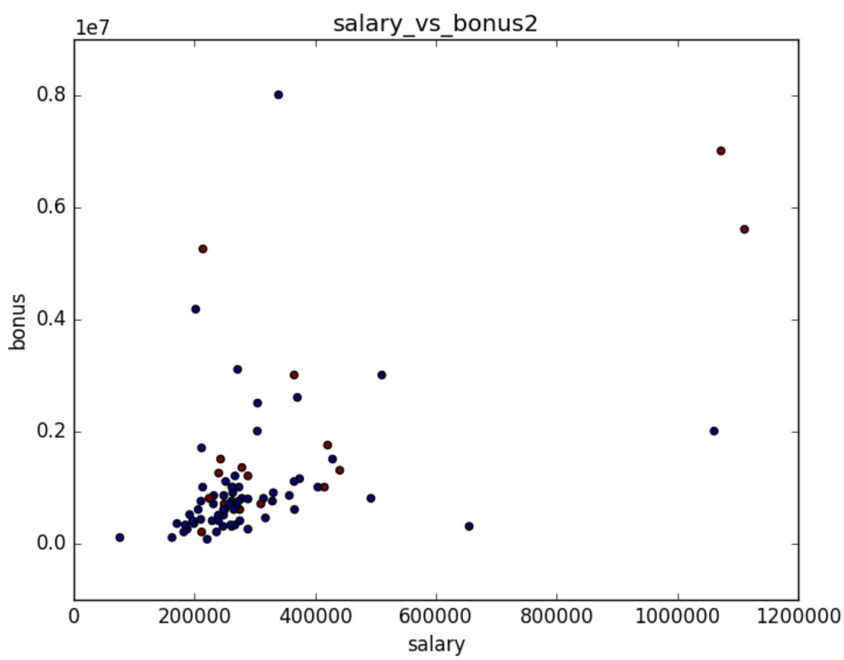
```
salary                       51
to_messages                  60
deferral_payments           107
total_payments               21
exercised_stock_options      44
bonus                        64
director_fees               129
restricted_stock_deferred   128
total_stock_value            20
expenses                     51
from_poi_to_this_person      60
loan_advances               142
from_messages                60
other                        53
from_this_person_to_poi      60
deferred_income              97
shared_receipt_with_poi      60
restricted_stock             36
long_term_incentive          80
```

All numeric variables have missing values, and the portion is pretty large in some variables, like “loan_advances”, “director_fees”. That would be harmful to machine learning, so I change those NaN values into 0. Also, I think it is referable when selecting features, since more valid values, the better.

When I scatter plotted the salary vs bonus, I found an outlier. See as below.



I checked the original table and got to know that the outlier was the total value, aggregating all values in a column. It was meaningless, so I deleted the row “TOTAL”, and the new plot was nicer.

