Identifying Fraud from Enron Email

Intro to Machine Learning Final Project – Udacity

Jason Bowles

Introduction

For me this project was about identifying potential 'persons of interest' among the employees at Enron. In the early 2000's Enron was an energy trader that went bankrupt because of massive corporate scandals that saw multiple arrests among their leadership and employees. The goal is to identify those who should be investigated even further, and not let anyone slip through the cracks. The flip side is that we should do our best to reduce false positives so that the investigation doesn't end up being a large portion of the company.

One of the tools at hand to accomplish our goal is a massive dataset of emails from people who worked at Enron, around 20,000 emails in total. We also have information gathered on 146 people that includes financial information, such as; salary, bonus's, expenses, counts of emails to other person's of interest and various stock information which was a part of their total compensation package. With this information at our fingertips we'll use various data analysis and machine learning techniques to attempt to prep the data and train algorithms on how to identify a 'person of interest'.

About the Data

Of the 146 individuals for which we have data, 18 of them have been identified as a 'person of interest' or poi. We have email history for all but 60 of the 146, which includes 4 poi for which we don't have emails. The information that we have for each includes 26 different pieces of data. Those pieces are summarized below with the number of valid values we have for that data and a min and max value.

```
FEATURE: to_messages              (VALID VALUES: 86 , MIN: 0                   , MAX: 15149)
FEATURE: deferral_payments        (VALID VALUES: 39 , MIN: -102500             , MAX: 32083396)
FEATURE: expenses                 (VALID VALUES: 95 , MIN: 0                   , MAX: 5235198)
FEATURE: poi                      (VALID VALUES: 146, MIN: 0                   , MAX: True)
FEATURE: long_term_incentive      (VALID VALUES: 66 , MIN: 0                   , MAX: 48521928)
FEATURE: email_address            (VALID VALUES: 111, MIN: 0                   , MAX: wes.colwell@enron.com)
FEATURE: deferred_income          (VALID VALUES: 49 , MIN: -27992891           , MAX: 0)
FEATURE: restricted_stock_deferred (VALID VALUES: 18 , MIN: -7576788            , MAX: 15456290)
FEATURE: shared_receipt_with_poi  (VALID VALUES: 86 , MIN: 0                   , MAX: 5521)
FEATURE: loan_advances            (VALID VALUES: 4  , MIN: 0                   , MAX: 83925000)
FEATURE: from_messages            (VALID VALUES: 86 , MIN: 0                   , MAX: 14368)
FEATURE: other                    (VALID VALUES: 93 , MIN: 0                   , MAX: 42667589)
FEATURE: combined                 (VALID VALUES: 146, MIN: 0                   , MAX: 1180207944.0)
FEATURE: to_poi_ratio             (VALID VALUES: 86 , MIN: 0                   , MAX: 1.0)
FEATURE: director_fees            (VALID VALUES: 17 , MIN: 0                   , MAX: 1398517)
FEATURE: bonus                    (VALID VALUES: 82 , MIN: 0                   , MAX: 97343619)
FEATURE: total_stock_value        (VALID VALUES: 126, MIN: -44093              , MAX: 434509511)
FEATURE: from_poi_to_this_person  (VALID VALUES: 86 , MIN: 0                   , MAX: 528)
FEATURE: from_this_person_to_poi  (VALID VALUES: 86 , MIN: 0                   , MAX: 609)
FEATURE: restricted_stock         (VALID VALUES: 110, MIN: -2604490            , MAX: 130322299)
FEATURE: salary                   (VALID VALUES: 95 , MIN: 0                   , MAX: 26704229)
FEATURE: total_payments           (VALID VALUES: 125, MIN: 0                   , MAX: 309886585)
FEATURE: email_body               (VALID VALUES: 86 , MIN: -0.0204070764205    , MAX: 0.0139732859815)
FEATURE: exercised_stock_options  (VALID VALUES: 102, MIN: 0                   , MAX: 311764000)
FEATURE: email_subject            (VALID VALUES: 86 , MIN: -0.00959976227841   , MAX: 0.00568118369682)
```

As you can see in this summary of values for the data we have that there are some very large values and several pieces of data (features), which have missing values (example: deferred_income has 97 missing values). As far as outliers I found that the leaders at Enron made a great deal more than others but first I needed to remove the “TOTAL” record from the dictionary. After that I attempted to remove those individuals who made substantially more than others. I started by removing the top 10% and worked my way down to 1%, but each one still had a negative effect on my final algorithm and I decided to not do any further outlier treatment outside of removing the “TOTAL” record (which was basically a total row on a report).

