Machine Learning: Identifying Fraud from Enron Email

By Carvus Byrd

Overview

Enron Corporation, which was one of the largest companies in the United States in 2000, was an American energy, commodities, and Services Company. In 2002, Enron had filed for bankruptcy because of accounting fraud. In the following federal investigation, a large database of over 600,000 emails generated by 158 employees of Enron was acquired. Subsequently, a copy of the database was purchased and released to public by Andrew McCallum. The dataset was used widely for machine learning studies. The project below outlines my methods for using "scikit-learn", a python library, to produce multiple algorithms to predict a "person of interest", or "POI", who may have been involved in the fraud at Enron.

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.

The goal of this project is to build a predictive model that can help identify "persons of interest" (or POI) with the Enron fraud scandal of the early 2000's. To create the predictive model, we are using the Enron email corpus, which is made up of both financial and email data from the actual company records. Using both sets of data, we can train machine learning algorithms to help determine POI's based on employees populated with the dataset.

The information we are concerned about in this case is somewhat summarized to include 19 features (20 if you include "email_address") to describe employees, their compensation, and how many emails they either sent or received from POI's. We concentrate our focus towards 146 employees, 18 of which are actually POIs.

It is also important to understand the data that is not always present for each feature and employee. I've used Pandas to help organize, analyze, and preprocess the information. The first table shows missing data by feature.

Table 1

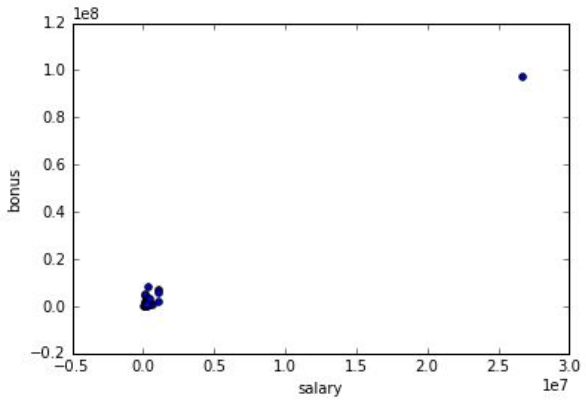
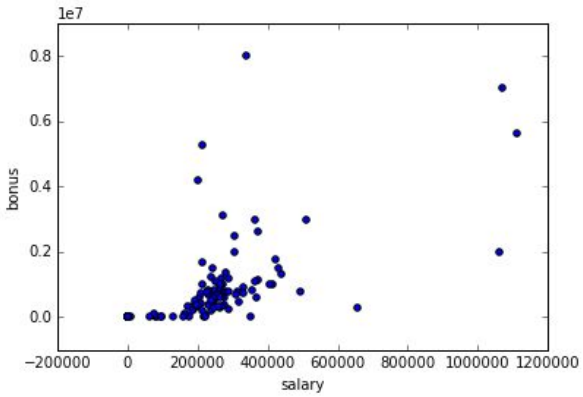
Count of Missing Values by Feature:	
loan_advances	**142**
director_fees	**129**

restricted_stock_deferred	**128**
deferral_payments	**107**
deferred_income	**97**
long_term_incentive	**80**
bonus	**64**
from_poi_to_this_person	**60**
to_messages	**60**
from_this_person_to_poi	**60**
from_messages	**60**
shared_receipt_with_poi	**60**
other	**53**
salary	**51**
expenses	**51**
exercised_stock_options	**44**
restricted_stock	**36**
total_payments	**21**
total_stock_value	**20**
poi	**0**

It's clear that some variables, like "loan_advances", does not have much data (only 4 data points as it is missing 142 and there are only 146 as mentioned).

Were there any outliers in the data when you got it, and how did you handle those?

Upon initial inspection we find a couple of obvious outliers that should likely be removed. The first point is that of a "Total" field. It was likely created when the information was summarized but was easy to spot when one graphs "salary" vs. "bonus" features (Both before removal and after removal graphs are below). This field was removed since it is not appropriate for it to remain.

