Enron-Predicting POIs

Thomas Roscher

25 September 2017

Introduction

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. In this project I use Machine Learning algorithms to build a person of interest (POI) identifier based on financial and email data made public as a result of the Enron scandal.

Data Exploration

The dataset contains 14 financial features and 6 email features of 146 employees. Because it is always worthwhile to take a good hard look at ones data to get acquainted with its quirks and properties, the first step is uni- and bivariate data exploration. Basically, the most important thing here is to check distributions and to look for any irregularities and patterns such as outliers, coding errors, or NAs. Note, that plots are not supposed to look "dashboard good" for this purpose. In order to properly investigate the data before modeling I therefore wrote functions that:

1. Check basic summary statistic
2. Check number of missing values
3. Check potential outliers
4. Plot all univariate distributions
5. Plot all bivariate distributions between features and label
6. Plot a heatmap to see if there is multicollinearity

It turned out that there are plenty of missing values, though arguably most of them were not missing but rather zero which honestly speaking is a horrible coding approach. For example, it is not clear if a missing value for the feature "bonus" means that the person did not get any bonus payments at all, or if this information was simply not provided in the documents. For this project I assumed the former and thus transformed all NAs to zero. I know that this approach is far from optimal but may be justified by the the fact that more often then not NA actually is supposed to be zero and that moreover imputation is not any good here due to the small sample size and the high NAs ratio for many variables.

Moreover, there were two coding errors, namely that the sum of each feature was mistakenly stored as "TOTAL" which caused a massive outliers for most features and that there was an observation which was clearly not an employee. Other extreme values encountered were basically just the top earners of the company incuding the CEO Kenneth Lay who altogether got 153 Mio. Dollars or John Lavorato who got 8 Mio Dollars just in bonus payments. Just removing such cases is arguably not the best option esspecially if one one has so few observations. However, one still should keep in mind that some models are more sensitive to such outliers than others. For instance, AdaBoost might treat those outliers as "hard" cases and put tremendous weights on outliers while decision tree might simply count each outlier as one false classification. Thus, standardizing or scaling might be appropiate for some algorithms.

Besides, data exploration revealed that many variables are highly skewed and that for some of the features there is no variation for cases were "POI"" is true. Those variables ("director fees", "load advances", "restricted sock defered") were dropped because the arguably do not help to distinguish between "POIs" and "no POIS". on the other hand, "bonus","expenses", "from poi to this person", and "shared receipt with poi" seem to be promising candidates as indicated by violin plots.

Feature Engenering

Next, I performed some feature engineering which attempts to increase the predictive power of learning algorithms by creating features from raw data that help facilitate the learning process. The basic idea is to combine, group, disaggregate and transform given features to new ones. Honestly, speaking I found feature engineering a bit rough in this case since new features aren't self-evident. My first plan was to create a feature called total wealth which would be the sum of total payments and total stock value. I then would use total wealth to create relative variants of all the financial features. I also would create relative values for the two variables that indicate email transfer between POIs and non POIs because here a ratio is much more precise then an absolute value. Unfortunately, it turned out that this approach is especially for tree based methods complete rubbish. Generally, most of the new features were not very influential or complete useless.

Thus I followed a much less stringent approach of much try and error. I ended up using the following additional features:

1. NEW_rel_poi_to_person = from_poi_to_this_person/to_messages
2. NEW_rel_person_to_poi_mess = from_this_person_to_poi/to_messages
3. NEW_rel_expenses1 = expenses/bonus
4. NEW_rel_expenses2 = expenses/salary
5. NEW_bonus_salary = bonus + salary

According to the feature importance plot at the end we see that at least "NEW_rel_poi_to_person", "NEW_bonus_salary", and "NEW_rel_expenses2" seem to be valuable as they belong to the top 10 most important features.

Splitting data

Splitting the data is essential for machine learning tasks. If I were in a data-rich situation, the best approach for model selection and model assessment is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model (Hastie & Tibshirani: 2017). Due to the small sample size I however rely on stratified cross validation to estimate model accuracy and avoid overfitting which occurs when performance on the training set gets better at a cost of the model generalizing less well. It basically happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize. Lastly, I use stratified cross validation because the dependent variable is highly imbalanced and I want to keep the original ratio between "POI = True" and "POI = False" in the sub-samples.

