Machine learning Enron project: Documentation of work

Introduction

Enron scandal is one of the largest corporate fraud case in American history. Its intriguing downfall is still today taught as case study and uniquely well-documented. During its investigation, the Federal Energy Regulatory Commission (FERC) made the controversial decision to share online the Enron e-mail corpus.

The dataset used for this project includes these emails together with financial data for 146 executives at Enron. The objective is to identify all persons of interest (POI) in the fraud, that is, employees that were indicted for fraud, settled with the government, or testified in exchange for immunity.

This document will describe the machine learning techniques used in building such POI classifier.

1. The Enron Data

The distribution of the poi class is clearly unbalanced with only 18 people defined as poi and 127 non-poi. This could be a problem later on when training/validating a model (some are more sensitive to small dataset than others).

The data exploration (done in mini projects) revealed the presence of the grand total row merged with the real employees. It was of course manually took off the dataset right away.

There are also two people standing aside from the rest:" LAY KENNETH L" and "SKILLING JEFFREY K". One could argue they are noise or mistakes to be cleaned up. However they are actually the two main protagonists of the scandal and the ones that "generate" most of the money (because holding the top positions at Enron company). By no surprise, they are identified as Person of Interest and therefore must be kept in our dataset.

The dataset provided was actually built upon two. They were merged on the people's name whenever possible (left join). The biggest concern is that they didn't include the same people, hence these pieces of data were filled with missing values. That's the reason why 34 people doesn't have any email information and 36 others with no financial data (neither payments nor stock options). With almost half (70/145) of incomplete records, this could have a serious impact on the performances of the model.

2. Feature Processing

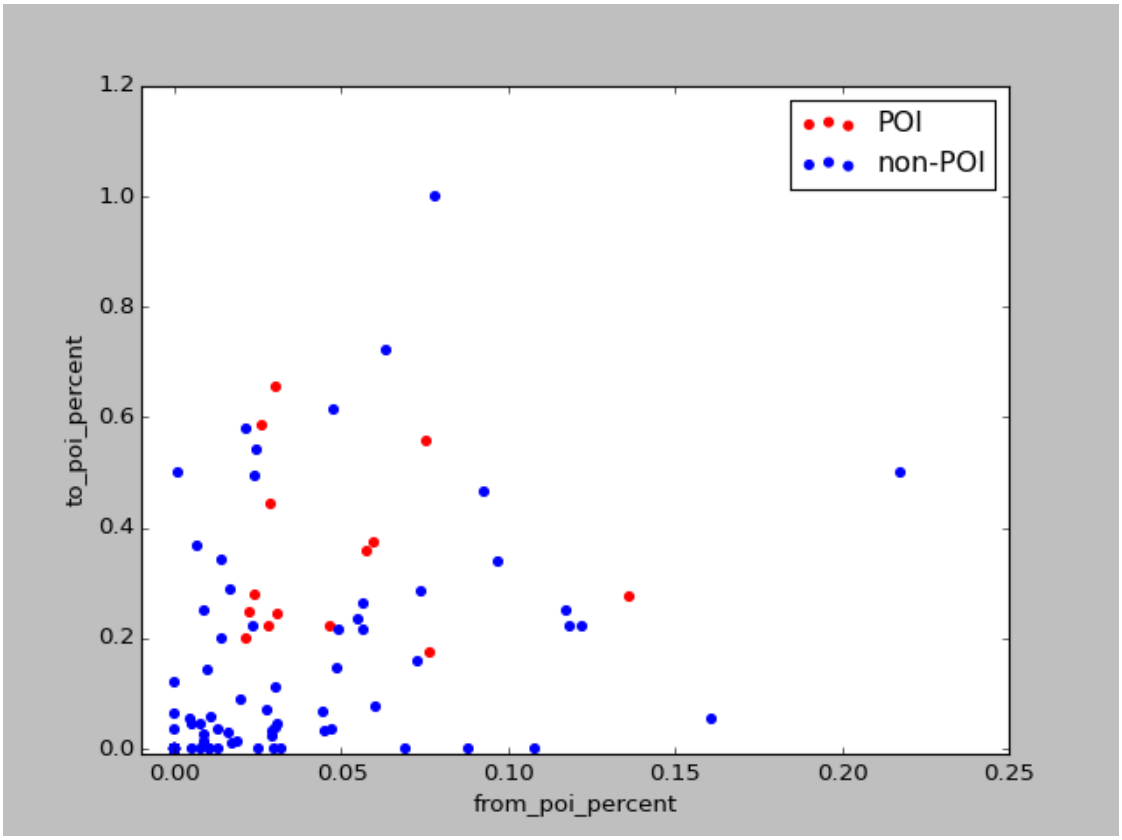
The data exploratory analysis I led, allowed me to identify the ‘salary', 'bonus', 'total_stock_value', 'exercised_stock_options' as significant features to start with. They indeed show a clear separation between poi from others.

I then leveraged the SelectKBest function offered by the sklearn Python library to refine the selection. Depending of the algorithm used, the list might be a bit different (in length or order).

I also created two new features to convert the number of messages from/to poi in percentage. They give a better idea of the relationship or closeness with poi than the number of occurrences itself. Strong connections could potentially lay out new poi:

- from_poi_percent = 'from_poi_to_this_person'/ 'from_messages'. The proportion of messages received from poi over the whole set of messages received.
- to_poi_percent = 'to_poi_to_this_person'/ 'to_messages'. The proportion of messages sent to poi over the whole set of messages sent.

According to the chart below, poi tend to gather above 0.2 to_poi_percent and 0.025 from_poi_percent.



