Stephen D. Wells
Udacity Nanodegree: Data Analyst
August 24, 2015

Identifying Fraud in Enron
Machine Learning Project

GOAL

In 2002, the Harvard's Ig Nobel Prize for the 'Most Creative Use of Imaginary Numbers' went to a company that two years prior had received it's sixth, 'America's Most Innovative Company' award from Fortune magazine[1]. In 2000, Enron was the seventh-largest company in America with over $100 Billion in revenue.[2] The next year as the result of fraudulent accounting practices, they were bankrupt.[3] During the Federal Energy Regulatory Commission's investigation, internal company emails were released to the general public.[4] Combining this dataset with the Securities and Exchange Commission financial data, along with Persons of Interest derived from newspaper articles we will answer the question, '**Can we identify Persons Of Interest (POI) solely from this dataset?**'

DATA EXPLORATION	
146	Total No. of Data Points
18/144	Allocation Across Class (POI/non-POI)
35	Total Gathered POIs
14.38%	NaN for Total Payments
0 of 18	POIs with NaN for Total Payments
21	Number of Features
95	Qualified Salary
111	Known Email Address

Features with Many Missing Values
Loan Advances
Director Fees
Restricted Stock Deferred

[1] https://en.wikipedia.org/wiki/Enron

[2] http://www.forbes.com/2002/01/15/0115enron.html

[3] https://en.wikipedia.org/wiki/Enron_scandal

[4] https://www.cs.cmu.edu/~./enron/

OUTLIER INVESTIGATION

Machine learning algorithms were used to make predictions and calculate the accuracy of those predictions. The worst case of outliers uncovered was due to a TOTAL in the financial statement which was read in as a line item. This was removed by hand. Four other outliers were left in as they were considered valid data points. A technique to programmatically remove outliers by discarding 10% of the points with the largest errors was introduced and used to help identify additional potential outliers, however it was determined that those points were valid. A careful reading of the field names revealed, 'THE TRAVEL AGENCY IN THE PARK' which, not being an individual, was removed.

LEADERSHIP	
CEO	Jeffrey Skilling
Chairman	Kenneth Lay
CFO	Andrew Fastow

The financial dataset does not include all POIs hence some are missing.

Kenneth Lay took home the most at **$103,559,793** during this time but his conviction was vacated due to his untimely death before he could appeal.

OPTIMIZE FEATURE SELECTION/ENGINEERING

These types of supervised ML algorithms require re-training at periodic intervals. The "best" algorithm for the Netflix Prize was presented with $1 Million, though it wasn't used in production because of the engineering effort required for it's minimal gains.[5] The challenge in this case was to achieve recall and precision values of 0.3+. In an effort to simulate a real-world scenario and utilize as few resources as possible, a minimum of features were selected.

Exercised Stock Options, Total Stock Value and Bonus were chosen in combination with sklearn's univariate feature selection **SelectPercentile,** which selects features based on the percent with

Sample SelectPercentage Results

Percentage: 15

Precision: 0.36

Recall: 0.38

Features:

- exercised_stock_options
- total_stock_value
- bonus

[5] http://www.forbes.com/sites/ryanholiday/2012/04/16/what-the-failed-1m-netflix-prize-tells-us-about-business-advice/

the highest scores. In this case, those scores are generated using the F- score of an ANOVA calculation with the attribute 'f_classif'.

The script 'feature_engineering.py' was broken out to display these results in a more meaningful way. It's output uncovers the internal list with weights (scores) assigned to each feature which you can review in the chart to the far right.

Sample Results by Percentiles

%	Precision	Recall	Features
10	0.22	0.20	2
15	**0.36**	**0.38**	**3**
20	0.32	0.33	4
25	0.30	0.32	5
30	0.28	0.29	6
35	0.27	0.29	6
40	0.26	0.27	7
45	0.26	0.29	8
50	0.27	0.30	9
55	0.29	0.32	10
60	0.29	0.33	11
65	0.30	0.32	12
70	0.28	0.30	12
75	0.29	0.31	13
80	0.30	0.32	14
85	0.30	0.32	15
90	0.29	0.30	15
95	0.30	0.32	17
100	0.29	0.31	18

SelectPercentage Feature Results

Weight	Feature
24.82	Exercised Stock Options
24.18	Total Stock Value
20.79	Bonus
18.29	Salary
11.46	Deferred Income
9.92	Long Term Incentive
9.21	Restricted Stock
8.77	Total Payments
7.19	Loan Advances
6.09	Expenses
5.24	From POI to this Person
4.19	Other
2.38	From this person to POI
2.13	Director Fees
1.65	To Messages
0.22	Deferral Payments
0.17	From Messages
0.07	Restricted Stock Deferred

New Features

fraction of deferred income to total payments

This feature assumes that non-POIs would believe that the company would continue to grow and be in a better position to make those payments. POIs would know that the foundations were crumbling and want to be paid before the company ran out of funds.

sample impact of new features using final algorithm (Decision Tree)

Feature	Precision	Recall
neither feature	0.37	0.38
milk	0.38	0.40
fraction of deferred income to total payments	0.40	0.39
combined features	0.40	0.41

milk (as in MILKing the company)

The value "milk" was scaled using the following ratio.

$$\frac{(\text{Expenses} + \text{Deferral Payments})}{1 + (\text{Loan Advances} + \text{Long Term Incentive} + \text{Deferred Income})}$$

The belief that POIs have an incentive to get as much out of the company as quickly as possible since they don't see a long-term future in the company, drove the decision to create a label "milk" as in 'Milking the Company'. Expenses include consulting, reimbursements from the company and deferral payments are distributions from deferred compensation. The company must pay these now whereas loan advances, long term incentives and deferred income are future payments. If employees believe that the company won't have the money to pay them in the future, they will want to collect what they can now.

