Identify Fraud from Enron Email

Mitchell Barnes-Wallace

Overview

In the early 2000's the Enron Corporation, which was one of the largest energy corporations in North America, was caught up in one of the biggest financial scandals in United States history. Many of the company's key executives were caught "cooking" the books, which for years, the company was able to get away with due to the complexity of their structure and finances. People at fault in this scandal were considered to be person's of interest or "POI"'s. In this study, I sought to identify Person's of Interest using machine learning algorithms.

Data Exploration & Outlier Investigation

The data used in this project came from two places, the email data of certain people at the company, and the financial data of employees at the company. In total there were 146 employees in the data sets. Looking at the financial data, one might assume that factors like bonuses, stock payouts, etc would indicate employees that could be POI's, as they are making the most money from the fraud. In the email data set, variables such as emails to and from an individual and a POI, might indicate that the person communicates a lot with a POI, and thus might be a POI themselves.

There were numerous missing values in both the financial data and the email data. For financial data, all of the NaNs were replaced as zeros. Additionally, there were two employees ROBERT BELFER and SANJAY BHATNAGAR, whose data had shifted when it was extracted, and put in the correct data back into the data set. Additionally, Eugene Lockhart had data that was completely empty, and I removed his information. In looking through the email data, I noticed a few strange outliers, 'THE TRAVEL AGENCY IN THE PARK' and 'Total', neither of these two data points are actually people, and thus were removed from the data set.

The last data cleaning that I addressed, was how to handle the NaN's in the email data set. Removing the entire employee would result in cutting our data set almost in half, which wouldn't work with how small our data set is to start. The second solution would be to replace all NaN's with the median value for its respective column in the data set. This will allow me to keep all of the dataset for the Machine Learning Algorithms, which for such a small dataset to begin with, is important. Following the cleaning the dataset contains In summary, the dataset contains 143 total employees, with 12 total features, 18 POI's and 125 Non-POI's.

Feature Selection

Initially, I decided to include all of the features in analyzing the data set (the full list will be provided below along with their SelectKBest Scores). Along with the features given, I decided to implement two engineered features:

- 'stocks_payments_delta' : difference between total stock value and total payments
- 'ratio_poi_total_messages': total emails to or from a POI divided by the total emails this person received.

These two features attempted to identify features in the data that were not previously accounted for, such has high percentage of overall emails to or from a POI might indicated that person is also a POI. Plugging all of these features into the SelectKBest, yielded the below results:

Feature Name	K-Score
exercised_stock_options	24.82
total_stock_value	24.18
bonus	20.79
salary	18.29
deferred_income	11.46
long_term_incentive	9.92
restricted_stock	9.21
total_payments	8.77
shared_receipt_with_poi	7.39
loan_advances	7.18
expenses	6.09
from_poi_to_this_person	4.23
other	4.19
ratio_poi_total_messages	3.82
from_this_person_to_poi	2.19
director_fees	2.13
to_messages	0.87
deferral_payments	0.22
from_messages	0.19
stocks_payments_delta	0.16
restricted_stock_deferred	0.07

Overall, it appears that the exercised_stock_options value potentially is the highest indicator as to whether or not this person is a POI, my two engineered features did ok, with the ratio feature performing 14th best, while the stocks minus payments feature scored second to last. For the algorithms ultimately tested, only the Ada Boost classifier and the Naïve Bayes did not have a min max scalar applied to them, as they are not impacted by this scaler. Additionally, all of the algorithms except for the Naïve Bayes, had PCA applied to them, in an effort to improve the scoring for each algorithm.

Algorithm Selection

To start, I tested a variety of un-tuned algorithms to see what could yield me accuracy and precision scores closest to 0.3, yielding the below results (best scores highlighted in yellow).

	Accuracy	Recall	Precision	F1
decision_tree	0.816	0.29	0.269	0.253
logistic_regression	0.868	0.0544	0.148	0.069
naive_bayes	0.8	0.242	0.256	0.216
nearest_neighbors	0.87	0.0719	0.281	0.11
random_forrest	0.859	0.146	0.289	0.172

Based on these results, I ran the Decision Tree and Naïve Bayes classifiers through a stratified shuffle grid search, along with an Ada Boost classifier, with the Decision Tree as the base estimator. The grid search, ran each algorithm and attempted to find the parameters producing the best F1 score. Since the assignment was to optimize both Precision and Recall, F1 score made the most sense to optimize as it is an average of the both precision and recall. Running these algorithms through the grid search, yielded the following results:

	Accuracy	Recall	Precision	F1
ada_dt	0.802	0.244	0.231	0.216
decision_tree	0.816	0.272	0.274	0.243
naive_bayes	0.838	0.299	0.342	0.297

With Naïve Bayes yielding the best results, I decided to go with this algorithm, not to many features to tune for this algorithm other than the K features to be included in the learning.

