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. 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']
```

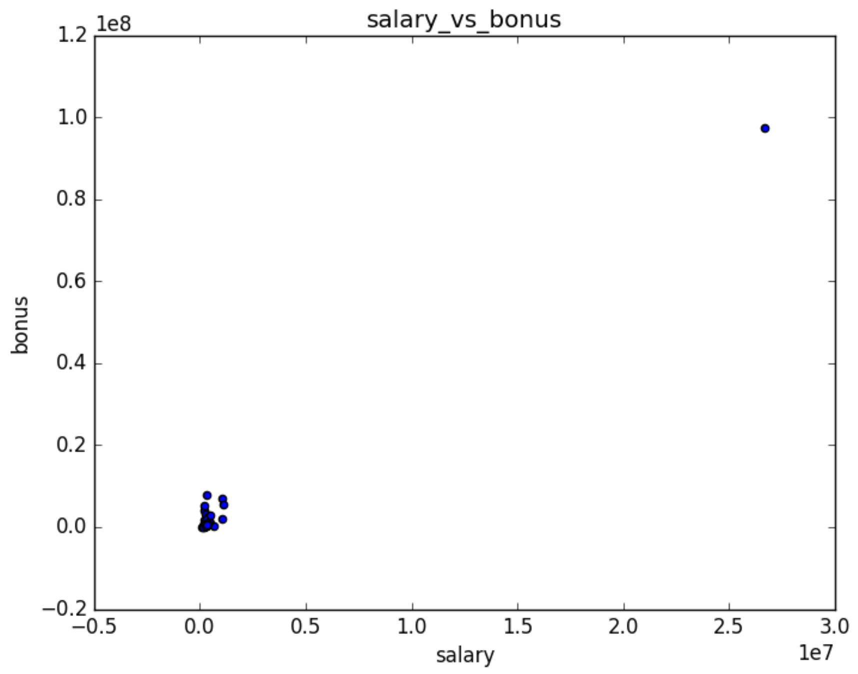
Among these variables, “poi” is boolean type, and “email_address” is string. All others are numbers. The following table shows the number of missing values each numeric variable has.

```
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
```

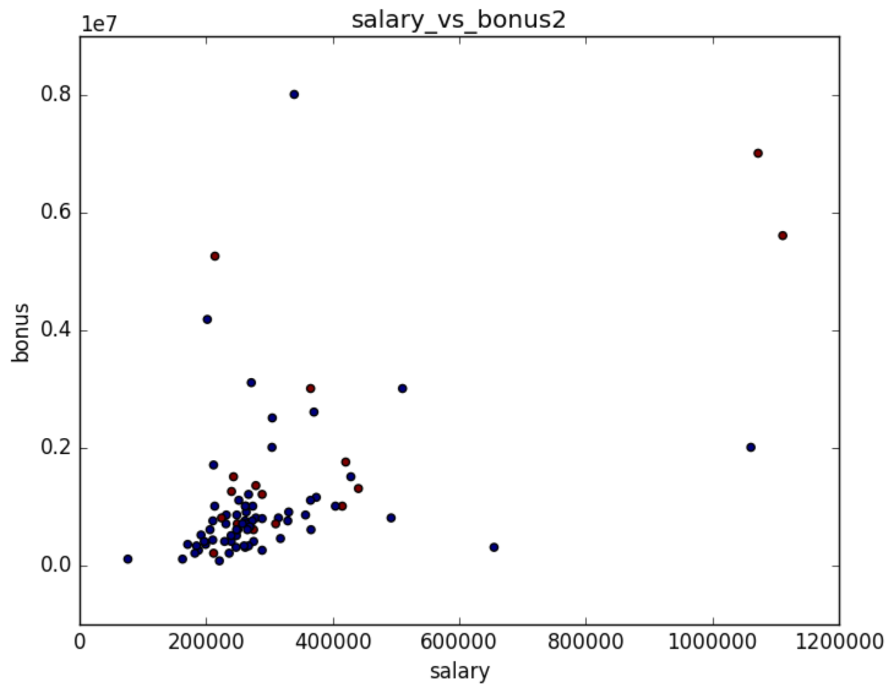
All numeric variables have missing values, and the portion is pretty large in some variables, like “loan_advances”, “director_fees”. That would be harmful to machine learning, so I change those

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.