Original Data	After removal of “Total” data point
	

The second was that of “THE TRAVEL AGENCY IN THE PARK” which does not seem to be a real name. This point was found when I reviewed the names associated with each data point. This field was also removed as I believed it was not a real name and therefore, not relevant.

The third outlier is “LOCKHART EUGENE E” as he does not seem to have any data associated with the variables we are considering for this project. The table below is output from a query to count missing values across features (It is condensed only to show the employees that are missing 16 or more features). Given that we only have 19 features, it’s obvious that Eugene doesn’t have any data points to consider. I removed this variable to keep the data as clean and neat as I could.

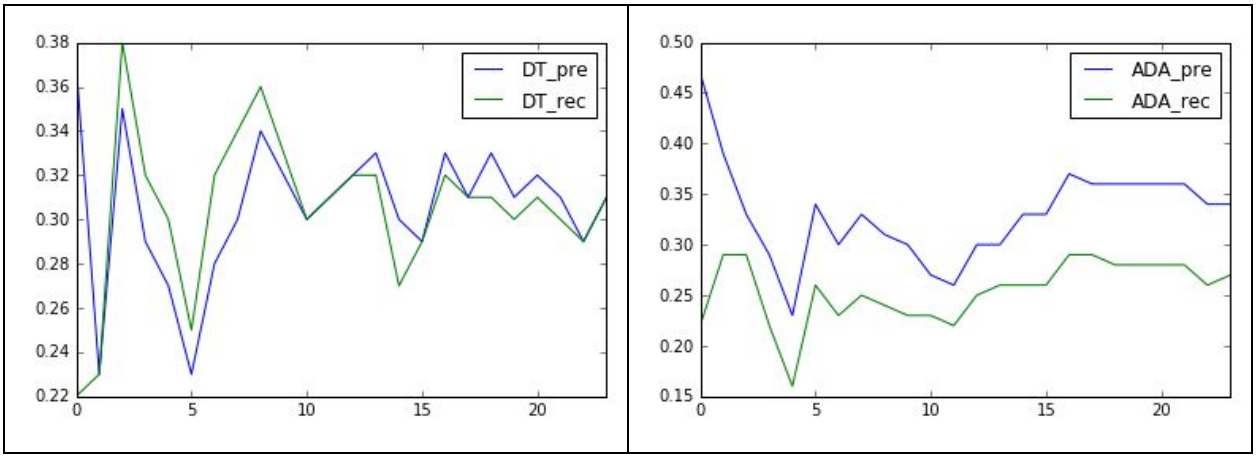
Count of Missing Values by Employees:	
LOCKHART EUGENE E	19
GRAMM WENDY L	17
WROBEL BRUCE	17
WODRASKA JOHN	17
THE TRAVEL AGENCY IN THE PARK	17
WHALEY DAVID A	17
SCRIMSHAW MATTHEW	17
CHRISTODOULOU DIOMEDES	16

