Enron Person of Interest Identifier

Introduction

The Enron Fraud was one of the largest Corporate Fraud cases in U.S. History. It happening after the advent of the internet and the detail with which the company was investigated afterwards, allows us to look at more of the everyday data that the company produced. In this project, the email and financial data for 146 was used in an attempt to use a machine learning algorithm to discern persons of interest in the case. A person of interest, in this case, is a person who was either indicted for fraud, settled with the government, or testified in exchange for immunity.

The Enron Data

The Enron data set includes financial and email information regarding 145 Enron employees, and 18 persons who were considered persons of interest. The Enron email corpus includes many emails sent to and from 84 of the persons listed in the data set. The Enron Data included a few outliers. One of which was found to be a spreadsheet “Total” Line which was removed. The other outliers were found to be people, who made a lot of money from the fraud, and were not removed.

Feature processing

As I knew that I would be using PCA (or an algorithm which included a feature selection parameter) I included all of the financial features in the data set. These features were: 'poi','salary', 'exercised_stock_options', 'deferral_payments', 'total_payments', 'other', 'bonus', 'restricted_stock', 'deferred_income', 'long_term_incentive','restricted_stock_deferred', 'total_stock_value', 'expenses', 'shared_receipt_with_poi'. Leaving out the email related ones as they would be used later.

I also included a TFIDF feature consisting of the single words used by all individuals who were considered Persons of Interest, and used by most of the people who were not Persons of Interest. I used the Select K Best feature selection algorithm to whittle the several thousand resultant word-value pairs down to the 50 most effective text features.

The text features selected were: pressed, coordiante, repurchases, chick, forecastingbob, feinsteinsmith, characterize, invalidation, janus, bhatnager, minutes, backdated, updatejan, bartram, bring, designesss, merenda, zdnet, edmiston, dinning, allegheny, suiza, levine, citibank, remarksrick, updatebill, issuance, updategeorge, ibmebs, selectionstrategy, patience, prejanus, lonon, netenas, firstworld, jonesenergy, updatepaula, ebsibm, longerdated, additonal, ciscon, businessesmike, asprelated, graduate, reemphasize, simulationscott, perminder, yesterdyas, traded, and bindra. No matter how much parsing I did I was unable to insure that all pairs of words were separated, and that no unwanted values made it in to the final dataset. However, these proved wonderfully effective as features in this particular exercise.

As an interesting point of information, I found that the email corpus data provided much stronger predictive value than I expected. My performance metric scores rose from a high precision score with a recall score below 0.2, to both of them well above 0.5.

Algorithm Selection and Tuning:

I chose to try several algorithms in my efforts to select and tune the best algorithm. I decided also to use Grid Search CV to help tune my parameters. I wanted to try K Nearest Neighbours, a Decision Tree Classifier, and an SVM. To see which of the methodologies would result in the best performance. I found that both the Decision Tree Classifier was just not quite as reliable as the other two classifiers. The SVC, when tuned with Grid Search CV, resulted in performance metrics which were almost as good as the KNN values, but had a lower recall value. I settled on the K Neighbours algorithm because it performed best for me. Providing me with an accuracy score of over 0.91 and precision and recall scores well above 0.5. Below I will list the parameters I put into the Grid Search CV parameter tuning algorithm and the resultant selected values for each of the algorithms that I tested and tuned.

Decision Tree Classifier:

Parameters and Resultant Values:

criterion: entropy

max_features: 0.05

min_samples_split: 5

min_samples_leaf: 1

class_weight: None

presort: False

Performance Metric Scores:

Accuracy: 0.86480

Precision: 0.48997

Recall: 0.34200

K Nearest Neighbours:

Parameters and Resultant Values:

Scaling with default MinMax Scaler

PCA:

n_components: 4
whiten: True

KNN:

n_neighbors: 1
algorithm: kd_tree
weights: uniform
leaf_size: 3

Performance Metric Scores:

Accuracy: 0.91547

Precision: 0.76599

Recall: 0.52700

Support Vector Classifier:

Parameters and Resultant Values:

kernel: rbf

C: 10

max_iter: 15

Performance Metric Scores:

Accuracy: 0.91333

Precision: 0.76718

Recall: 0.50250

Validation and Performance

The validation method built in to the test method was used to validate my results. This used a stratified shuffle split cross validation method. This randomly splits and shuffles the data into training and test sets. In this case it was split one-thousand times, and allowed each run of the algorithm to make 15000 predictions, in spite of having a data set with only 145 individuals and 18 persons of interest. In more general terms validation is the process by which you split you data into training and test sets. This allows you to estimate your algorithm's performance, and helps to prevent over-fitting. A classic mistake in cross validation is to use the same data to train the algorithm that you use to test the algorithm. This results in over-fitting. Over-fitting will negatively impact your models predictive value for any data that is not identical to the training/test set.

Discussion and Conclusions

First I will define the performance metrics used. Precision, in this case, means the likelihood that the prediction of a person of interest is actually a person of interest. With a precision of 0.76599, 23.5% of positive results would be false positives. Likewise, recall is the likelihood that in a test set with a person of interest in it, that person would be identified as a person of interest. With the recall score I got with my final model 52.7% of the time a person of interest would be identified given that there was one in the test set.

The numbers that I got are reasonably good. I feel that this sort of classifier may help in narrowing down a wide field of suspects, into a smaller more easily investigated suspect pool. That said, however, this level of accuracy would not be high enough to classify someone purely based on this information.

The biggest challenge that I encountered was the TFIDF feature. It required much fiddling with, removing features which were useless for one reason or another, like features which consisted of just numbers, and trying to remove as much as possible the signatures in emails. All of this parsing increased the run time dramatically, and resulted in a data set which was massive and difficult to work with. I managed after quite a lot of playing around with it and waiting for it to run, to find that it was in fact worth the effort. It resulted in the largest increase in my algorithms performance metrics.

The only way that I can think of to make the algorithm more accurate is to provide the algorithm with more data. For some reason only 18 of the 35 identified Persons of Interest were included in the data set. With more data to draw from you could expect more accuracy from a less finely tuned algorithm, thus risking over-fitting a lot less.