Model Variables/Feature Selection

Feature Engineering

For the final model, I settled on the following features (or variables);

email_subject, to_poi_ratio, combined, from_messages, expenses, deferred_income,other, restricted_stock, long_term_incentive, deferral_payments, email_body, restricted_stock_deferred

Of these variables, the following were engineered and not part of the given dataset; to_poi_ratio, combined, email_subject, email_body. To settle on these variables I first did some exploratory data analysis to see if I can find some commonalities between poi and non-poi. What I noticed, and from a little intuition is that compensation seemed to play a part in the role of a poi. But instead of trying to pull in every compensation piece of data I did a combination of all variables that appeared to have a role in the overall amount of money owed to an individual. The following fields were totaled together to form a new field called “combined”; salary, bonus, total_stock_value, total_payments, exercised_stock_options. The key here is that if the field did not exist in the dataset and was set to the value “NaN”, I defaulted the value to zero instead of excluding that row. (My thinking was that it makes sense that not every employee may have stock options or received a bonus). The one exception to this rule is “salary”, If I did not have salary for the employee then I didn’t believe it would create a good metric, so rows/individuals that did not have a salary in the data were set to “NaN” (Not a Number) for the combined variable. There were other monetary type fields included in the final dataset, but were not included in combined. I did not feel that these fields should be lumped into the new “combined”, because they were less directly related to the person’s total compensation package.

I also spent quite a bit of time implementing some text learning as the remaining fields to be used in the final model. Working with a co-worker I learned a technique of adding text learning to non-text features. This method is to build your text documents and create the "TfidfVectorizer”, then using any support vector machine as the algorithm, extract the decision function and use that value as part of the final model. This value will indicate the strength of the text (the larger number means a higher likelihood that the individual is a poi). I did this in the for the final model by parsing 20,000 emails for the 86 individuals in the final dataset and creating an email_body value and email_subject value for each person. The email subject was extracted by using a regex function to parse the email header. My thought was that often I have sent and received emails where the subject contains as much or more content than the email body. This seemed like an opportunity to use that to our advantage in the model. For the LinearSVC model I used to score the text, I build in a recursive function to eliminate the heighest weighted words from the documents. In this portion, I focused more highly on the recall than the precision to make sure that the value returned from text learning to the final model did not act against the other variables and falsely exclude a poi in the final classifier.

Feature Scaling

Of the final variables only 1 variable was ultimately scaled. This was the to_poi_ratio engineered variable. From my analysis of the emails I noticed that the range of total emails sent by the individuals in the dataset, was fairly large. Which means that a person who sent over 1000 total emails with 100 of those sent to a poi, would be weighted the same as an individual who only sent 200 total emails with 100 of them being sent to a poi. I felt that we needed to think about the ratio of total emails sent to a poi instead of the raw volume. Which essentially scaled the from_this_person_to_poi feature included in the final dataset (from_this_person_to_poi/from_messages).

Feature Selection

The final features were settled on by comparing the weight of each feature and removing the lowest weighted feature and re-running the model and checking for score improvement. Below is the weights for each variable in the initial run, followed by the weights of each variable in the final run.

Initial Run

```
The weight of email_subject, is: 0.069323494581
The weight of email_body, is: 0.00503245044456
The weight of to_poi_ratio, is: 0.0217134189531
The weight of combined, is: 0.0725195293707
The weight of from_messages, is: 0.0114867893844
The weight of expenses, is: 0.0463848226247
The weight of deferral_payments, is: 0.00615401923947
The weight of deferred_income, is: 0.0236568462806
The weight of director_fees, is: 0.0
The weight of loan_advances, is: 0.0
The weight of long_term_incentive, is: 0.00631815047869
The weight of other, is: 0.0671871992371
The weight of restricted_stock, is: 0.0372227805908
The weight of restricted_stock_deferred, is: 0.00800049881488
The weight of to_messages, is: 0.0
```

