DANDP5 – Identify Fraud from Enron

Udacity
Data Analyst Nanodegree

By Anna Signor

1
The goal of this project is to, using machine learning techniques, create a classifier to predict whether or not a certain individual is a person of interest in the Enron case. The massive fraud case gave origin to many interesting points of reflection, one of them being the question: is it possible to determine whether or not a member of a group or organization bears responsibility for a reprehensible act traced back to said group or organization? If so, how? Obviously this specific case dates back to the 1990's, but the question itself is a current one.

A significant amount of data, that normally is internal to a company, became public knowledge as consequence of legal proceedings. In this project the data will be used to train and test a machine learning algorithm to determine whether or not a given individual is a Person of Interest, or POI.

Please see addendum I: Outlier Investigation and Feature Selection Documentation for details on outliers were handled. I used simple plots to visualize the data and investigate if an outlier made sense. For example, in director fees, there are lot of zeroes and a few large values, but that makes sense. However, a single employee with a salary of several tens of millions, did not. In fact, that was a total.
In investigating missing data, outliers and inconsistencies, a few points were removed for having negative values that seemed inconsistent with the type of feature, and a the key/values "TOTAL" And "THE TRAVEL AGENCY IN THE PARK" were removed, as they don't seem to refer to people.
I ended up removing to points for not referring to people, and all that had 0 for salary. I thought the 0 salaries were strange, and in fact, it had a positive impact on metrics when removed, despite reducing the number of points to 94.

Algorithm	**precision**	**recall**	**F1**	**accuracy**
Tuned random forest, including zero-salaried points	0.32	0.37	0.34	0.81
Tuned random forest, excluding zero-salaried points	0.34	0.38	0.36	0.82

2

Please see addendum I: Outlier Investigation and Feature Selection Documentation for details on feature selection. I wanted to use the financial features alone, to see if we can get answers only from financial information. I started with all of them, and by plotting them one by one against a simple equally spaced x-axis (I inverted if negatives), with the labels as colors. I was able to visually eliminate the ones that did not look discerning.

I figured a lot of the features would be redundant, which is why I engineered a smaller amount of features using PCA. I experimented with plotting features one against the other to see some alignment, however I decided the PCA method will do a much better job than my intuition in finding the correlated components.

I created a feature by taking the ratio between stock options that were exercised, and the total stock value. However, there was not a positive impact on performance.

Algorithm	**precision**	**recall**	**F1**	**accuracy**
Tuned random forest, including new featured	0.33	0.37	0.34	0.81
Tuned random forest, excluding new feature	0.34	0.38	0.36	0.82

Scaling was used for two reasons: to be able to feed the same data into other algorithms that were tried and because I used PCA.

The final number of data points after cleaning was 94 and 12 features were utilized. The balance of classes is: 77 non-POIs and 17 POIs.

3

The algorithm used was Random Forest. I also tried Gaussian and Bernoulli Naive Bayes. The first metric I considered was accuracy. However, I quickly realized the best strategy was to optimize to F1, because that is the metric on which the project will be evaluated, and more importantly, the data is extremely imbalanced. In preliminary further tuning of each classifier, I was able to get Random Forest to respond better to parameter variations trying to get precision and recall to improve.
Part of this perhaps has to do with the fact that this is a small and well known data set, which makes it easy to intuit which Forest parameters to adjust and how, given that Decision Trees is not a black box method. **Please see Addendum II: Recorded Metrics during Development.**

4

Tuning the parameters of an algorithm is the process of setting the parameters of the instance of the algorithm in use (the class), to best respond to your data and your needs. In this case, with Random Forests, as would have also been the case with Decision Trees, the biggest risk is to overfit, meaning to

create an overcomplicated decision boundary, causing the algorithm to make good prediction on the specific training set used, but fail to do so in other subsets of the data. This is a more pronounced issue with Decision Trees, but not absent in Random Forests. When using Forests or Trees, it is fundamental to cross-validate as a way to avoid overfitting.
I tuned the parameters in conjunction with validation using `GridSearchCV`. Addendum II: Recorded Metrics during Development.