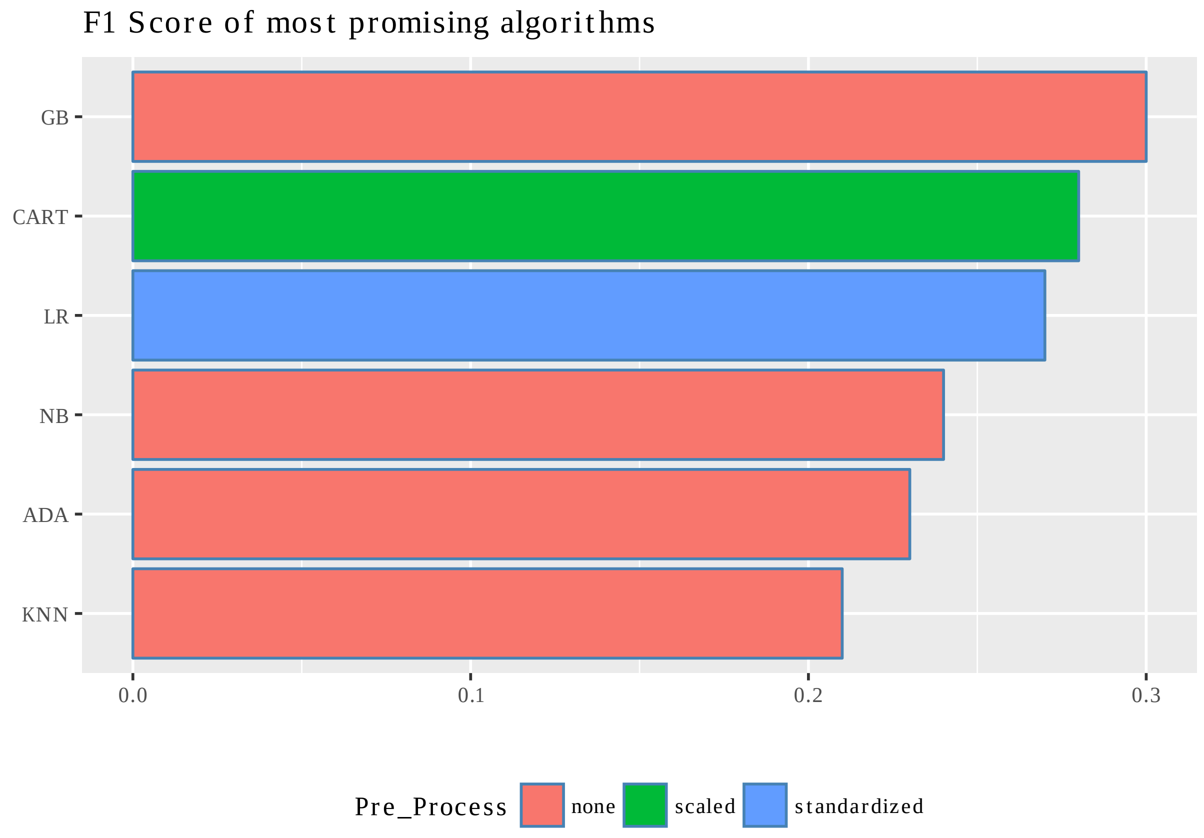
Pre-Processing

Preparing data is required to get the best results from some machine learning algorithms and may include scaleing features, centering features, standardizing features, Box-Cox transforming features, as well as, feature selection and dimensionality reduction. Note that the former is different from the latter. Both methods seek to reduce the number of attributes in the dataset, but a dimensionality reduction method does so by creating new combinations of attributes (e.g. PCA), whereas feature selection methods include and exclude attributes present in the data without changing them.

Spot checking algorithms

Obviously the first question that rises at this point is which algorithms is best to train a model on my data. Unfortunately no one method dominates all others over all possible data sets. Thus this question can only be answered by experience and trial and error. I decided to first run a variety of models with default parameters settings in three different versions: with no transformation, with standardized features, and with scaled features. This approach is especially reliable in this case because sample size and thus computing time is very low.

In order to asses the quality of the models a vast number of different measures, including ROC, Accuracy, RMSE, Kappa, F1-score are available. When choosing a metric three considerations are paramount. First, the nature of the dependent variable, secondly a potential emphasis of some of the different types of errors a model can make, and thirdly the balance of the dependent variable. I rely on the F1-score which is the weighted average of precision and recall to evaluate models. The plot below shows the results for some promising models. While there where some total failures such Linear Discriminant Analysis and surprisingly Random Forest, most of the algorithms scored above 0.20 with default settings and all features selected. Gradient Boosting scored best with 0.30.