Feature Importance

with sample algorithm performance

- **.30** Exercised Stock Options
- **.27** Bonus
- **.19** Total Stock Value
- **.14** Fraction of Deferred Income to Total Payments
- **.11** Milk (defined below)

FEATURE IMPORTANCE

As seen in the table to the left, each feature adds to the final result. They are sorted by their respective value. Though these values change with each iteration, you can count on Exercised Stock Options always providing more weight to the final algorithm than that of our created features.

PICK AN ALGORITHM

The goal is to classify individuals as either a POI or non-POI. Since this is a classification problem, regression and clustering algorithms were eliminated. Further to this, it's not a "Big Data" issue as there are less than 100,000 samples, so there was no need to go with an SGD Classifier or kernel approximation. This required testing of the following five algorithms:

- Naive Bayes (GaussianNB)
- Support Vector Machines
- LinearSVC
- Ensemble: Random Forest
- Decision Trees

GaussianNB was simple to implement but even with excellent recall numbers, no amount of tuning afforded respectable precision results. Support Vector Machines turned out great precision values and only 0.0025 for recall. LinearSVC returned both precision and recall numbers that looked more even, but didn't yet yield the desired results. SVM algorithms are not scale invariant and therefore required scaling as they deal with Euclidian distances. RandomForest resulted in higher precision than the recall number and it took a long time to process. Decision Trees gave back very fast results and the defaults were very close to the desired precision and recall targets.

Algorithm	Precision	Recall
GaussianNB	0.15	0.75
SVC	0.71	0.00
LinearSVC	0.20	0.30
RandomForestClassifier	0.45	0.24
tree	0.33	0.34

TUNE AN ALGORITHM

Algorithm tuning is vital to improving performance as the attributes for a given method help to refine how that algorithm is processed. After tuning the algorithms above, the DecisionTreeClassifier was chosen and tuned using the following parameters:

GridCV Trials of Tuning Settings on DecisionTreeClassifier

several parameters were tuned and included the following…

criterion gini, entropy

splitter best, random

max_features 1, 2, 3, 5, 9, 0.1, 0.2, 0.25, 0.5, 0.75, 0.8, 0.9, 0.99, auto, sqrt, log2, None

max_features 0.1, 0.2, 0.25, 0.5, 0.75, 0.8, 0.9, 0.95

class_weight None, auto

max_leaf_nodes None, 2, 3, 4, 5, 6, 7, 8, 9, 10

After starting with a grid search (GridCV) the parameters were manually tuned for the decision tree. Interestingly, adding *max_leaf_nodes* as a parameter to **GridSearchCV**( ) yielded worse results**.** In this case, the default parameters gave the best results, with the exception of changing the **splitter** attribute from *best* to *random*. The best algorithm-tune combination from these results were selected for the final analysis.

CLASSIFIER

A decision tree classifier was used in the final product. This decision was based on a balance between precision and recall values. For instance, though the SVC and Random Forest algorithms returned great precision results their recall values were abysmal, while on the other hand GaussianNB returned great recall but limited precision values. LinearSVC was tested and removed because the overall numbers were too low as opposed to being one-sided.

EVALUATION MATRICS

Recall that the goal was to achieve higher than 0.3 for both precision and recall. In this case the average precision value was 0.39. which translates to mean that 39% of the time we don't label a POI as a Non-POI. Recall looks at the other side of the equation and in plain English means that 40% of the time we correctly classify POIs.

VALIDATION STRATEGY

To validate the data it was split into two groups which were labeled as training and testing. The algorithm was then used to train the data and validated using the testing data. A naive mistake would be to use the testing data to train, because even though it will generate great results in a simulated environment, those results will not translate to a real-world environment. Since the dataset is small, a **StratifiedShuffleSplit** as part of the cross_validation package from sklearn is the best choice. It will return stratified randomized folds (1,000 in this case) which are produced by preserving the percentage of samples for each class.

Precision and Recall Values

	Precision	Recall
mean	**0.39**	**0.40**
median	0.39	0.40
max	0.42	0.42
min	0.38	0.38
first quartile	0.39	0.39
third quartile	0.40	0.40

The DecisionTreeClassifier was run 1,000 times where it recorded the precision and recall values each time. These statistics were generated using this data.

ALGORITHM PERFORMANCE

Code was added to the *poi_id.py* script to run the algorithm with the tuning parameters 1,000 times and record the results into a csv file (*results.csv*). The table to the left clearly shows that precision and recall results above 0.3 for each iteration were achieved.

PERSONAL INSIGHTS

Enron's upper management didn't write their own emails. In fact, according to this dataset, Ken Lay never wrote a single email as he had a personal assistant write and send emails on his behalf. This means that we should not use word choice as part of our analysis, as it would only be analyzing his assistant's vocabulary. The metadata, "to" information is slightly more useful as we can assume that management gave direction as to where their emails were sent. Though limited information could be garnered from the emails under the assumption that non-POIs emailed each other more than they emailed POIs, it proved possible to manipulate the financial dataset with enough insight to make that analysis obsolete. This is why I dropped the email metadata from my analysis.

I did run across a video[6] the other day that claimed that you can tell who a director is at Enron by comparing the number of the emails they sent to the number of nodes. In other words, those that emailed people who didn't email each other were more likely to be a director in the company.

[6]https://www.youtube.com/watch?v=GBzoNgqF-gQ&t=38m17s