Identifying fraud from Enron Email

Report by Thomas Draebing

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"]**

Enron was one of the biggest energy companies of the world with revenues of 111 Billion dollars in 2000. In 2001 a willful fraud conspiracy in the company was revealed, which lead to the downfall and bankruptcy of the company. This case of major corruption was one of the biggest of its kind. It is especially interesting for data analysts since the financial data and company internal communication was made public. In this project for the Udacity Data Analyst Nanodegree the aim is to use machine learning techniques to investigate whether the persons of interest in this fraud case can be predicted using the available data sets. In future such a model could be used to get an estimate whether a person of interest given its financial and communication data might be a suspect in a fraud case. Thus the whole circle of suspects, which in big companies like Enron can be huge, could be prescreened and the priorities in an investigation could be adjusted accordingly.

The data has 21 features comprising financial data like the salary, bonus or stock held by the respective person and also data about the email communication like how many messages were send by a person and how many were send to POIs. The data set contains 145 entries in total of which only 18 are POIs, thus is strongly unbalanced. The data contains quite a lot NAs, up to 90% per feature, which generally is bad for modeling. The feature email_address was removed, since it identifies every entry with a unique value, which would lead to overfitting. The entry TOTAL, which summarizes the financial data of all entries was removed as well.

The data needed some preprocessing before starting to use models to further investigate the data. First the data set was divided into a train- and test set to allow for proper validation in later models and to simulate preprocessing of unknown data.

At first outliers were removed. To this end outliers were defined by a greater distance than three standard deviations from the mean for the respective feature. Since a quick analysis of the mean difference in the features between POIs and non-POIs showed that in most variables POIs on average get higher values and since the aim of fraud is to increase one's financial value outliers in the financial data could be good identifiers for POIs and were left in the data set. For the communication features outliers were removed, since a lot of communication would be not very indicative for a crime and could be rather job related. This features include from_messages, from_poi_to_this_person, from_this_person_to_poi, shared_receipt_with_poi, to_messages.

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"]**

The data for most features is skewed, defined here as a skewness coefficient bigger or smaller than 1 or -1 respectively. To deal with the skewness the data was transformed using the decadic logarithm. Since the variables 'deferred_income', 'from_poi_to_this_person', 'from_this_person_to_poi', 'shared_receipt_with_poi contain zeros, 1 was added to all values of the respective features to allow for taking the logarithm. Some features did not show an improved skewness coefficient after transformation or are binary variables and were excluded from transformation. The transformed features include: 'bonus', 'deferral_payments', 'deferred_income', 'exercised_stock_options', 'from_messages', 'from_poi_to_this_person', 'from_this_person_to_poi', 'long_term_incentive', 'other', 'restricted_stock', 'to_messages', 'total_payments', 'total_stock_value'.

The scales of the different features are quite different, especially because enot all features were transformed. This could be a problem, when trying to determine the importance of data, since features having large values like salary might overshadow features like 'other'. Thus MinMaxScaling was applied.

The features were analyzed using several different techniques to measure their importance:

i) The correlation coefficient matrix revealed high correlations (> 0.7) between some variables:

- total_stock_value : exercised_stock_options (0.83)
- to_messages : from_poi_to_this_person (0.80)
- salary : bonus (0.78)
- shared_receipt_with_poi : from_poi_to_person (0.76)
- to_messages : shared_receipt_with_poi (0.75)
- from_this_person_to_poi : from_poi_to_this_person (0.74)
- total_stock_value : restricted_stock (0.71)

ii) A lasso model was trained, but was unable to model the data, resulting in all coefficients being 0. Thus this method was dropped.

iii) A decision tree was used to calculate the feature importances. Some of them were 0 and were thus considered to be of little importance for the model:

- deferral_payments
- deferred_income
- director_fees_bin
- from_messages
- from_poi_to_this_person
- loan_advances_bin

- restricted_stock_deferred_bin
- salary
- shared_receipt_with_poi
- to_messages

iv) A k-best selector was trained and the scores extracted. The following features had scores in the lowest quartile:

- deferral_payments (0.012)
- from_messages (0.410)
- exercised_stock_options (0.781)
- loan_advances_bin (0.972)

v) Further evaluation scores from the final model were observed, when adding or removing features. Features largely improving the model performance were kept in the data set. The final data set included the following features:

- bonus
- deferred_income
- exercised_stock_options
- expenses
- long_term_incentive
- other
- restricted_stock
- salary
- restricted_stock_deferred_bin
- shared_receipt_with_poi
- to_messages

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"]**

The algorithm used for the final model is AdaBoost. Further Gaussian Naïve Bayes, Random Forest and Support Vector Machines were tested. The accuracy of Gaussian Naïve Bayes is only about half of the other algorithms. Looking at the predictions, the problem with using accuracy as an evaluation metric becomes obvious. SVM for example predicted every entry to be negative. Since the data is very unbalanced, this leads to high accuracy scores. Thus other metrics like Precision and Recall were also calculated.

The random forest model shows the highest accuracy and precision. The best recall, F1- and F2-score was observed for Gaussian Naïve Bayes. The metrics of the Adaboost model were in between those of the other classifiers. Nevertheless I chose AdaBoost since the other models seem unbalanced. Gaussian Naïve Bayes has a very high number of false positives, which considering the presumption of innocence might be problematic. The problem with random forest on the contrary is that it classifies too many positives as negatives.