Tuning parameters

Next, I try to further improve results of the Gradient Boosting model by optimizing paramaters. In order to do so I manually set different values for the parameters provided by Gradient Boosting. More, precicely I used the following grid which lead to an improvement:

- 'learning_rate': [0.1, 0.05, 0.02],

- 'n_estimators': [50, 100, 150, 200, 400],
- 'max_depth': [2, 3, 4],
- 'min_samples_split': [2, 4, 6],
- 'min_samples_leaf': [1, 3, 5],
- 'min_weight_fraction_leaf': [0, 0.1, 0.2]

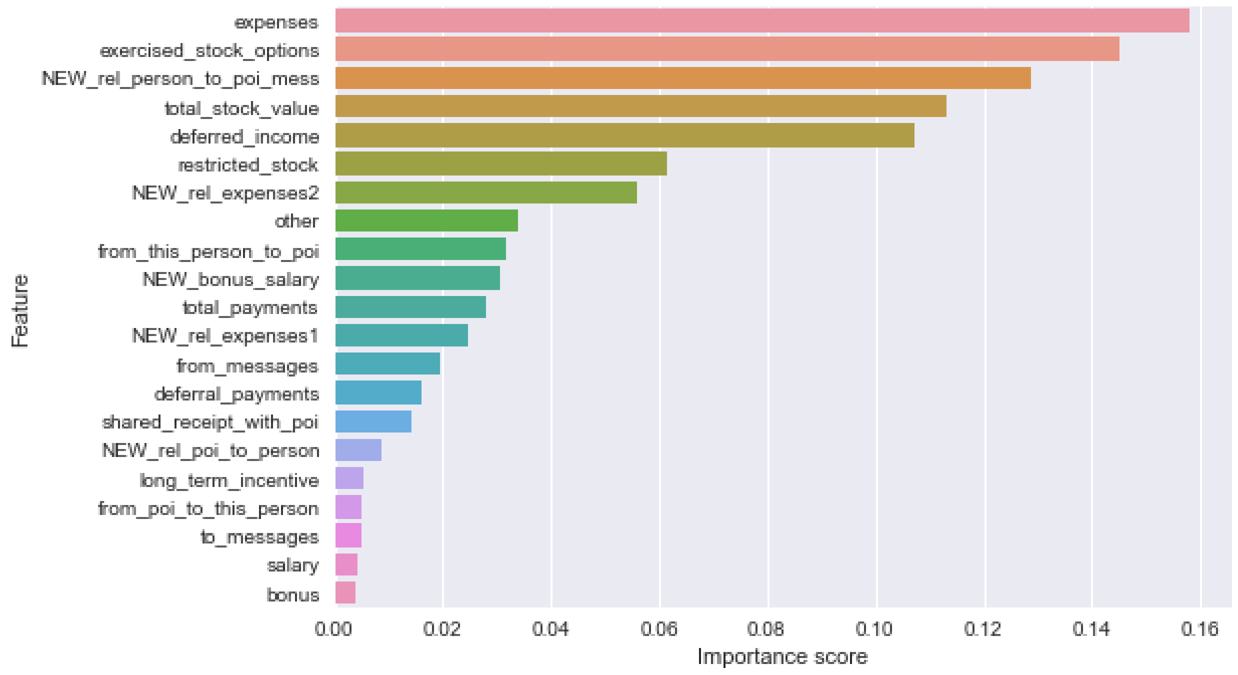
Using the assesment function provided by Udacity I get the following results:

- Accuracy: 0.85
- Precision: 0.43
- Recall: 0.31
- F1: 0.36

The results show that overall 0.86 percent of the observation were correctly classified. However, the issue with accuracy is that it is quiet uselesss if you want to predict a pretty rare condition/event. For such cases Precision and Recall are much better. For the final model we see that of those classified as POI 0.44 percent actually were a POI (Precision = True positive / (True positive + False positive) and that of those that are in fact POIs, 0.31 percent were classified that way. (Recall = True positive / (True positive + False negative). Obviously, Precision and Recall may differ a lot thus F1 which is the harmonized mean of both may also be used to judge a model.

Feature selection, dimensionality reduction and feature importance

Additionally, I testes if I can train a more parsimonious models that yields the same accuracy. Therefore I ran the Gradient Boosting model with preceding PCA analysis and with preceding SelectKBest (which selects the top k features that have maximum relevance with the target variable). It turned out that feeding principal components causes results to become slightly worse. The same applies to feeding less features. Lastly, Gradient Boosting offers the option to check which variables are most important. I therefore run the model on a 80 perecent sized trainig set to asses importance. Results are shown below. Note that, I also excluded the 5 least important features which again caused validation scores to drop.


expenses
exercised_stock_options
NEW_rel_person_to_poi_mess
total_stock_value
deferred_income
restricted_stock
NEW_rel_expenses2
other
from_this_person_to_poi
NEW_bonus_salary
total_payments
NEW_rel_expenses1
from_messages
deferral_payments
shared_receipt_with_poi
NEW_rel_poi_to_person
long_term_incentive
from_poi_to_this_person
to_messages
salary
bonus
Feature
Importance score