Only to_poi_percent will be playing a significant role in two of the models trained later. Sending information to poi (upward) looks more suspicious that receiving some (downward). Content of the message must be different too.

I initially thought of scaling the features to improve the performance of the model, but it proved later to have hardly any impact on the model. So I skipped this step.

3. Algorithm Selection and Tuning

I first ran a Gaussian Naive Bayes model as a reference over the fifth most relevant items…

Features	score
exercised_stock_options	5.345
to_poi_percent	4.205
Bonus	2.427
Salary	1.699
Total_stock_value	0.164

…and was surprised to see how good it fit as is:

Accuracy	Precision	Recall
0.849	0.456	0.3

I then trained a K nearest neighborhood model but noticed it came out to a single rule assigning all the people to non-poi (the size of the poi sample being too small in the dataset), that is underfitting. So not very interesting.

I tried applying a min-max scaling transformation to the data, thinking the different scales between financial and emails data could have a significant impact on the models' accuracy. It however didn't showed any improvement to neither the GaussianNB nor the KNearestN eighborhood models (accuracy and F1 score stayed exactly the same). I therefore disabled this transformation step.

Finally,I built a basic decision tree to compare with the GaussianNB. Unlike what I did previously for the two first models, I preferred leveraging the *feature_importance_* attribute returned by the DecisionTreeClassifier function (over the SelectKBest function) to select the best features.

Starting with all the features, the tree algorithm selected only 5:

Features	importance
bonus	0.39
expenses	0.22

to_poi_percent	0.17
shared_receipt_with_poi	0.16
exercised_stock_options	0.06

And surprisingly, it already met the requirements with default parameters:

Accuracy	Precision	Recall
0.83364	0.40	0.35

The GaussianNB model got slightly improved by including a sixth parameter (‘deferred_income': 4.205 score):

Number features	Accuracy	F1 score
5	0.849	0.362
6	0.852	0.404
8	0.846	0.405
10	0.840	0.34

The decision tree, on its side, hits its best score with the parameters tuned as follows:

- criterion ='entropy'
- splitter='best'
- min_samples_split=2
- max_depth=2
- Number of features: 2

Parameter tuning can be simply thought of as process to enable the algorithm to perform the best. That is, seeking how to set the parameters to their optimal values so that the learning task complete in the best way possible (in our case the highest F1 score).

Even though the algorithm was given 5 features, it eventually kept only 2:

Features	importance
Expenses	0.78
to_poi_percent	0.22

This optimal combination was learned by using the sklearn GridSearchCV function (and manually confirm afterward). That’s how the model performed after tuning:

Accuracy	Precision	Recall
0.805	0.47	0.55

So both the accuracy and F1 score got improved.

Trying out other combinations failed to discover a better set of parameters:

Criterion	splitter	Accuracy	F1 score
‘entropy”	‘best’	0.805	0.506
‘entropy’	‘random’	0.808	0.032
‘gini’	‘best’	0.803	0.499
‘gini’	‘random	0.807	0.013

min_sample_split	max_depth	Accuracy	F1 score
2	2	0.805	0.506
2	4	0.856	0.455
2	5	0.845	0.383
4	2	0.825	0.442
6	2	0.825	0.442

4. Analysis Validation and Performance

For validation, I tried to stick as much as possible to the method implemented in the tester.py script, that is, a k-stratified split cross-validation (based on the StratifiedShuffleSplit function from sklearn library). I just limited the number of iterations to 100 for performance reasons. In that way, the model built will almost match the tester.py score. By default, the StratifiedShuffleSplit function retains 90% of data to train the model and 0.1% to test it (and measure performance).

Depending on what we try to achieve, both the Gaussian Naïve Bayesian and Decision Tree could be selected. The earlier has a better accuracy, the latter a higher Precision and Recall. I understood that the goal of this project was to reach at least 0.3 Precision and Recall, hence optimizing the model accordingly. So, under these circumstances, the decision tree is the best one:

Accuracy: 0.805	**Precision**: 0.468	**Recall**: 0.553
	F1: 0.507	**F2**: 0.534
Total predictions: 11000	**True positives**: 1106	**False positives**: 1257
	False negatives: 894	**True negatives**: 7743

5. The Discussion and Conclusions

The precision can be interpreted as the percentage of people relevantly classified as POI. A 47% precision indicates that amongst all the poi predicted, only 47% were actually identified as poi, the rest (53% being misclassified). Recall measures the percentage of POI that were actually found. A 55% recall means that amongst all the actual poi, only 55% of them were correctly identified by our model. The 45% remaining were missed.

It is of course best to have Precision and Recall balanced (approximately equal) and closed to 1 (1 being a perfect model, hence suspicious). The K-nearest neighbor model is a good counter example: even though its accuracy was pretty good, it got seriously penalized by a lack of precision and recall. A deeper analysis revealed that it was predicting everyone as non-poi with not many errors, mainly because of the low number of actual poi in our dataset.

I can't really say the model I trained is very robust, as it misclassifies 53% of the people and misses 45% of them. 50% chances of error (F1 score) is obviously not enough. On another hand the number of poi in the dataset is way too low (18 over 145) to achieve higher performances. Besides the fact that almost half of them (35 initially) got discarded for various reasons. More data will definitely help to train/validate a better model. Paying attention to the content of the emails (with text analytics as taught in mini project) might also be something valuable to do.

References:

- Enron scandal
- Scikit-learn documentation
- SelectKBest function (sklearn)
- GridSearchCV function (sklearn)
- StratifiedShuffleSplit function (sklearn)
- Recall and Precision tradeoff