Identifying Fraud at Enron

Author: Stas Sajin

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? [relevant rubric items: "data exploration", "outlier investigation"]

Background Story

In 2001 Enron Corporation faced one of the greatest corporate governance scandals. The management of the company was misinterpreting the value of the company through unethical accounting practices, such as aggressive revenue recognition and offloading of liabilities through special purpose entities. Moreover, the company had a harmful corporate culture, which put a constant emphasis on stock price. For instance, most of the management compensation comes from stock options, which gives management an incentive to game the stock price in the short term at the expense of future profitability of the company.

Objectives

The following report implements several machine learning algorithms that try to identify persons of interest (POI) among Enron executives. A POI is someone who had a direct involvement in accounting and financial machinations surrounding Enron's case.

Dataset

We have two datasets. The first datasets contains information on executive compensation. The second dataset is a very large corpus of emails from Enron employees. The corpus, which comprises over 600,000 emails was acquired by the Federal Energy Regulatory Commission. It is important to note that this corpus is very unique, since no public or private company would allow for its emails to be part of the public because they might contain highly sensitive information. Because of the different nature of the two datasets, we essentially have to types of features: email features and financial features.

Although we have a very large corpus of emails available, I will generally try to refrain from using email features. I will do this for a couple of reasons:

1) Discarding email features and only focusing on financial features allows one to have better generalizability when trying to identify if a company has a harmful corporate culture. Financial data on executive compensation is made public to shareholders for every publically traded company, while email data is almost never available. Hence, if someone asks a data expert to identify the features that are common for corporate insiders that engage in fraudulent behavior, they can generally only use financial data to make their predictions.

2) Some email features, such as 'from_this_person_to_poi, from_poi_to_this_person, shared_receipt_with_poi have data leakage, since the creation of these features was not validated through tests and training sets. See discussion here.

Outliers

The financial data contains several problematic rows that were removed:

* The row "TOTAL" was removed because it identifies the sum for each feature rather than the actual individuals in the dataset.

*The row "THE TRAVEL AGENCY IN THE PARK" was removed because it does not correspond to an actual person

*The financial information for 'BELFER ROBERT' and 'BHATNAGAR SANJAY' was corrected (see discussion here)

Other than the data for the rows above, none of the data looked out of the ordinary. There were several individuals with excessive levels of compensation for some financial features, but their data was kept as it is.

Descriptive information about the dataset

A lot of financial information in the dataset was identified as having missing data. Nonetheless, this interpretation of missingness is incorrect, since all the NaN values for financial data should've been substituted with 0. In other words, the data was not missing in the sense that we don't know what the values for those cells. Hence, all financial features in the dataset were replaced with 0.

The email features also had some missing values. I refrained from replacing those rows with 0 or removing them. Instead, I replaced those rows with the median of each email feature broken down by POI.

These are some characteristics about the dataset:

Number of persons: 144

Number of POI: 18

Number of features: 21

Proportion of POI: .125

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. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

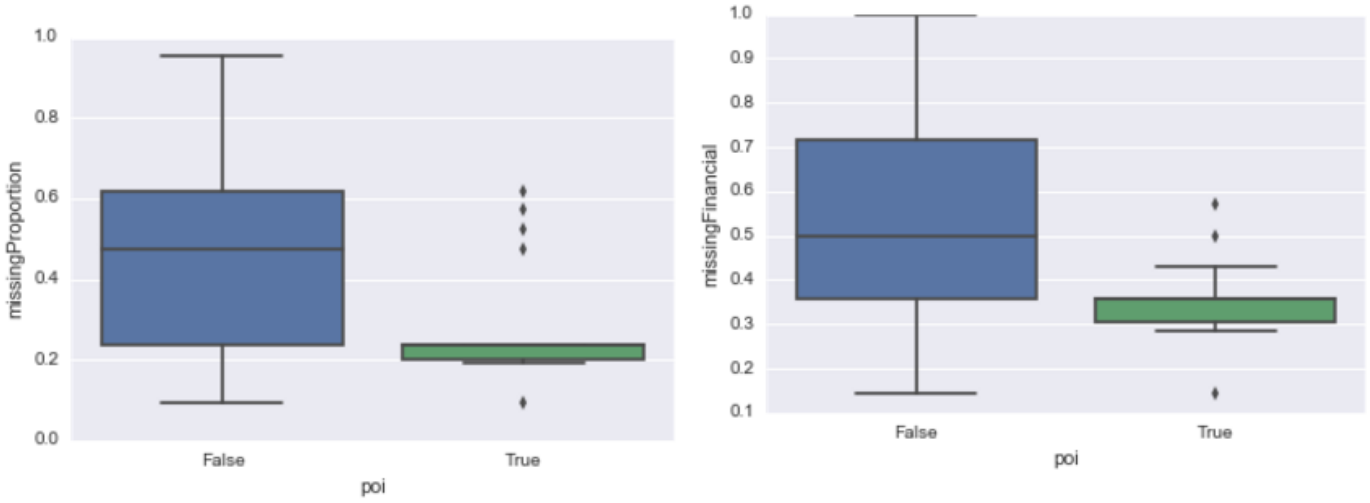
Before performing any feature selection, I tried to first create a few more features that would hopefully be useful during prediction. I created these features before I ran any sort of models.

