Enron POI Identifier Report

Martin Welss

Introduction

The Enron case was one of the biggest cases of company fraud in the history of the USA. In the last years before the collapse, the managers created more and more complex accounting strategies to pretend fabulous profits but in reality produced only losses. These strategies where backed by the accountants of Andersen Consulting which disintegrated in the follow up turmoil and in the course of the trial. There is an excellent book which describes the rise and fall of Enron in great detail: “The Smartest Guys in the Room” by Bethany McLean.

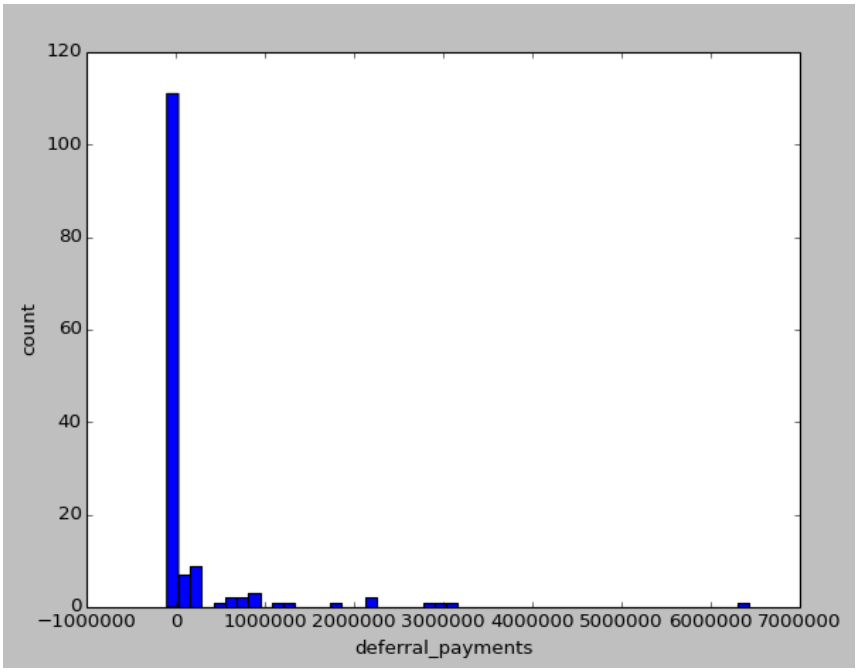
System Information

This analysis was executed on a Fedora Linux with Python 2.7

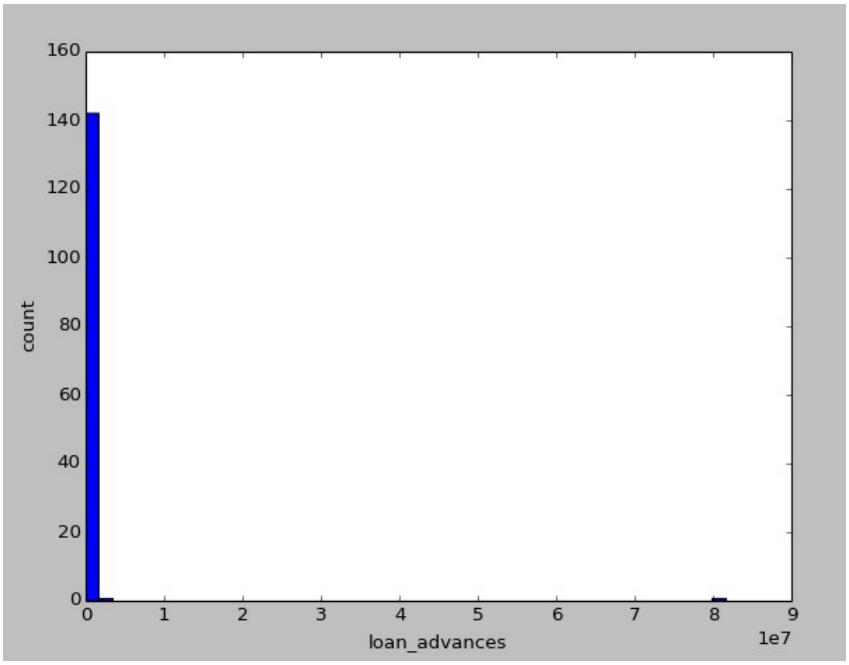
The Enron Dataset

The poi_names.txt file contains a list of POIs and preceding each POI is a flag, which signals if the inbox of that person is available. There are 35 POIs in this list and we have the inbox of only of four of them. In the actual dataset there are records of 146 persons that are potential POIs. The goal of the project is to write a machine learning program that identifies the POIs out of the complete dataset.

The first step is to scrutinize the dataset for irregularities and outliers. Two outliers can be found in the dataset: the first is a total from a spreadsheet export and the second a travlel agency. These outliers are removed from the dataset at the beginning of the program. Looking at the data manually by plotting the distributions, one can see that most of features have many missing values. I included two diagrams as examples:



The feature “deferral payments” has more than 100 missing data points and the following distribution of “loan advances” has more than 140 missing data points (that's nearly all!)