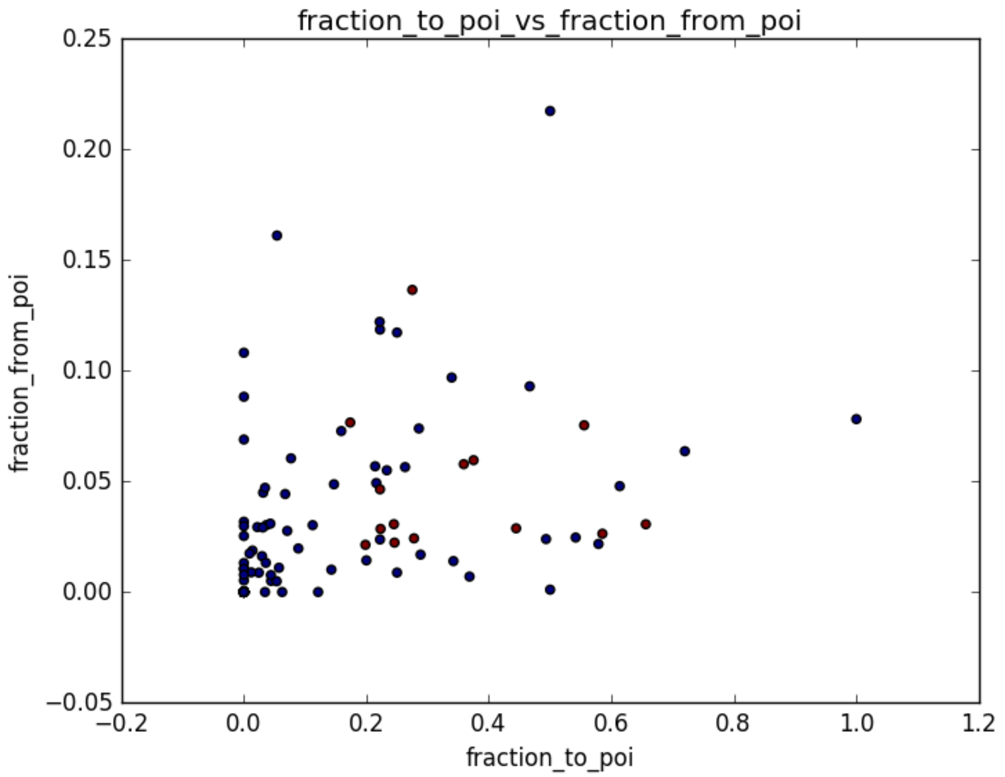


We are able to know the relationship clearly from this picture. Color distinguishes poi and non-poi group.

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.

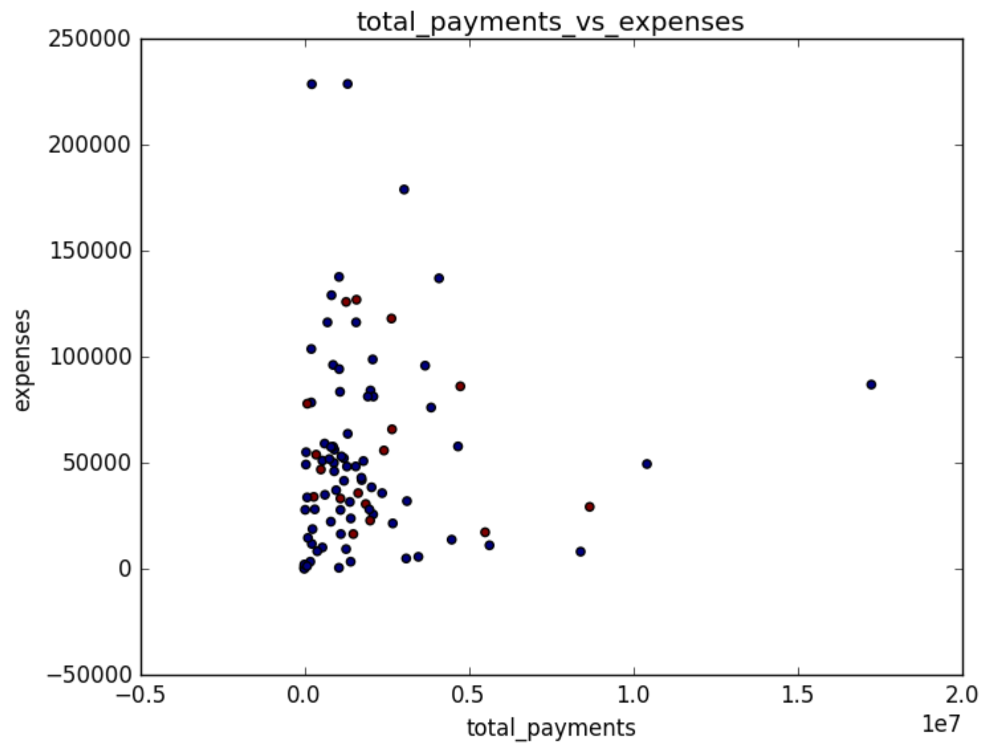
The final features I selected are “fraction_to_poi”, “shared_receipt_with_poi”, “fraction_from_poi”.

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.

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 SelectKBest method to pick up features. But before implementing selection, I deleted some variables. "from_this_person_to_poi", "from_poi_to_this_person", "to_messages" and "from_messages" were not necessary since I had already had fractions. "deferral_payments", "director_fees", "restricted_stock_deferred", "loan_advances" and "deferred_income" have too many missing values and I would not consider them.

