# Machine Learning Final Project: Identify Fraud from Enron Email

Submitted by Rebecca Mayer, 4 January 2016

# Short Questions

## (1) 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? [relevant rubric items: "data exploration", "outlier investigation"]

This project aims to identify persons of interest (POIs) in the Enron corporate fraud scandal by analyzing public data on employees, directors and associates of the company. The dataset for the project contains information on 145 individuals, of which 18 (12%) are POIs. Features include 14 financial fields based on information that was made public during the legal proceedings, along with five indicators of email communication patterns derived from the May 7, 2015 version of the Enron email corpus. The availability of the POI class labels enables the use of supervised learning techniques to carry out this classification exercise. However, the small proportion of POIs in the dataset limits the ability to subdivide the data into distinct, representative subsets for training and testing.

The data contained many outliers, which I identified using summary statistics and univariate plots. The first outlier detected was in the 'TOTAL' row. I dropped the entire row from my dataset since it was just a 'spreadsheet quirk' as noted in Lesson 7. Using the boxplot definition of outliers (values more than 1.5X the interquartile range), all 14 financial indicators contained outliers. For 12 of the 14, I decided that the statistical outliers were consistent with the underlying exponential distribution of the variables and thus should not be removed. *loan_advances* contained only three non-zero values. Rather than modifying or dropping the three values, I decided to include the feature in the Lasso regularization function to see if the whole feature would wash out (it did). *restricted_stock_deferred* contained extreme outliers at the top (BHATNAGAR SANJAY) and bottom (DERRICK JR. JAMES V) of the distribution. However, I decided not to drop those points for two reasons: 1) there was no indication that the data points were incorrect; and 2) the classification algorithms require complete observations without missing data, so to remove outliers from one category I would have had to drop those observations across all categories. I felt this would result in too great a loss of useful data. In the end, I dropped only one observation due to outliers, the TOTAL line.

The dataset also contained many missing values. For the financial data, I determined that missing values represented zero values in the original source material and recoded the missing data accordingly. For the email data, I used imputation to handle missing values, replacing missing data points with the median value. I used the median rather than the mean because the data was skewed.

## (2) 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 does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale

**behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]**

I created and tested four new features:

| Feature | Definition | Rationale |
|---|---|---|
| *email_available* | dummy variable indicating whether the individual's email data is included in the dataset | see whether the large number of missing email indicators (59/145) was random or meaningful in terms of POI identification |
| *from_poi* | proportion of person's sent mail that was sent to a POI | use ratio to improve comparability |
| *to_poi* | proportion of person's received mail that was received from a POI | use ratio to improve comparability |

| Feature | Definition | Rationale |
|---|---|---|
| *shared_poi* | proportion of person's received mail that had a shared receipt with poi | use ratio to improve comparability |

After adding the new features to the dataset and removing the original email indicators, I used sklearn's Lasso function to decide which features to use in my POI identifier. Lasso is a variant of least squares regression that uses a penalty factor to draw the least contributing predictors towards a coefficient of zero. In order to avoid distortion of coefficient values due to features with widely differing scales, I used sklearn's StandardScaler to scale and center the features.

Running Lasso requires tuning of the alpha parameter, which modifies the size of the penalty factor in the minimization function. If alpha is set to 0, the sklearn Lasso function is equivalent to ordinary least squares. Using trial and error with different alpha values, I generated five different feature sets and evaluated their impact on the performance of an SVM model. The results of testing different feature sets based on different alphas were as follows:

| alpha | F1 score |
|---|---|
| 0.015 | 0.46311 |
| 0.010 | 0.40230 |
| 0.012 | 0.39464 |

| alpha | F1 score |
|---|---|
| 0.040 | 0.34230 |
| 0.025 | 0.26779 |

The parameter setting of alpha=0.015 produced the highest F1 score, so that was the setting that I used in the final model. The features that I retained were those that produced a Lasso regression coefficient greater than 0.001, as shown in the table below.