Feature Selection

Inspired by the analysis above and the fact of many missing values, I ranked the features by counting the NaN values and the “Hit count”. A data point is a “hit”, if its value is not NaN and the person is a POI. So the first line in the table means that for the feature “total_stock_value” there are only 20 NaNs and 18 POIs have a not NaN as value.

I ranked them from most promising (total_stock_value) to the least meaningful feature (restricted_stock_deferred).

Feature	NaN count	Hit count
total_stock_value	20	18
total_payments	21	18
expenses	51	18

other	53	18
restricted_stock	36	17
salary	51	17
bonus	64	16
to_messages	59	14
from_poi_to_this_person	59	14
from_messages	59	14
from_this_person_to_poi	59	14
shared_receipt_with_poi	59	14
exercised_stock_options	44	12
long_term_incentive	80	12
deferred_income	97	11
deferral_payments	107	5
loan_advances	142	1
director_fees	129	0
restricted_stock_deferred	128	0

Furthermore I created two more features:

- bonus_by_salary which is the bonus divided by salary which should give a good impression of the relation between bonus and salary for that particular person.
- email_fraction_from_poi which is the fraction of emails the person received that was sent by a POI. The idea is that POIs are likely to sent more messages to other POIs

I then used SelectKBest feature selection (code in select_kbest.py) that prints the top 10 features ranked by importance:

1. total_stock_value
2. bonus
3. salary
4. bonus_by_salary
5. restricted_stock
6. total_payments
7. expenses
8. email_fraction_from_poi
9. other
10. from_this_person_to_poi

The new features rank 4 and 8, so at least the new feature bonus_by_salary is good candidate for further use.

But using the features from SelectKBest did **not** give the mandatory results of greater values than 0.3 for precision and recall. I therefore wrote my own feature selector that uses naïve bays and tester.py directly. It can be found in the file mw_feature_selector.py.

I let it run to find the best combinations of 3 and 4 features respectively and ranked them by f1. Here are the top 20 for each (complete list is in best_features.txt):

F1	Precision	Recall	Acc.	Features: 3
0.532	0.363	1.000	0.707	['restricted_stock_deferred', 'deferred_income', 'director_fees']
0.482	0.568	0.418	0.872	['deferred_income', 'exercised_stock_options', 'from_messages']
0.464	0.609	0.374	0.856	['bonus', 'deferred_income', 'from_messages']
0.452	0.543	0.387	0.866	['deferred_income', 'total_stock_value', 'from_messages']
0.451	0.587	0.366	0.838	['bonus', 'deferred_income', 'long_term_incentive']
0.447	0.573	0.366	0.871	['deferred_income', 'total_stock_value', 'restricted_stock']
0.446	0.540	0.381	0.865	['deferred_income', 'total_stock_value', 'long_term_incentive']
0.446	0.533	0.384	0.864	['deferred_income', 'total_stock_value', 'exercised_stock_options']
0.445	0.635	0.342	0.858	['to_messages', 'bonus', 'deferred_income']
0.444	0.550	0.372	0.867	['deferred_income', 'exercised_stock_options', 'restricted_stock']
0.443	0.562	0.366	0.869	['deferred_income', 'expenses', 'exercised_stock_options']
0.441	0.579	0.355	0.871	['salary', 'deferred_income', 'exercised_stock_options']
0.434	0.535	0.364	0.864	['deferred_income', 'total_stock_value', 'expenses']
0.431	0.521	0.368	0.861	['deferred_income', 'exercised_stock_options', 'shared_receipt_with_poi']
0.431	0.538	0.359	0.854	['deferred_income', 'exercised_stock_options', 'long_term_incentive']
0.430	0.587	0.339	0.850	['bonus', 'deferred_income', 'from_poi_to_this_person']
0.429	0.606	0.333	0.853	['salary', 'deferred_income', 'expenses']
0.426	0.576	0.338	0.834	['salary', 'deferred_income', 'long_term_incentive']
0.424	0.510	0.363	0.859	['deferred_income', 'exercised_stock_options', 'email_fraction_from_poi']
0.424	0.550	0.344	0.856	['salary', 'deferred_income', 'from_messages']