Parameter Tuning

While Naïve Bayes didn’t require parameter tuning other than the number of K features selected, I also tuned the other two algorithms selected for parameter tuning. Parameters in machine learning algorithms play a large roll in the overall performance of the algorithm, they can both boost performance and sabotage performance. As previously alluded to, I used a stratified shuffle grid search algorithm in order to select the correct parameters for the 4 algorithms selected for the final scoring. For each algorithm the following parameters were tuned.

Algorithm	Parameter	Tested Parameter Values
Ada Boost	Number of Estimators	Range from 1-10
Decision Tree	Criterion	Gini or Entropy
	Minimum Samples per Leaf	Range from 5-50, increments of 5
	Maximum Depth	Range from 1-10

The stratified shuffle grid search cross validator tuned each of these parameters to which parameters produced the best F1-score, as it is an average of both precision and accuracy.

For the best performing mode, in this case it was Naïve Bayes, I iterated through all of the features in the data set and found the K features that returns the best results:

K	Accuracy	Precision	Recall	F1
2	0.847	0.328	0.192	0.226
3	0.842	0.332	0.240	0.258
4	0.845	0.361	0.273	0.287
5	0.846	0.365	0.278	0.292
6	0.845	0.356	0.284	0.295
7	0.844	0.356	0.299	0.304
8	0.847	0.370	0.298	0.308
9	0.848	0.368	0.298	0.306
10	0.847	0.360	0.296	0.305
11	0.847	0.360	0.300	0.307
12	0.834	0.328	0.310	0.294
13	0.821	0.315	0.327	0.292
14	0.810	0.308	0.341	0.290
15	0.796	0.300	0.349	0.285
16	0.761	0.281	0.393	0.284
17	0.718	0.256	0.401	0.264
18	0.692	0.236	0.408	0.251
19	0.692	0.236	0.423	0.259
20	0.689	0.238	0.433	0.267
21	0.688	0.239	0.435	0.268

I selected the highlighted row (K=11), as it has the highest F1 score, with also having Precision and Recall above 0.30.

Validation

In training a machine learning algorithm, there must be a separate testing dataset from the training set in order to prevent overfitting of a MLA. The MLA trains on the training set, and then is scored on the testing set. In order to optimize a MLA, the training set must be split into a training set and a validation set, and the scores here are optimized before the MLA is then tested on the testing set. For example, in the case of a decision tree algorithm, it would be really easy for the MLA to make many exceptions in order to fit its algorithm perfect to the dataset, however with validation, the validation dataset is used to tune the MLA before it is applied to the test set.

For my parameter tuning I used a GridSearch with a Stratified KFold validation method. This method provides stratified folds of the dataset that maintains the same percentage of each class as the full dataset. This prevents training groups from not having any POI's, as there are only 13% POI's in the total dataset, and it would be easy to sample a group without any POI's within the group.

Final Performance

The final results for the Naïve Bayes machine learning algorithm produced the below results through the tester.py script.

Accuracy	Recall	Precision	F1	F2
0.850	0.319	0.418	0.362	0.335

The results of this tester script indicates that 84.9% of all of the data points were classified correctly (e.g. POI vs. non-POI). This metric isn't particularly useful for our purposes, as most of the data isn't a POI, and what we are really concerned about its how accurate we can predict whether or not someone is a POI, the recall and precision metrics play a bigger role. Precision tells us, for all of the predicted POI, how many were accurately predicted POI's. For our example, 41.8% of all predicted POI's were actually POI's. This is a pretty solid estimate, as the overall percentage of POI's in the data set is around 13%, so we are more accurately predicting POI's then just randomly pulling a name out of the dataset. Recall is a measure of percentage many true labels were actually predicted true by the algorithm. In our example, our recall score is a 31.9%, which means that 68.1% of all POI's will not be classified correctly. This value is pretty darn high, as most of the POI's will not be classified as such, and the non-POI's could be classified as a POI in their stead, which could ruin lives for innocent people. If I were to re-tune this algorithm, I would likely put an emphasis on recall, as this metric is most pertinent in identifying POI's.

Sources:
1. https://en.wikipedia.org/wiki/Precision_and_recall
2. http://optunity.readthedocs.io/en/latest/notebooks/notebooks/sklearn-automated-classification.html
3. https://arxiv.org/pdf/1710.04725.pdf
4. https://machinelearningmastery.com/handle-missing-data-python/