* **Feature 1:** I noticed that in the original dataset a lot of individuals had missing data. For instance, "LOCKHART EUGENE E" had missing information on 95.24% of all 21 original features. On the other hand, LAY KENNETH L had almost 90% of the data present. I realized that it is very likely that individuals who are heavily involved within the organization, are also the ones who have less missing data available. So I created two new features.

-MissingProportion: this counts the proportion of missing values among all 21 original features.

-FinancialMissing: this feature counts the proportion of zeros (i.e., NaN) in financial features for the original set of financial features.

The point of these features is that a person who is heavily involved within the organization will likely have a lot of clout in getting the board to reward them multiple types of compensation. I tried to assess the quality of these features by looking at how they relate to the POI class. The left plot shows the pattern of missingness for all features, broken down by POI class. The right plot shows the proportion missing for financial features, broken down by POI class. You can note that POI have less missing data and more information on financial data overall when compared to non-POI.



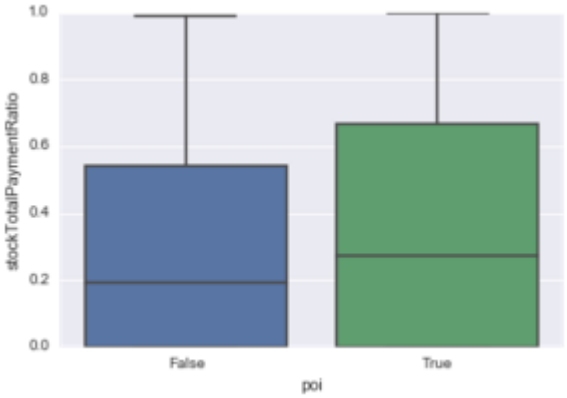
salary	50
to_messages	58
deferral_payments	106
total_payments	21
exercised_stock_options	44
bonus	63
restricted_stock	34
shared_receipt_with_poi	58
restricted_stock_deferred	127
total_stock_value	19
expenses	49
loan_advances	141
from_messages	58
other	54
from_this_person_to_poi	58
poi	0
director_fees	129
deferred_income	95
long_term_incentive	79
email_address	33
from_poi_to_this_person	58

* **Feature 2:** Another thing I noticed about the dataset was the pattern of missing data for email_address feature. There were a total of 33 emails present, which I thought was unusual. One thing I considered is that during investigation, they kept the emails of the people who were most likely under suspicion of fraud, and that the emails of other individuals were not kept on record. I decided to create another feature called "email_present", which had a binary value of 1 or 0, indicating if there was an email address associated with a person or not. I found this feature to be very interesting, since it indicated that among people who were POI, all of them had an email address. In other words, out of 33 individual with an email address, 18 of them were POI.

