# Project 5: Identify Fraud from Enron Email

By Jacob Kreisler

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?**

   The goal of this project is to identify patterns in a dataset of Enron employees so that we can predict persons of interest, or POIs, in the fraud case. With machine learning, we can build powerful algorithms that take in data features and return predictions.

   The dataset includes information on 145 Enron employees, 18 of them actual POIs. Each employee has 21 features that describe their financial and email activity. In some cases, the value for an employee's feature is labelled as "NaN." For example, the salary feature is only given for 95 of the 145 employees, the rest having "NaN" as its value.

   By charting the data, an outlier is obvious. Diving into the data, we can see that one data point, "TOTAL," sums the total values of each feature, so we remove this from our data using data_dict.pop('TOTAL', 0)

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"]**

   To select my features, I use a decision tree ensemble, Extra Trees, to provide me with feature importance. I use this to help in removing or adding certain features that will increase overall performance, especially accuracy, precision, and recall. I try to leave as few as possible while balancing strong performance metrics. These features are: salary, total payments, exercised stock options, bonus, total stock value, expenses, loan advances, deferred income, long term incentives, and to ratio, a feature I created that takes the proportion of emails an employee sent to a POI against the total emails he sent. The table below shows feature importance:

| | |
|---|---|
| Salary | 0.08352988 |
| Total Payments | 0.09144073 |
| Exercised Stock Options | 0.09977992 |
| To Ratio | 0.09984691 |
| **Bonus** | **0.14505672** |
| Total Stock Value | 0.08045897 |
| Expenses | 0.11699462 |
| Loan Advances | 0.01224601 |
| Deferred Income | 0.13952073 |
| Long Term Incentive | 0.13112552 |

The “To Ratio” feature was created to show patterns between employees that had more relative communication with POIs. By dividing the number of emails an employee sends to a POI, over the total number of emails he sends, we can get a sense of just how much that person was speaking to flagged employees and could serve as an effective predictor.

Feature scaling was unnecessary since the unit being observed in all features is the dollar, so we are able to compare effectively. For the “to ratio” feature, since we are comparing employees based on their relative email activity, the feature is already scaled. I actually created one more feature, a proportion of total payments over total stock value, but testing this feature greatly reduced performance and so was omitted. Here are the precision and recall results with the added “To Ratio” feature:

| | Precision | Recall |
|---|---|---|
| Without added feature | 0.48880 | 0.34900 |
| With feature | **0.5325** | **0.3605** |

