Identifying Fraud from Enron Emails and Financial Data

Swaroop Kallakuri

Monday, November 16, 2015

`Introduction to Machine Learning Project`

Data Analyst Nanodegree

Introduction

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, there was a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for to executives.

Utilizing `scikit-learn` and machine learning methodologies, I built a "person of interest" (POI) identifier to detect and predict culpable persons, using features from financial data, email data, and labeled data--POIs who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Short Questions

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

The goal of this project was to utilize the financial and email data from Enron to build a predictive, analytic model that could identify whether an individual could be considered a "person of interest" (POI). Since the dataset contained labeled data--culpable persons were already listed as POIs--the value of this model on the existing dataset is limited. Rather, the potential value such a model may provide is in application to other datasets from other companies, to potentially identify suspects worth investigating further. The dataset contained 146 records with 14 financial features, 6 email features, and 1 labeled feature (POI). Of the 146 records, 18 were labeled, a priori, as persons of interest. Through exploratory data analysis and cursory spreadsheet/CSV review, I was able to identify 3 candidate records for removal:

- `TOTAL`: This was an extreme outlier for most numerical features, as it was likely a spreadsheet artifact.
- `THE TRAVEL AGENCY IN THE PARK`: This record did not represent an individual.
- `LOCKHART EUGENE E`: This record contained no useful data.

The `TOTAL` record became most apparent after visualizing the financial data on scatter-plots. Following data-cleaning, 143 records remained.

What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that doesn't come ready-made in the dataset--explain what feature you tried to make, and the rationale behind it. If you used an algorithm like a decision tree, please also give the feature importances of the features that you use.

In order to optimize and select the most relevant features, I leveraged the use of the `scikit-learn`'s `SelectKBest` module to select the 10 most influential features. Their associated scores and number of valid (non-NaN) records are listed in the table below:

Feature	Score↑	Valid
exercised_stock_options	24.815	102
total_stock_value	24.183	126
bonus	20.792	82
salary	18.290	95
deferred_income	11.458	49
long_term_incentive	9.922	66
restricted_stock	9.212	110
total_payments	8.772	125
shared_receipt_with_poi	8.589	86
loan_advances	7.184	3

Interestingly, `loan_advances` had a considerably high score, despite having only 3 non-NaN values. The K-best approach is an automated univariate feature selection algorithm, and in using it, I was concerned with the lack of email features in the resulting dataset. Thus, I engineered a feature, `poi_interaction` which was a ratio of the total number of emails to and from a POI to the total number of emails sent or received. I also included a financial aggregate feature, `financial_aggregate`, that was the combined sum of `exercised_stock_options`, `salary`, and `total_stock_value` to capture how much wealth--liquidated and invested--an individual has. Both features were included in the final analysis, for a total of 12 features, as they slightly increased the precision and accuracy of most of the machine learning algorithms tested.

Prior to training the machine learning algorithm classifiers, I scaled all features using a min-max scaler. This was vitally important, as the features had different units (e.g. # of email messages and USD) and varied significantly by several orders of magnitude. Feature-scaling ensured that for the applicable classifiers, the features would be weighted evenly.

What algorithm did you end up using? What other one(s) did you try?

The final algorithm that I ended up using was a logistic regression.The primary reason and motivation for using this was because the prediction outcomes were of binary classification (POI or non-POI). A canonical case use for logistic regression is tumor benign/malignancy prediction based off of measurements and medical test results, and I sought to emulate this behavior in predicting whether or not someone was a person of interest based off of their financial and email behavior.

I tried several algorithms, with a K-means clustering algorithm performing reasonably sufficient. K-means clustering performed well, as the objective was to segregate the dataset into POI/non-POI groups. I also tested, with marginal success, an AdaBoost classifier, a support vector machine, a random forest classifier, and stochastic gradient descent (using logistic regression).