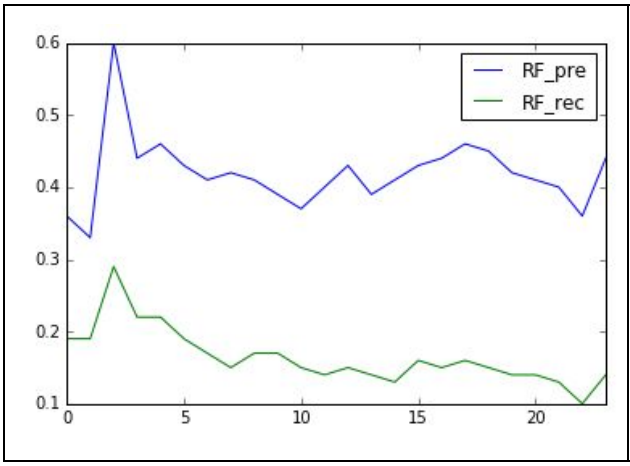
CLINE KENNETH W	16
GILLIS JOHN	16
SAVAGE FRANK	16
WAKEHAM JOHN	16

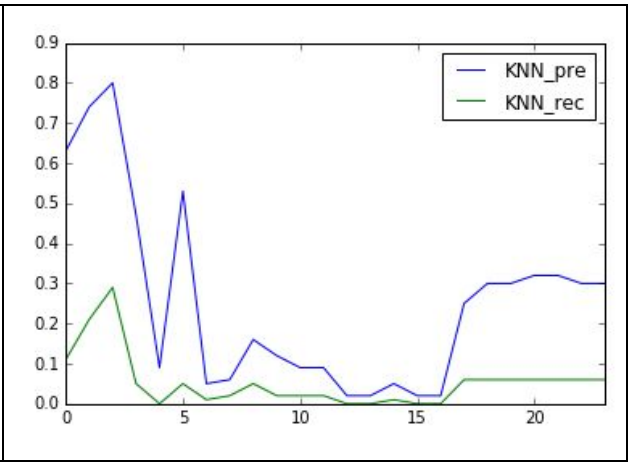
I'll note that we are now down to 143 true, non-outlier, employees, 18 of which are actually POIs.

What features did you end up using in your POI identifier, and what selection process did you use to pick them?

Initially, I wanted to use all 19 features (plus the others I created, more on that below) as I thought this would provide the best return of information. However, I decided to let python and SelectKBest do this work for me by iterating through each classifier, while also cross validating with "StratifiedShuffleSplit", and measuring precision and recall at each "k" level to determine which features to use. For the four algorithms I used, I plotted the precision and recall at each level (below), this should help me determine how many and which features to use for each model.







I'll detail more on my reasoning later but I'm looking for the K value that maximizes my recall. For each model, I've outlined which features provided this maximum recall.

Model	K	Features Used	Precision	Recall
Decision Tree	3	bonus restricted_stock_deferred exercised_stock_options	0.35	0.38
AdaBoost	2	bonus exercised_stock_options	0.39	0.29
Random Forest	3	bonus restricted_stock_deferred exercised_stock_options	0.60	0.29
K-Nearest Neighbors	3	bonus restricted_stock_deferred exercised_stock_options	0.80	0.29

Did you have to do any scaling? Why or why not?

Yes, scaling was performed in the analysis, using the MinMaxScaler module. It is used because the features in this dataset have unique ranges (i.e. Some features deal with monetary values that go into the tens of millions of US dollars while a few features are percentages that only range from zero to one). Also, K-Nearest Neighbors is one of the algorithms that works much better when the data is scaled.

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.

I was able to engineer five new features. The first was a grand total of money being paid to the employees, “Total_Compensation”. I thought this would help summarize the amount of money being paid towards employees. The lessons and mini-projects led me to believe that POI’s generally get paid a lot more than non-POI’s so I wanted to further explore that theory.

The second and third created features explore the percentage of emails sent to and received from POI’s. These features should present a better picture of how often employees correspond, via email, with POI’s. I would hypothesize that the higher the percentage of emails sent to or received from a POI, then the more likely he or she is also a POI (i.e. guilty by association).

The fourth created feature, Retention Incentives, helped show reasons why a person would like the company to continue to do well. It is made up of the sum of “long_term_incentive” and “total_stock_value”. I was hoping this variable my point persons with large stake in the future of Enron.

The engineered feature is a ratio of “salary” to “bonus”, called “Salary_Bonus_Ratio”. I hope this feature helps show differences among the two variables. I have a theory that someone committing fraud would get a bonus unbefitting of their salary.

I will note that my features were not used in the modeling as they were not selected by SelectKBest. Below, I’ve provided a table to show their scoring relative to the rest of the variables. It appears that if the model had chosen a K value of maybe four, five, or six, some of the engineered variables would have utilized within the models.

SelectKBest Output
salary score is: 18.2896840434
to_messages score is: 1.64634112944
deferral_payments score is: 0.224611274736
total_payments score is: 8.77277773009
exercised_stock_options score is: 24.8150797332
bonus score is: 20.7922520472
restricted_stock score is: 9.21281062198
shared_receipt_with_poi score is: 8.58942073168
restricted_stock_deferred score is: 0.0654996529099
total_stock_value score is: 24.1828986786
expenses score is: 6.09417331064
loan_advances score is: 7.18405565829
from_messages score is: 0.169700947622
other score is: 4.187477507
from_this_person_to_poi score is: 2.38261210823
director_fees score is: 2.12632780201
deferred_income score is: 11.4584765793
long_term_incentive score is: 9.92218601319
from_poi_to_this_person score is: 5.24344971337
percentage_from_poi score is: 3.12809174816
percentage_to_poi score is: 16.409712548

total_compensation score is: 17.8087911742
retention_incentives score is: 18.0020883914
salary_bonus_ratio score is: 10.7835847082