F1	Precision	Recall	Acc.	Features: 4
0.494	0.339	0.913	0.689	['loan_advances', 'restricted_stock_deferred', 'deferred_income', 'director_fees']
0.473	0.578	0.400	0.873	['deferred_income', 'total_stock_value', 'expenses', 'from_messages']
0.472	0.579	0.399	0.873	['deferred_income', 'total_stock_value', 'expenses', 'restricted_stock']
0.470	0.574	0.399	0.872	['deferred_income', 'expenses', 'exercised_stock_options', 'restricted_stock']
0.469	0.564	0.401	0.870	['salary', 'deferred_income', 'exercised_stock_options', 'long_term_incentive']
0.465	0.565	0.396	0.870	['deferred_income', 'expenses', 'exercised_stock_options', 'from_messages']
0.462	0.556	0.395	0.869	['salary', 'deferred_income', 'total_stock_value', 'from_messages']
0.458	0.554	0.391	0.868	['salary', 'deferred_income', 'exercised_stock_options', 'from_messages']
0.457	0.549	0.392	0.867	['deferred_income', 'total_stock_value', 'from_messages', 'restricted_stock']
0.455	0.545	0.391	0.866	['deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options']
0.453	0.544	0.389	0.866	['deferred_income', 'expenses', 'exercised_stock_options', 'long_term_incentive']
0.453	0.530	0.396	0.864	['deferred_income', 'total_stock_value', 'expenses', 'long_term_incentive']
0.452	0.550	0.384	0.867	['deferred_income', 'total_stock_value', 'exercised_stock_options', 'restricted_stock']
0.452	0.513	0.404	0.860	['deferred_income', 'total_stock_value', 'from_messages', 'long_term_incentive']
0.450	0.534	0.389	0.864	['deferral_payments', 'deferred_income', 'exercised_stock_options', 'from_messages']
0.449	0.525	0.392	0.863	['deferred_income', 'exercised_stock_options', 'from_messages', 'restricted_stock']
0.449	0.550	0.379	0.857	['to_messages', 'bonus', 'deferred_income', 'long_term_incentive']
0.447	0.499	0.404	0.857	['deferred_income', 'total_stock_value', 'exercised_stock_options', 'long_term_incentive']
0.445	0.524	0.387	0.862	['salary', 'deferred_income', 'total_stock_value', 'exercised_stock_options']
0.444	0.513	0.392	0.860	['deferred_income', 'exercised_stock_options', 'from_messages', 'long_term_incentive']

Pick and Tune Algorithm

Parameter Tuning

All classifiers except GaussianNB have many parameters that influence their performance. That is why parameter tuning play an essential part in practical machine learing. It is often a good idea to let the computer do the timely task of tuning the parameters by using GridSearchCV.

Performance Metrics

To measure the performance of an algorithm one can apply different metrics. In this project the focus is on precision and recall. Precision measures how many of the *returned* positives were *true* positives and recall measures how many of the true positives were actually returned. In both cases a higher value (closer to one) is better.

Validation

Validation ensures that the data is evenly distributed over the test and training sets. The code in tester.py uses StratifiedShuffleSplit cross validation with 1000 folds which I find to be very appropriate for this task.

DecisionTree

My first shot was the DecisionTreeClassifier which I tuned with GridSearchCV. The correnponding code can be found in poi_id_tree.py. A decision tree does not need feature scaling. I tried several combinations and finally went with this parameter grid:

```
param_grid = {
   'min_samples_split': [2, 3, 4],
   'max_depth': [3,4,5],
   'criterion': ['gini', 'entropy'],
   'splitter': ['best', 'random']
}
```

Here are the results:

'total_stock_value', 'salary', 'bonus','bonus_by_salary':
Accuracy: 0.83862 Precision: 0.40041 Recall: 0.09850

'total_stock_value', 'restricted_stock', 'salary', 'bonus','bonus_by_salary':
Accuracy: 0.84879 Precision: 0.37255 Recall: 0.08550

'total_stock_value','restricted_stock','salary','bonus','bonus_by_salary','total_payments':
Accuracy: 0.85433 Precision: 0.31313 Recall: 0.07750

The problem is the low value for recall that does not pass the project rubric. A low recall often is a sign for overfitting.

SVM

I then switched to svm and the correpondimg code can be found in the file poi_id_svm.py. Svm has many parameters to tune and needs feature scaling. I tried three different kernels: linear, rbf and poly and applied GridSearchCV to optimize the parameters, but I ran into technical problems in sklearn itself (segmentation fault), so I abandoned svm.

GaussianNB

I finally went with GaussianNB which has two attractive properties: it does not need feature scaling and has no parameters to tune. So it is all about feature selection. The corresponding code can be found in the file poi_id.py

Looking at the tables in the previous chapter about feature selection shows that there are many feature combinations that match the mandatory values for precision and recall. I found it interesting to see that between 3 and 4 features there is no big difference and that the top combinations of 3 features are slightly better than the top combinations of 4 features.

Regarding own newly created features, the feature email_fraction_from_poi seems to by more valuable because it shows more often in the top feature combinations than bonus_by_salary.

Discussion and Conclusion

The decided to go with GaussianNB: that gives on my computer a precision of 0.57 and a recall of 0.42. This is not really satisfying because it means there are more misses than hits. I was really wondering why DecisionTree did not give better results, since theoretically it should be well suited for a binary outcome: poi or not poi. I think one of the main problems with this dataset are the many missing values. Maybe it would be one way to improve the results to go back to the data collection and try to complete the data for the features.