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”]**

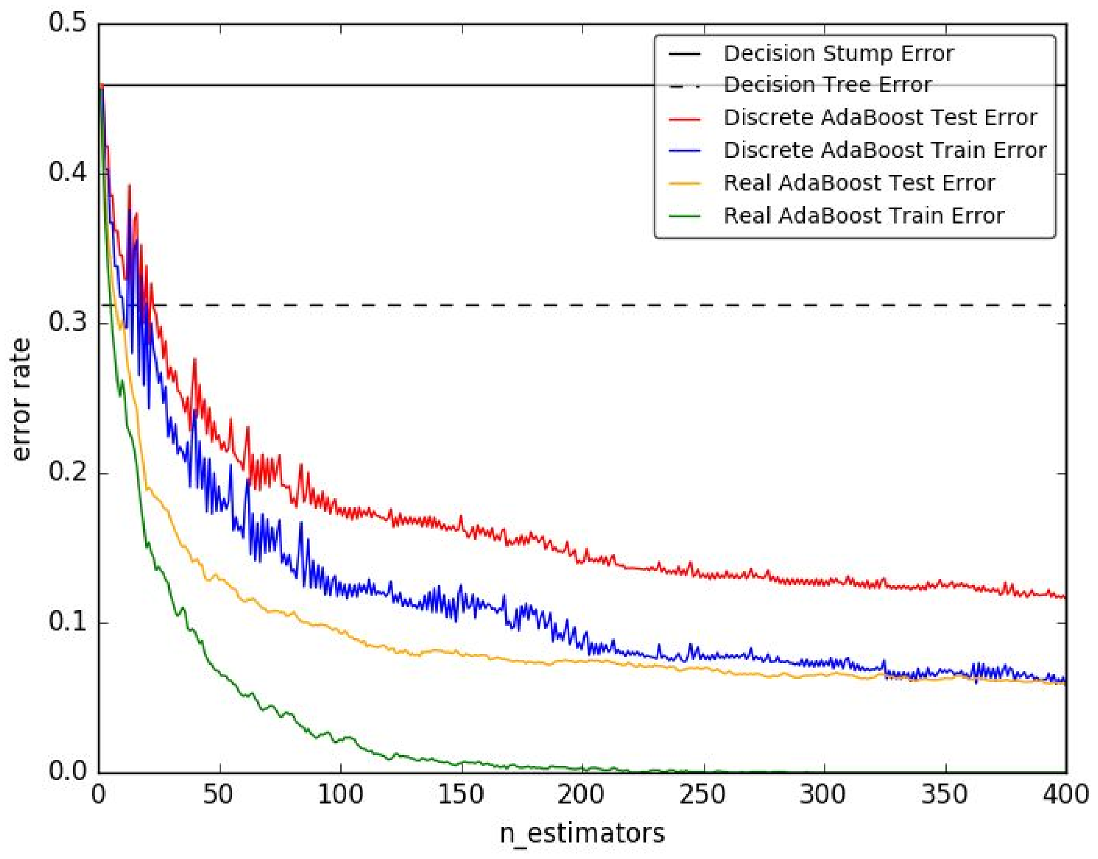
   I decided to use the AdaBoost classifier with the n_estimators=100 parameter. I used several other algorithms to compare performance, including KNeighbors, SVC, GaussianNB, DecisionTree, and RandomForest. These algorithms generally worked well, however, it was AdaBoost that ultimately gave me the best performance.

| | KNeighbors | AdaBoost | GaussianNB | DecisionTree | RandomForest |
|---|---|---|---|---|---|
| Precision | 0.45962 | 0.5325 | 0.34377 | 0.31083 | 0.46667 |
| Recall | 0.23900 | 0.3605 | 0.31850 | 0.30850 | 0.16450 |

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”]**

   When you tune the parameters of an algorithm, you are customizing how that particular algorithm runs. While each algorithm is different, the wrong parameters could mean poor performance results from overfitting or overgeneralizing the data. I used GridSearchCV to help tune the AdaBoost with the n_estimators parameter. The parameter n_estimators tells the algorithm how many decision trees to use when combining to form a prediction. By increasing the amount of decision trees, we are able to make better predictions with the data. The default value for n_estimators is 50, I used GridSearchCV to input a range of n_estimator values from 50 to 400 to find the optimal parameter. This returned a value of 250, as that seems to be the place at which the balance between better performance and overfitting is maximized.

| | N = 50 | N = 150 | **N = 250** | N = 350 | N = 400 |
|---|---|---|---|---|---|
| Precision | 0.46784 | 0.51597 | **0.5325** | 0.53702 | 0.53463 |
| Recall | 0.3455 | 0.3555 | **0.3605** | 0.3735 | 0.3705 |



http://scikit-learn.org/stable/_images/sphx_glr_plot_adaboost_hastie_10_2_0011.png

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 in which we compare our training data to our test data. We do this so that we can estimate our performance on an independent dataset, or many independent datasets, to be sure that we are not overfitting the data and making assumptions that don't actually exist based off of our training data. I used a train/test split with a parameter of test_size = 0.25 for my analysis.

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"]**

Two evaluation metrics to quantify my performance are precision and recall. Precision measures how well the algorithm observes a certain class that actually is that class. So, in this project, precision scores how well the algorithm is at tagging POIs. My precision score is 0.51737, so if my algorithm selects 10 employees as being POIs, about 50% of them will actually be POIs. Recall would quantify how well the algorithm did at picking up all instances of a certain class. So, in this project, my recall of 0.35750 means that since there 18 total POIs, my algorithm would tag about 36% of them, say 6 of the 18 POIs.