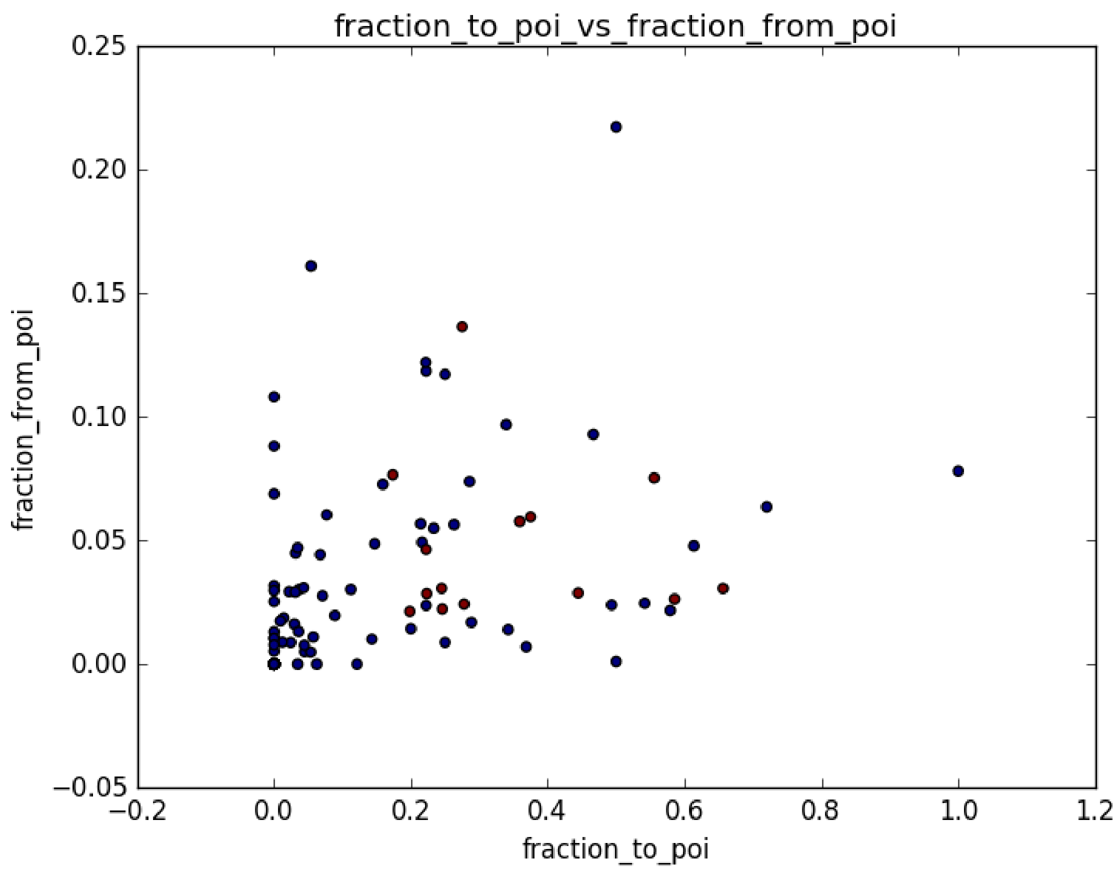


We are able to know the relationship clearly from this picture. Color distinguishes poi and non-poi group.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values.

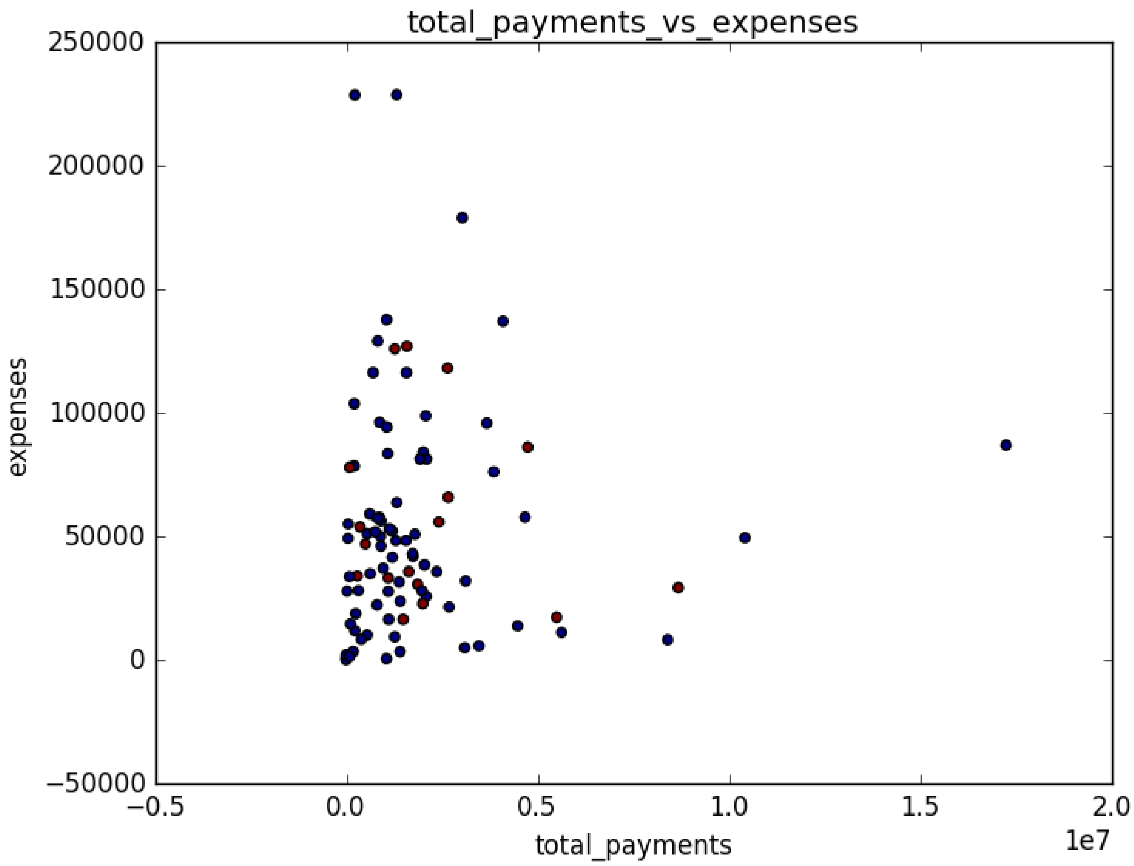
The final 4 features I selected were "fraction_to_poi", “exercised_stock_options”, 'expenses', 'shared_receipt_with_poi', which I believed was the best combination.

