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.

2

I wanted to use the financial features alone, to see if we can get answers only from financial information. I figured a lot of the features would be redundant, which is why I engineered a smaller amount of features using PCA. Since the number of principal components cannot be included in the grid for cross validation, different component numbers were tested on the out of the box classifier, and then after tuning it was revisited to ensure the best number of PCs.

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.

In investigating missing data, outliers and inconsistencies, a few points were removed for having negative values that seemed inconsistent with the type of feature.

3

The algorithm used was Random Forest. I also tried Gaussian and Bernoulli Naive Bayes, as well as C - Support Vector Classification. The first metric I considered was accuracy, for which, in conjunction with dimensionality reduction, Random Forests and Naive Bayes perfomed the best. 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.

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 issues 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.

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 via 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. An empty dictionary was then passed to the variable "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 subsample
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	45
min samples leaf	1	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	5
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	0.0 (default)
n estimators	10	8, 9, 10, 12, 14	10 (default)
n jobs	1	1, 2, 4, 8, 16	1 (default)

Comparing the performances before and after tuning:

algorithm	precision	recall	accuracy
Random Forest out of the box	0.37	0.13	0.85
tuned Random Forest	0.30	0.34	8.82

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, with the cross validation parameter set to StratifyShuffleSplit, due to the small size of the dataset. 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 0.34
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 82%
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.

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