I tuned the parameters of the logistic regression and K-means clustering classifiers using exhaustive searches with the following parameters:

- Logistic regression: `C` (inverse regularization parameter), `tol` (tolerance), and `class_weight` (over/undersampling)
- K-means clustering: `tol`

K-means clustering was initialized with K (`n_clusters`) of 2 to represent POI and non-POI clusters. Additionally, K-means performed better by including only the 8 best features and the 2 additional features.

Interestingly, the inclusion of auto-weighting in logistic regression (selection of feature weights inversely proportional to class frequencies) improved recall at the expense of precision. I opted to remove auto-weighting and maintain the evenly weighted feature-scaling discussed in the previous section.

The other algorithms were tuned experimentally, with unremarkable improvement.

What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is performed to ensure that a machine learning algorithm generalizes well. A classic mistake is over-fitting, where the model is trained and performs very well on the training dataset, but markedly worse on the cross-validation and test datasets. I utilized two methods for validating the analysis:

1. In `evalute.py`, I took the average precision and recall over 1000 randomized trials with the dataset sub-sectioned with a 3:1 training-to-test ratio.
2. In `tester.py`, I utilized the provided K-folds (k=3) evaluator to provide metrics on the classifiers.

Give at least 2 evaluation metrics, and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

The main evaluation metrics utilized were `precision` and `recall`. Precision captures the ratio of true positives to the records that are actually POIs, essentially describing how often 'false alarms' are (not) raised. Recall captures the ratio of true positives to the records flagged as POIs, which describes sensitivity. Due to the unbalanced nature of the dataset (few POIs), `accuracy` is certainly not a good metric, i.e. if 'non-POI' had been predicted for all records, an accuracy of 87.4% would have been achieved. For the logistic regression and K-means clustering algorithms, I observed conflicting results between both validation methods described in the previous section. The results are as follows:

Validation 1 (Randomized, partitioned trials, n=1,000)

Classifier	Precision	Recall	Features
Logistic Regression	0.433	0.282	12
K-means Clustering, K=2	0.396	0.318	10

Validation 2 (Stratified K-folds, K=3)

Classifier	Precision	Recall	Features
Logistic Regression	0.606	0.389	12
K-means Clustering, K=2	0.333	0.111	10

Both algorithms performed reasonably well, given that the dataset is both sparse (few POI) and noisy. Given the context of assisting and enabling securities and fraud investigators, I would argue that precision is secondary to recall. Simply put, with the objective of 'flagging' individuals for further human-led investigation, it is more important that suspect individuals are included than innocent individuals be excluded. A high recall value would ensure that truly culpable individuals were flagged as POIs and would be investigated more thoroughly.

In optimizing for higher recall value, including auto-weighting in logistic regression proved to substantially increasing recall:

Logistic Regression with Auto-weighting

	Precision	Recall
Validation 1	0.248	0.521
Validation 2	0.171	0.444

Conclusion

The most challenging aspect of this project was the sparse nature of the dataset, with very few (18) POIs. Most of the algorithms employed perform much better in balanced datasets. An interesting further study would be to employ anomaly detection algorithms (as used in other cases of fraud, like credit card checking) to identify persons of interest. Furthermore, the inclusion of temporal features may have improved the study. Many of the features were cumulative in nature, and normalizing over employment period or identifying payment 'spikes' could have increased the fidelity of the machine learning algorithms employed.

Resources and References

- Introduction to Machine Learning (Udacity)
- Machine Learning (Stanford/Coursera)
- scikit-learn Documentation

Files

- `data/`: dataset files and `pickle` objects

- `data/emails_by_address/`: email message data (not used)
- `tools/`: helper tools and functions
- `scripts/enron.py`: containes helper functions and exploratory data analysis
- `scripts/evaluate.py`: custom validation metrics
- `scripts/poi_email_addresses.py`: prints dataset email addresses (not used)
- `scripts/poi_id.py`: POI identifier
- `scripts/tester.py`: Udacity-provided file, produces relevant submission files