Persons of interest may have closer relationships with each other via e-mail. However, “to_messages” and “from_messages” are not good indicators to measure their close relationships. Fractions are. So I created two variables, “Fraction_to_messages” and “Fraction_from_messages” to measure the fraction of emails they received from and send to POIs. The picture below shows the allocation across POI and non-POI.



POIs (red points) locate topper and righter than vast majority of non-POIs do significantly.

I guess total_payments and expenses might be significant different across POIs and non-POIs, so I plotted it. There was an outlier, “LAY KENNETH L”. I deleted it and plotted again as below.



I haven’t got any significant differences. I deleted “LAY KENNETH L” temporarily only because I would like to see the above distribution clearly. The dataset used in the following process did include “LAY KENNETH L”.

Besides, based on the graph I drew in question1, there were few differences in terms of bonus and salary between POIs and non-POIs.

I decided to use both SelectKBest and DecisionTree to pick up features. But before implementing selection, I deleted some variables firstly. “from_this_person_to_poi”, “from_poi_to_this_person”, “to_messages” and “from_messages” were not necessary since I had already had fractions.

SelectKBest scores are highly based on the absolute value of each variable. Variables having large absolute value, like “salary”, obviously have higher variance than “frantion_to_poi” has. So I had to create a function called “scaler” to rescale remaining values based on the theory of MinMaxScaler in order to make scores more reliable. Specifically, this function regards “NaN” as an invalid value, so “NaN” would remained the same and other numeric values would be changed. One should notice that the “scaler” was only used to select features with SelectKBest method. It was not used in DecisionTree method and the final dataset to be tested.

However, dataset with “NaN” cannot be trained appropriately. There are several ways to deal with NaN issue, filling them by mean, median or mode of each column, using Imputer function. I also got to know that replacing NaN with median is the most robust way. So I used median method to replace NaN before selecting features.

Then came DecisionTree. Features with scores are below,

```
salary:0.0941792978552
expenses:0.0234501973531
fraction_to_poi:0.422649841967
long_term_incentive:0.328154447986
shared_receipt_with_poi:0.131566214838
```

Then came SelectKBest. Features ordered by scores are below,

```
[('exercised_stock_options', 5.9875618635559071),
 ('fraction_to_poi', 4.7914665456160836),
 ('deferred_income', 3.1663170230177791),
 ('total_payments', 2.0561050377373764),
 ('restricted_stock', 1.7848452804097328),
 ('shared_receipt_with_poi', 1.5083678497274913),
 ('restricted_stock_deferred', 1.1510950255387684),
 ('fraction_from_poi', 0.87490047242806024),
 ('other', 0.82047218866354876),
 ('long_term_incentive', 0.71403837418666527),
 ('salary', 0.69176038911741278),
 ('expenses', 0.12264743550958482)]
```

I compared the two lists and “fraction_to_poi”, “shared_receipt_with_poi”, “salary”, “long_term_incentive” are listed on both two lists so I selected them as the potential features. Meanwhile, I took “exercised_stock_options” and “deferred_income” into consideration as they topped SelectKBest list. I tried many combination of these features, as well as many kinds of classifiers for many times and found that the 4 features, “fraction_to_poi”, “shared_receipt_with_poi”, “exercised_stock_options” and “deferred_income” would be the best combination. Performances go down if more or less than 4 features were selected. I finally made the determination as mentioned at beginning.

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

I ended up using Gaussian Naïve Base algorithm. I also used DecisionTree and RandomForest.