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

In all, I tried K-Nearest Neighbors, Random Forest, Adaboost, and Decision Tree. I settled on Random Forest as it had the larger Recall score (I find this variable extremely important in this situation (more explanation on this reasoning is below) and secondly, it had the largest Precision score as well. The other models scored similar (as in the table above) in fact, the initial models (without tuning) had the Decision Tree model as the highest Recall score model but "fine-tuning" propelled Random Forest to be the model of choice.

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?

Most algorithms have various parameters that allow a user the opportunity to "tune" to provide better performance and prediction capabilities. Sometimes the tuning doesn't help performance but other times it can have a drastic effect on the performance. Because of this, it is appropriate to find the most optimal values for important parameters within an algorithm. I will warn that you should pick your parameters carefully as iterating through n_clusters on the K-Means algorithm will produce incorrect results.

I tuned all algorithms using GridSearch. For each algorithm, I was able to identify a few parameters to tune, continue using cross validation, and labeled "Recall" to be the scoring metric that should be maximized. A good example is my Decision Tree Classifier, it had the parameters below tuned via GridSearchCV.

Initial Parameter Options	Final Tuned Parameters
parameters = {'criterion': ['gini', 'entropy'], 'min_samples_split': [2, 6, 8, 10, 15, 20, 30], 'max_depth': [2, 6, 8, 10, 15, 20, 30]}	DecisionTreeClassifier(class_weight=None, **criterion='gini'**, **max_depth=8**, max_features=None, max_leaf_nodes=None, min_samples_leaf=1, **min_samples_split=2**, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

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 the process of measuring how well our chosen algorithm captures and predicts new information beyond the dataset used to train it. It is meant to prevent a classic mistake of overfitting data. Overfitting occurs when the same dataset is used for both training and testing.

My validation methods consisted of two parts. The first is that I ran "StratifiedShuffleSplit" to produce 1,000 samples of training data and testing data from my original training data. The neat thing about each sample is that it will retain the original class integrity (POIs vs Non-POIs). This allows each sample to be a helpful learning tool for the algorithms. The results for this procedure are above. I was also able to run the "test_classifier" function for each algorithm variation to produce true "test" metrics for accuracy, precision, and recall. The results for this procedure are below.

Model	Precision	Recall
Decision Tree	0.365	0.381
AdaBoost	0.358	0.350
Random Forest	0.567	0.403
K-Nearest Neighbors	0.497	0.393

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 most important evaluation metrics utilized in this project were precision and recall. Precision indicates the percentage of positive predictions were correct (i.e. How many predicted POIs were actually POIs?). In this model, my final algorithm, Random Forest was able to produce a precision score of .342. This is made up of the 911 true positives over the sum of all positive predictions (911 + 1755). Recall indicates the percentage of positive cases did the model predict to be positive (i.e. How many actual POIs were predicted to be POIs?). In this model, my final algorithm was able to produce a recall score of .456. This is made up of the 911 true positives over the sum of true positives and false negatives (911 + 1089).

Given the context of assisting fraud investigators, it is easy to see that precision is secondary to recall. Put another way, with the goal of identifying employees for further human-led investigation, it is more important that suspect employees are investigated than innocent individuals be removed from consideration. A high recall value would help make truly culpable employees flagged as POIs and would be investigated more thoroughly.

Conclusion

For this project, I used the Random Forest algorithm to create a predictive model to identify employees who may have been involved in fraud at Enron during the early 2000s. The model is not great but I believe could at least help investigators narrow down their search.

References

Introduction to Machine Learning (Udacity)

scikit-learn: machine learning in Python

Enron - Wikipedia

Enron Corpus - Wikipedia

Precision and Recall - Wikipedia

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.