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"]**

Nearly every machine learning algorithm possesses parameters that can be changed to adjust the algorithm to the data set at hand. This could be parameters that determine, how conservative an algorithm is or if the algorithm predicts well, but runs too slow for your application, it can be tuned to be faster e.g. by reducing the number of estimators. Not tuning the parameters can lead to unnecessarily high computational costs, overfitting and overall underperforming models.

To tune the AdaBoost algorithm for the Enron dataset, the evaluation scores were observed upon changing the following parameters:

- n_estimators
- learning_rate
- max_depth of the DecisionTree base_estimator
- min_samples_leaf of the DecisionTree base_estimator

The results were plotted (refer to the ipython notebook) and the parameters giving the best overall scores were chosen. Interestingly for this particular task except for n_estimators all the default parameters gave the best results. The parameters n_estimators and learning_rate were set to 100 and 1 respectively. Since they possess a tradeoff, adjusting one of them would be enough. The min_samples_leaf parameter determines how many samples a leaf (the final node on the tree) has to have. Small values lead to models that are prone to overfitting, since for a value of 1 every sample has its own leaf and thus noise may be captured. Thus higher values are most of the time preferable. For this dataset a value of 1 worked best overall, maximizing the recall and f-scores. This is probably due to the small sample size and even smaller amount of POIs. The scores stabilize at a parameter value of 10. Thus for bigger sample sizes, this would probably be the value to choose. Max_depth determines how much splits the algorithm is allowed to perform. The best results were achieved by not restricting the tree depth. To small values result in good accuracy, but bad recall, since a lot of false negatives are predicted. This gets better with rising values for max_depth.

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"]**

While models might represent a giving dataset, which it is trained on, very well, they might perform poorly on new data. This is called overfitting. What happens is that the model does not generalize very well, but also fits noise. To detect overfitting, the original dataset is divided into at least two partitions, the training set and test set. The model is then trained on the training set. The test set is used to evaluate the model's prediction capability.

In the case of the Enron dataset a single training- and test set split was performed to validate model-based preprocessing steps. The full model was then validated using the cross-validation function StratifiedShuffleSplit from sklearn. Using just a single split for the whole validation process might lead to problems, since, especially in a small, unbalanced dataset as used here, the distribution of labels might be uneven or the chosen training set shows a pattern not present in the test set. This are problems cross-validation can solve. There are several methods available. The stratified shuffle split method produces a given amount of different training- and test sets, by creating partitions containing a different composition of samples from the same original dataset. This method shuffles the data beforehand to avoid a bias caused by ordered data (e.g. all POIs clustered together) and preserves the overall target label percentage between training- and test sets. By training and testing the model on a big number of these training- and test sets outliers in data structure can be averaged out.

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"]**

Models per definition are not exact representations of reality, but try to predict or depict it as close as possible. From that follows, that there is a certain error that has to be considered. There are several metrics to measure the model's performance. Those validation scores weight the different error types differently. The metrics used in this project are the following

a) Accuracy:

$$Accuracy = \frac{True\ Negatives + True\ Positives}{True\ Negatives + False\ Negatives + True\ Positives + False\ Negatives}$$

The Accuracy is problematic in the case of the Enron dataset, since the data is very unbalanced. This causes the phenomenon called accuracy paradox. Just predicting only non-POIs results in about 88 % correct predictions (accuracy = 0.88), since there are only roughly 12 % POIs. An accuracy score of 0.88 would normally considered to be very good, but in this case the model is actually performing very poorly. This happened for the SVM algorithm as mentioned above.

b) Precision:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

The precision score takes only positive predicitions into account. That is useful, when as in the case of the Enron dataset the data is unbalanced. Thus the paradox as described for accuracy does not happen for the precision score, but a very conservative algorithm could still get a high precision score, while performing poorly, as it is the case for a model that predicts a few true positives and no false positives, but a lot of false negatives.

c) Recall:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The recall represents the fraction of true positive predictions of all samples that are positive. Thereby it counters the problem of the precision score, but ignores false positives. For this reason precision and recall are often used together.

d) F1-score:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

The F1- score combines precision and recall and thereby results in a score that is not as strongly influenced by unbalanced predictions, but is also less easy to interpret and the given information cannot be transferred to the model as for the single values.

e) F2-score:

$$F2 = 5 \cdot \frac{Precision \cdot Recall}{4 \cdot Precision + Recall}$$

The F2-score works like the F1-score, but weighs precision and recall differently. This puts a higher weight to detect all true positives at the cost of false positives.
The aim of this project was to build a prediction algorithm to predict persons involved in fraud. Picking the best evaluation score should take into account the presumption of innocence. Thus a score punishing false positives like F0.5 or F1 would be a good choice, but on the other hand it might also be helpful to look at the F2-score and maximize it to capture as much true positives as possible, but keeping in mind that there is a high probability of innocent people being falsely accused. Since a model should never be used as a definitive prove, a model with maximized F2-score could still narrow down the number of persons to investigate further. Using another model with a maximized F0.5-score might allow to differentiate between high interest POIs and a wider field of probable POIs as predicted by the F2-maximizing model, thus helping to prioritize the investigations.

The scores the AdaBoost model reached were:

- Accuracy: **0.8298**
- Precision: **0.3454**
- Recall: **0.309**
- F1: **0.3262**
- F2: **0.3156**

Using the provided tester function results in slightly higher scores. That happens because the scaling and transformation are done on the whole dataset and not on the divided dataset.