Because of the long time it takes to run even a simple random forest grid search, it was done in iterations. One parameter and 3 values at a time, `GridSearchCV` was ran and each time the best value for a given parameter is determined, it is extracted with the `best_estimator` and hard-coded into the instance of `RandomForestClassifier` passed as starter classifier to `GridSearchCV`.

The table below has the full grid used over the many iterations Addendum II will have all the recorded metrics and parameter stages. An empty dictionary was then passed to `params` with the sole purpose of being able to display the code with the validation part included. The same or similar performance should be observed by designating `clf` as an instance of Random Forest Classifier with the parameter values in the third column of the table below specified for the instance.

Parameter	Default (out of the box)	Values in grid search	Value utilized
class weight	None	None, balanced subsample, balanced	balanced
criterion	Gini	entropy, gini	entropy
max depth	None	10, 20, 40, 45, 50	40
max features	Auto	4, 5, 6,7,None	7
max leaf nodes	None	10, 20, 40, 43, 45, 50, 55, 60	None (default)
min samples leaf	1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1 (default)
min samples split	2	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	2 (default)
min weight fraction leaf	0.0	0.0, 0.1, 0.2, 0.3, 0.4, 0.5	0.1
n estimators	10	8, 9, 10, 12, 14	8
n jobs	1	1, 2, 4, 8, 16	1 (default)

Comparing the performances before and after tuning:

Algorithm	precision	recall	F1	accuracy
Random Forest out of the box	0.43	0.14	0.21	0.86
tuned Random Forest	0.34	0.44	0.38	0.81

It is not surprising that the algorithm would produce seemingly good accuracy and precision in its out of the box version. As explained above, without tuning the forest is likely to overfit and with the imbalance of classes skewing negative, those two metrics being inflated is exactly what one would expect. The big indication here that the forest needed tuning was the very poor recall.

5
The validation was done using `GridSearchCV`. Some pitfalls of validation are to tune to the wrong metric, for example. When using a tool like `GridSearchCV`, it is important to ensure the metric being used is the same that you actually want to optimize.
A different, more obvious mistake one can make is to fail to keep training and testing data rigorously separate. Testing on the data used for training will artificially produce good metrics.

6 (also see table above)
Recall - average 0.44
Recall measures the rate at which the classifier can make a correct positive prediction, that is to say, the amount of correct positive predictions compared with the amount of actual positive-labeled data points. This was the metric that was the hardest to bring up above 0.3. What this means is that my classifier was having a hard time “seeing” the POIs, and yielding a lot of false negatives.
Accuracy - average 81%
Accuracy is the rate at which the algorithm correctly classifies a data point. This is not a great metric on which to rely in most cases, this one included. This is so because of the class distribution imbalance, meaning there are much more non-POIs than POIs, an algorithm that simply classifies all points negatively will yield an impressive looking accuracy. It is no surprise the accuracy decreased with tuning.

References

Forum posts:

https://discussions.udacity.com/t/error-indexerror-index-140-is-out-of-bounds-for-size-128/210967/5

https://discussions.udacity.com/t/help-im-stuck/226383

https://discussions.udacity.com/t/doing-worst-with-gridsearchcv/210499/6

https://discussions.udacity.com/t/stuck-on-tuning-my-classifier/209595

https://discussions.udacity.com/t/tester-py-error-message/205672

https://discussions.udacity.com/t/pca-not-affecting-svm-and-random-forest/200381/4

https://discussions.udacity.com/t/importerror-no-module-named-feature-format/175368/12

Scikit Learn Documentation:

http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

http://scikit-learn.org/stable/modules/naive_bayes.html#bernoulli-naive-bayes

http://scikit-learn.org/stable/modules/svm.html