And then, I used MinMaxScaler to rescale remaining values because variables have quite different scaling. For example, difference between 0.2 and 0.4 in fraction to poi may be similar to zero compared to difference between 100000 and 200000 in salary, but it did make sense.

Then came SelectKBest. Features ordered by scores are below,

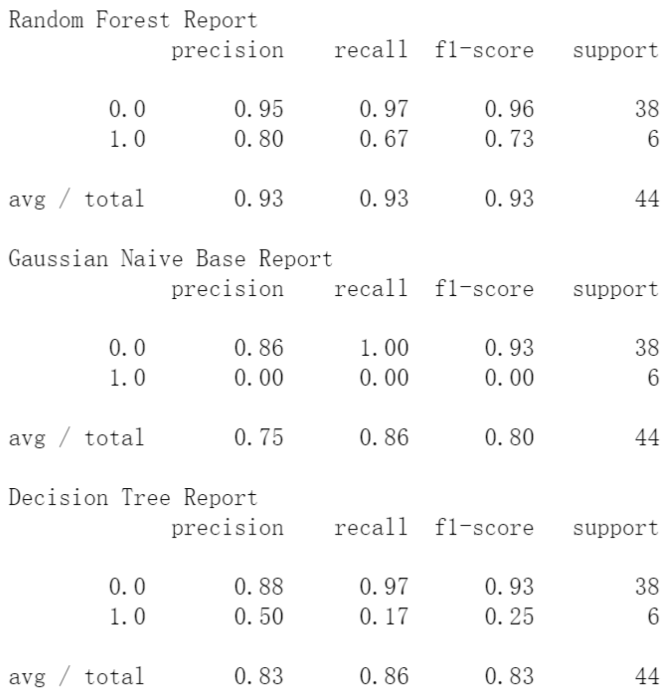
```
[('fraction_to_poi', 4.8651166745023104),
('shared_receipt_with_poi', 2.5830387081185138),
('fraction_from_poi', 0.89898709531446541),
('total_payments', 0.19332368085963492),
('exercised_stock_options', 0.12283159261136967),
('total_stock_value', 0.089030409573143676),
('other', 0.038906433204328586),
('restricted_stock', 0.014468151204615802),
('long_term_incentive', 0.012221465067343534),
('expenses', 0.0065551069122766455),
('salary', 2.8292142738982088e-05)]
```

“fraction_to_poi” and “shared_receipt_with_poi” are the two with highest scores. I tried the first 2 or 3 features, as well as many kinds of classifiers for many times and 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 Random Forest algorithm. I also used Gaussian Naïve Base and Decision Tree. Gaussian Naïve Base had the worst performance.

Random Forest and Decision Tree did not have significant difference. Random Forest was slightly better. Below are what I got from poi_id.py code for the three algorithms.



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

        0.0       0.95      0.97      0.96        38
        1.0       0.80      0.67      0.73         6

avg / total       0.93      0.93      0.93        44

Gaussian Naive Base Report
             precision    recall  f1-score   support

        0.0       0.86      1.00      0.93        38
        1.0       0.00      0.00      0.00         6

avg / total       0.75      0.86      0.80        44

Decision Tree Report
             precision    recall  f1-score   support

        0.0       0.88      0.97      0.93        38
        1.0       0.50      0.17      0.25         6

avg / total       0.83      0.86      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 Random Forest by running tester.py, where min_samples_split = 5, when it hit the best performance.

```
Accuracy: 0.87433      Precision: 0.42193    Recall: 0.35400
F1: 0.38499    F2: 0.36578
```

The following table shows the performance of Decision Tree by running tester.py when it hit the best performance.

```
Accuracy: 0.85511     Precision: 0.34830    Recall: 0.34900
F1: 0.34865    F2: 0.34886
```

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.

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 = 1,2,3,4,5,6,7,8, respectively.

	Accuracy	Precision	Recall	F1 score	F2 score
1	0.873	0.401	0.284	0.332	0.302
2	0.873	0.401	0.284	0.332	0.302
3	0.872	0.404	0.324	0.360	0.337
4	0.872	0.411	0.349	0.377	0.360
5	0.874	0.422	0.354	0.385	0.366
6	0.874	0.422	0.354	0.385	0.366
7	0.875	0.424	0.341	0.378	0.355
8	0.875	0.417	0.320	0.362	0.336

See that when min_samples_split = 5 or 6, the Random Forest algorithm has the best performance.

Also, I found that when min_samples_split = 4 or 5, the Decision Tree 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: 0.87433     Precision: 0.42193    Recall: 0.35400
F1: 0.38499    F2: 0.36578
```

Accuracy is 0.874, which means that 87.4% samples were predicted to be POI or non-POI correctly. Precision is 0.422, which means that 42.2% samples which were predicted to be POI were true POI, and the remaining were actually non-POIs. Recall is 0.354, which means that 35.4% 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