Below are what I got from poi_id.py code for the three algorithms.

```
Random Forest Report
             precision    recall  f1-score   support

        0.0       0.88      0.95      0.91        39
        1.0       0.00      0.00      0.00         5

avg / total       0.78      0.84      0.81        44

Gaussian Naive Base Report
             precision    recall  f1-score   support

        0.0       0.93      0.95      0.94        39
        1.0       0.50      0.40      0.44         5

avg / total       0.88      0.89      0.88        44

Decision Tree Report
             precision    recall  f1-score   support

        0.0       0.92      0.87      0.89        39
        1.0       0.29      0.40      0.33         5

avg / total       0.85      0.82      0.83        44
```

The result above highly depended on how I split dataset into training and testing group. Performances might be quite different if I changed “test_size” or split them randomly again. So we should test them for several times and get the average value.

The following table shows the performance of Gaussian Naïve Base by running tester.py.

```
Accuracy: 0.86257     Precision: 0.52722    Recall: 0.36800
F1: 0.43345    F2: 0.39166
```

The following table shows the performance of Random Forest by running tester.py, where min_samples_split = 8, when it hit the best performance.

```
Accuracy: 0.86921     Precision: 0.57538    Recall: 0.32250
F1: 0.41333    F2: 0.35358
```

The following table shows the performance of Decision Tree by running tester.py when it hit the best performance, where min_samples_split = 12.

```
Accuracy: 0.87079     Precision: 0.58341    Recall: 0.33400
F1: 0.42480    F2: 0.36523
```

As we can see, Gaussian Naïve Base has a better overall performance, especially its recall. Random Forest and Decision Tree are not bad, but their F1 and F2 scores are lower than previous one.

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? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one

you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier).

To tune the parameters of an algorithm means to try different values of a parameter until find the best performance. If I don't do this well, I would not find best ones, and the average performance might get worse when sample size get larger or more cross validations are done.

Take an example of how I tuned parameters.

When test DecisionTree, I changed the value of min_samples_split many times and finally got the best one. The following table shows the performance of Random Forest when min_samples_split = 6-14, respectively.

	Accuracy	Precision	Recall	F1 score	F2 score
6	0.849	0.458	0.307	0.367	0.328
7	0.853	0.477	0.313	0.378	0.336
8	0.854	0.484	0.321	0.386	0.344
9	0.858	0.506	0.327	0.397	0.351
10	0.862	0.529	0.336	0.411	0.362
11	0.870	0.575	0.334	0.422	0.364
12	0.871	0.583	0.334	0.425	0.365
13	0.871	0.583	0.334	0.425	0.365
14	0.865	0.547	0.338	0.417	0.365

See that when min_samples_split = 12 or 13, the DecisionTree algorithm has the best performance.

Also, I found that when min_samples_split = 8, the RandomForest algorithm had the best performance. There is no need to tune Gaussian Naïve Base.

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

I split the dataset into 2 group, 70% of which were training group while 30% of which were testing group, using cross_validation.train_test_split algorithm.

If the test size was set too large, training data would not be enough to train the model. If the test size was set too small, testing data would be too small to be tested.

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.

I used accuracy, recall, precision and f1 score to measure performance. Accuracy is not a good indicator since the number of POI is too small and I just took it as a reference.

Take the example of the following table to interpret the metrics.

```
Accuracy: 0.86257     Precision: 0.52722    Recall: 0.36800
F1: 0.43345    F2: 0.39166
```

Accuracy is 0.863, which means that 86.3% samples were predicted to be POI or non-POI correctly. Precision is 0.527, which means that 52.7% samples which were predicted to be POI were true POI, and the remaining were actually non-POIs. Recall is 0.368, which means that 36.8% samples which were POIs were predicted to be POI correctly, and others were predicted to be non-POI.

References:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.from_dict.html

http://stackoverflow.com/questions/9622163/save-plot-to-image-file-instead-of-displaying-it-using-matplotlib-so-it-can-be

https://civisanalytics.com/blog/data-science/2016/01/06/workflows-python-using-pipeline-gridsearchcv-for-compact-code/

http://stackoverflow.com/questions/31655950/scikit-learn-pipeline-grid-search-over-parameters-of-transformer-to-generate-da

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

http://stackoverflow.com/questions/25017626/predicting-missing-values-with-scikit-learns-imputer-module

https://github.com/scikit-learn/scikit-learn/issues/3782