| Lasso coefficient | feature |
|---|---|
| 0.037193 | bonus |
| -0.054944 | deferral_payments |
| -0.060743 | deferred_income |
| -0.000000 | director_fees |
| 0.072078 | exercised_stock_options |
| 0.024956 | expenses |
| -0.000000 | loan_advances |
| 0.000000 | long_term_incentive |
| -0.000000 | other |
| 0.000000 | restricted_stock |

| **Lasso coefficient** | **feature** |
|---|---|
| 0.000000 | restricted_stock_deferred |
| 0.009133 | salary |
| -0.000000 | total_payments |
| 0.000000 | total_stock_value |
| 0.000000 | email_available |
| 0.069353 | from_poi |
| 0.000000 | to_poi |
| 0.000000 | shared_poi |

Following Lasso, three of the four new features that I created were eliminated from the final analysis. Only *from_poi* produced a non-zero coefficient large enough to pass the threshold value of 0.001. The final SVM model using the modified email features produced an F1 score of 0.46, and achieved precision and recall scores above the minimum threshold of 0.3. The original features resulted in poorer model performance, with an F1 score of 0.41 and a recall score of 0.288, which was below the minimum threshold. Based on these results I determined that the new features improved final algorithm performance.

The final features included in the POI identifier were: ['bonus', 'deferral_payments', 'deferred_income', 'exercised_stock_options', 'expenses', 'salary', 'from_poi'].

## (3) What algorithm did you end up using? What other one(s) did you try? How did model

**performance differ between algorithms? [relevant rubric item: “pick an algorithm”]**

I tested five different classification algorithms: GaussianNB, SVM, Decision Tree, Random Forest and K Nearest Neighbors. SVM and K Nearest Neighbors were the best performers, achieving F1 scores of 0.46 and 0.44, respectively. GaussianNB was able to meet the minimum precision and recall values of 0.3, but produced a lower F1 score of 0.37. Decision Tree and Random Forest failed to achieve the minimum required precision and recall levels of 0.3. In the final analysis I used the best performer, SVM.

## (4) What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: “tune the algorithm”]

To tune an algorithm is to adjust its performance by specifying the values of input parameters that are allowed to vary. If the tuning is not optimized, the algorithm will not achieve the best performance that it is capable of reaching. For this exercise, I used GridSearchCV to automatically select the best parameters from among a range of values based on F1 score.

The SVM parameter values that I evaluated using GridSearchCV were:

- 'kernel': ('linear', 'rbf', 'poly')
- 'C': [1, 10, 100]

- 'gamma': [0.125, 1, 10]
- 'degree': [4, 5, 6, 7, 8]

After fitting the grid search algorithm, I printed the best model parameters using the GridSearchCV best_estimator_ attribute and copied them into my SVM object.

## (5) What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: “validation strategy”]

Validation is the process of evaluating how good your model is, such as by testing how well the model generalizes to independent datasets. One classic mistake would be to use the same data for training and testing. If you do this, there is a good chance your model will be overfit to the data and will fail to perform as well as expected on independent datasets. If your data is transformed before running the classification algorithm, for example using PCA, you also want to avoid the mistake of refitting your test data before running the PCA transformation. You need to transform your data using the transformation parameters generated from the test data.

I used sklearn's train_test_split function to create a separate training set for validating model performance while tuning the parameters of my Lasso and GridSearch functions. During this process, I ran Lasso on the entire dataset to select my features, used GridSearchCV on the training data to set the parameters of my SVM model, and then tested SVM model performance using the instructor-provided tester.py functions. Because the tester.py function used its own validation approach based on StratifiedShuffleSplit, I did not use the exact test data that I created using train_test_split to validate the model. However, the fact that I preserved some independent data for testing during the parameter selection process should still have provided some buffer against overfitting.

## (6) 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. [relevant rubric item: “usage of evaluation metrics”]

My SVM model achieved an average precision score of 0.41, indicating that about 4 in 10 predicted POIs were correct. The average recall score was 0.53, meaning that about half of all the POIs in the dataset were correctly identified.