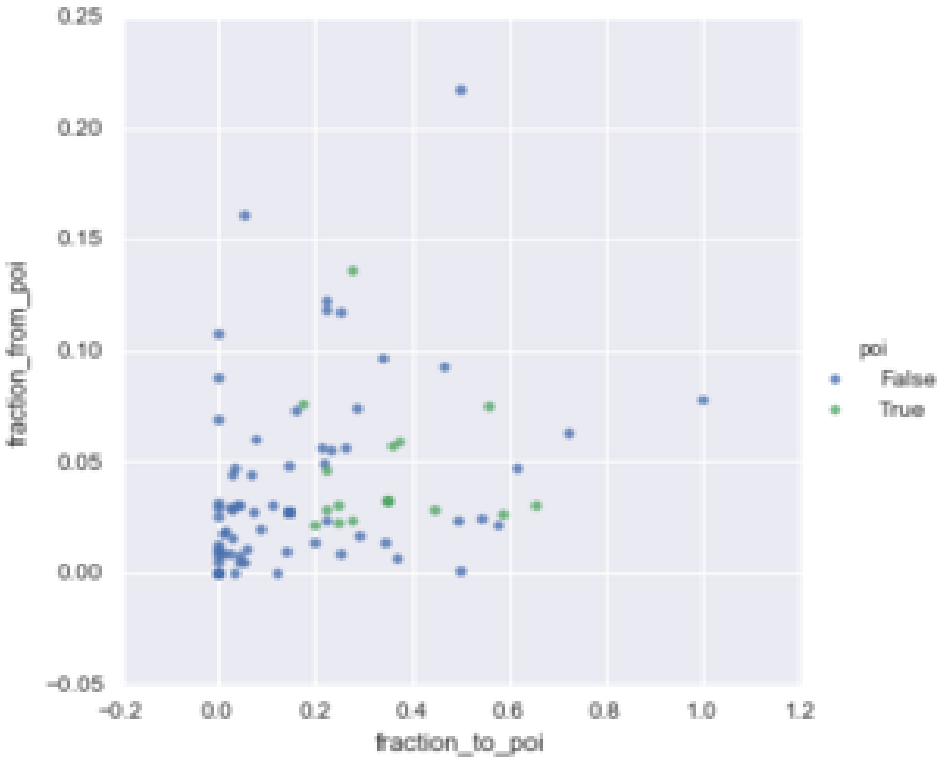
In Bayesian terms, if the only information I give you is that the person has an email address, then you can estimate the probability that the person is a POI at 54.5%.

Note that I'll not be using "email_present" feature in the final model (see my answers to question 1 for why I don't use email features in final model evaluation).

*__Feature 3:__ The third feature I made looked at the ratio between total stock compensation over total compensation: (exercised_stock_options)/(total_payments+exercised_stock_options).I assume that insiders were more likely to exercise their stock options before the company stock would devalue in price, so this ratio would be higher for insiders who had an involvement in financial/accounting fraud. You can see below that the median is higher for POI true for this ratio.



*__Feature 4 & 5__: These were the features identified in class. These features measured the interconnectedness between poi by looking at the ratios for from_this_person_to_poi/from_messages and from_poi_to_this_person/to_messages. In other words, it showcases that fraudsters are more likely to communicate more often among each other.



After feature creation process was completed, I ended up with the following list of features (the list contains both financial and non-financial features).

features=['salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercised_stock_options', 'bonus', 'restricted_stock',
'shared_receipt_with_poi', 'restricted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances',
'from_messages', 'other', 'from_this_person_to_poi', 'director_fees', 'deferred_income', 'long_term_incentive',
'from_poi_to_this_person', 'missingProportion', 'missingFinancial', 'email_present', 'stockTotalPaymentRatio',
'fraction_to_poi',
'fraction_from_poi']

I ended up creating two sets of models. In one set, I used all the features outlined above. In the second set, I used only financial features. All features have been scaled using MinMaxScaler() because descriprive statistics for each feature indicated that the numerical features had a non-normal distribution.

The final model that is submitted to tester.py used the following features:

features_list= ['poi','salary', 'deferral_payments', 'total_payments', 'exercised_stock_options', 'bonus', 'restricted_stock', 'restricted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances',
'other', 'director_fees', 'deferred_income', 'long_term_incentive', 'missingFinancial', 'stockTotalPaymentRatio']

I did not use SelectKBest when tuning the parameters of the final model because it would give me worse performance when compared to using all features. The feature importance for the final AdaBoost Model are as follows:

	feature	importance
0	other	0.24
1	salary	0.18
2	exercised_stock_options	0.12
3	expenses	0.12
4	total_stock_value	0.10
5	total_payments	0.06
6	restricted_stock	0.06
7	stockTotalPaymentRatio	0.06
8	deferral_payments	0.02
9	bonus	0.02
10	deferred_income	0.02

3) What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]

Below, you will find a table with the algorithms that have been used, the performance (precision, recall, and F1) for each algorithm, the run times. Performance is broken down on whether I used all the features in the feature list (email+financial features) or only financial features. The performance metrics are for 100 stratified randomized folds using the StratifiedShuffleSplit from sklearn. All models were run with default parameters. Note that the green cells represent the cases where precision and recall was above .3.

