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.

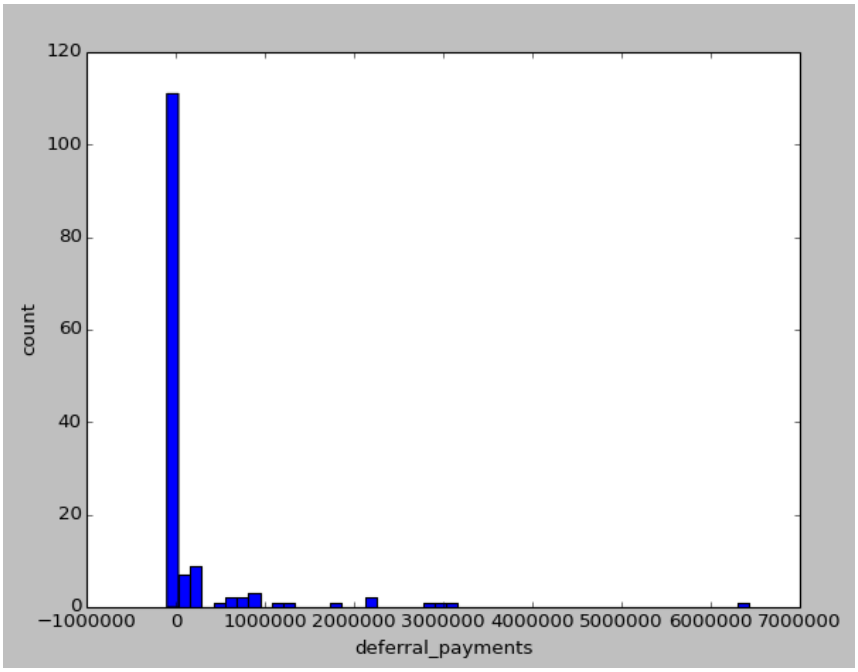
System Information

This analysis was executed on a Fedora Linux with Python 2.7

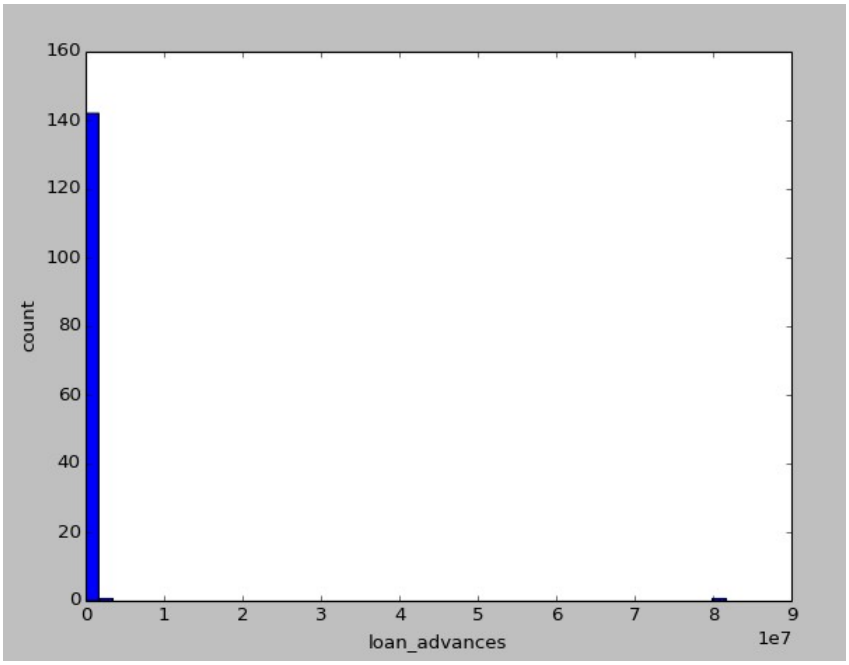
The Enron Dataset

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. One outlier can be found in the dataset which is a list of totals from a spreadsheet export. This outlier is 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:

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 leat

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.

GaussianNB

The first classifier that I tried was a GaussianNB. GaussianNB has two attractive properties: it does not need feature scaling and has no parameters to tune. The corresponding code can be found in the file poi_id_nb.py

Features	Precision	Recall	Run
'salary', 'bonus', 'total_stock_value', 'exercised_stock_options'	0.38902	0.25150	1
'salary', 'bonus'	0.29543	0.12600	2
'poi','salary', 'bonus','bonus_by_salary'	0.34045	0.15950	3
'poi','salary', 'bonus','email_fraction_from_poi'	0.27056	0.12500	4

I first tried several variations of the given features and came up with run 1. To test my created features I first made a reference run 2 with just salary and bonus and then added each new features in a subsequent run. bonus_by_salary increases the precision and the recall but email_fraction_from_poi decreases both.

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.

DecisionTree

I finally went for a DecisionTreeClassifier which I tuned with GridSearchCV. 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

Discussion and Conclusion

The best shot was DecisionTree with 4 features which gives a precision of 4%. This is not really satisfying because it means there are more misses than hits. I was really wondering why Naive Bayes often gave better results than the fine tuned DecisionTreeClassifier. 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.