	All features				Financial Features Only			
Model	**Precision**	**Recall**	**F1**	**Time (s)**	**Precision**	**Recall**	**F1**	**Time (s)**
ExtraTreesClassifier	0.465	0.32	0.364	11.17	0.308	0.21	0.239	12.074
RandomForestClassifier	0.501	0.36	0.403	13.0	0.265	0.17	0.20	12.83
GaussianNB	0.320	0.425	0.332	1.458	0.393	0.47	0.392	1.435
DecisionTreeClassifier	0.708	0.655	0.647	1.803	0.239	0.275	0.240	1.901
ExtraTreeClassifier	0.446	0.4	0.395	1.61	0.246	0.27	0.239	1.717
AdaBoostClassifier	0.502	0.455	0.453	53.69	0.418	0.325	0.348	52.55
GradientBoostingClassifier	0.59	0.525	0.527	36.72	0.319	0.24	0.26	33.77
KNeighborsClassifier	0.265	0.16	0.195	2.02	0.265	0.16	0.195	2.086

There are a couple of things to note about the algorithms above. When using all features, for almost all the models, except for the K-NN classifier, the performance on precision and recall was above .3 threshold. Decision Tress seems to beat every model when all features are used. On the other hand, using only financial features, there end up only two models that have precision and recall above .3. In the next section, I will tune the AdaBoost classifier algorithm and see if I can squeeze any more performance improvements through feature selection, scaling, and parameter tuning.

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). [relevant rubric item: "tune the algorithm"]

Parameter tuning essentially involves trying out different parameter values that would hopefully improve model performance during cross-validation. The default parameter setting in sklearn is a general purpose setting and might not provide the best performance. Parameter tuning is an important aspect of model creation process. For instance, if a dataset has many predictors, ensemble methods like Random Forests can *generally* benefit from larger number of estimators and greater tree depth, and the precise parameter for number of trees and tree depth can be identified through parameter tuning. Generally, one should try a variety of tuning parameters. It is very important to tune the parameters on the train set only, otherwise we would inadvertently lead to overfitting if we tuned on the whole dataset.

I tuned the parameters for the AdaBoost model using a parameter grid. When tuning the parameters for this algorithm, I used the helpful suggestions in this post. The parameters were tuned in a pipeline using GridSearcCV with the following sequence: 1) scale the features using MinMaxScaler, 2) run the model for each parameter grid, 3) cross-validate the model using 20 stratified randomly shuffled splits 4) select the model that maximizes F1. I did not use any feature selection algorithm because they led to lower performance during cross-validation.
The final model selection converged to and AdaBoostClassifier eith a SAMME.R algorithm, learning rate of 1, and 50 n_estimators.
The parameter grid used was as follows:

```
param_grid = {
              'model__n_estimators':[25, 50, 100, 200, 400],
              'model__learning_rate':[.5, 1, 1.5],
              'model__algorithm':['SAMME', 'SAMME.R'],
              'model__random_state':[42],

             }
```

5) What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]

Validation is a crucial step in testing machine learning performance. The purpose of validation is to provide us a way to find the sweet spot in the bias-variance tradeoff, thus increasing generalization performance of the model. One mistake that is often made in validation (see also discussion from Hastie and Tibshirani) is that validation needs to be performed on feature selection, parameter selection, and feature scaling. For instance, one should not perform the PCA on the whole dataset and then do feature selection. Similarly, one should not perform feature scaling on all the data and then do validation. It is important that scaling, feature selection, and parameter selection be performed for train data first and then applied to the test data.
The final model used the stratified shuffle split approach. The benefit of this approach is that it not only splits randomly the train and tests sets, but also ensures that the proportion of true or false labels for POI is about the same for the train and test sets. I used a total of 20 splits when validating the data for parameter selection. The tester.py uses 1000 splits to validate the final model.

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. [relevant rubric item: "usage of evaluation metrics"]

The final performance for the classifier was
Precision: 0.43378
Recall: 0.339
F1: 0.38058
The precision (aka positive predictive value) measures the proportion of POI the model has identified as true POI. In other words, among all the POI that the model has identified, only 43% were true POI, with the rest being false alarms. The model essentially falsely accuses 57% of the people as POI.

Recall (aka true positive rate or sensitivity) measures the proportion of people that were identified as POI among all the POI that are in the test set. Lower recall values means that the model makes a lot of misses. A value of .339 for recall means that about 64% of the POI were not identified by the model.

References

http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
https://discussions.udacity.com/t/gridsearchcv-is-making-my-decision-tree-perform-worse/160739/2
http://pandas.pydata.org/
https://stanford.edu/~mwaskom/software/seaborn/
https://en.wikipedia.org/wiki/Precision_and_recall
http://stackoverflow.com/questions/28822756/getting-model-attributes-from-scikit-learn-pipeline