Final Run (format changed so that it could be sorted more easily)

```
0.069323494581 is the weight of email_subject
0.0217134189531 is the weight of to_poi_ratio
0.0725195293707 is the weight of combined
0.0114867893844 is the weight of from_messages
0.0463848226247 is the weight of expenses
0.0236568462806 is the weight of deferred_income
0.0671871992371 is the weight of other
0.0372227805908 is the weight of restricted_stock
0.00631815047869 is the weight of long_term_incentive
0.00615401923947 is the weight of deferral_payments
0.00503245044456 is the weight of email_body
0.00800049881488 is the weight of restricted_stock_deferred
```

I did try SelectKBest and PCA with little success, (the final model score ended up lower). However I do like the power of these tools to get you started in the analysis and tuning of an algorithm. Another method of feature selection would be to build in a recursive function as I did with text learning section of the model. In the code there I printed out the removed features and stored the values in an array to save time on future runs.

Algorithm Selection

I tried several different algorithms for my final model and the text learning piece of this code. Ultimately I settled on using LinearSVC for the text learning portion and GradientBoostingClassifier for the final model.

Text Learning Model

In the text learning portion of the model I started with a Decision Tree Classifier and implemented a recursive function to select off the highest weight if it was scored as .2 or higher. The problem was that the Decision Tree kept only doing 1 split and scoring 1 feature at 1.0 and the rest at 0.0, the recursive function ran for several hours and never fully completed. I tried several tuning techniques like adjusting min_samples_split and min_samples_leaf which had little effect. I almost gave up trying to add in a text learning component because it was taking so long to complete and returning an extremely low recall score. (final score is unavailable because I was too impatient to let it complete). As a last ditch effort I tried the LinearSVC and initially got the same results as the decision tree. However once I adjusted the "C" parameter (the penalty parameter), down iteratively down from 1000 to .001 I got dramatically better results. The final results are as follows:

Email Subject: Accuracy = .6227, Precision = .1393 and Recall = .7004

Email Body: Accuracy = .8202, Precision = .2924 and Recall = .8480

With my goal for text learning being to get the recall number as high as possible without too low precision, so as not to disqualify somebody too soon before the final model.

As I mentioned above, I used the decision_function results to pull back into the dataset as a point for each employee.

Final Model

For the final model that pulled in the remaining variables and the text learning output I tried several different algorithms, mostly because I was curious as to the result of each.

For the testing period of models I tried to stick with a small number of features to test before trying more features in the final algorithm. This allowed me to run and re-run the algorithms more quickly when tuning and testing. The test variables I used were: email_subject, email_body, to_poi_ratio, and combined.

Here is summary of the models that I tried and their average scores:

GridSearchCV (with LinearSVC and trying several different C values [0.001, 1, 0.01, 10].

Accuracy:	.38
Precision:	.15
Recall:	.78
F1:	.30

LinearSVC (runs = 2)

Accuracy:	0.677
Precision:	0.20027
Recall:	0.47
F1:	0.29833
F2:	0.42659

AdaBoost (runs = 11)

Accuracy	0.839654545
Precision	0.444775455
Recall:	0.371227273
F1:	0.398323636
F2	0.380021818

GradientBoosting (runs = 14)

Accuracy	0.87229
Precision	0.528802143
Recall	0.401
F1	0.455735
F2	0.421167857

You can see that the best performance was from GradientBoosting. Which I found by doing some research on ensemble methods. What I found was that it was similar to AdaBoost, which I used a Decision Tree as the base classifier.

Algorithm Tuning

Most algorithms for machine learning do not come out of the box ready to go. Most have several parameters that can be passed in that change default behavior. For example with a Decision Tree, how many values of a feature must be present before making a split, and how many total splits should be made? These parameters help in tuning the bias and variance trade off. Making sure to allow the model to pick on the importance of a feature (under-fitting the model) and guard against the model putting too much value on a particular feature (over-fitting the model).

With the final model I did the majority of my tuning with the learning_rate, as I found this parameter had more of an effect on the performance of the model than other parameters such as; n_estimators, max_depth and min_samples_split. The values I tried were (1, .75, .5, and .25). I found that as the learning_rate was set higher the precision was lower and the recall was higher, but on the lower values the precision was higher and the recall lower. I settled on using a learning_rate of .5 for the final model.

For AdaBoost, I focused mainly on tuning the max_depth parameter, which I found that the higher number used for max_depth of nodes the more over-fitting the model tended to have. With LinearSVC I used the GridSearchCV to try several different values for the C parameter.

Algorithm tuning is a little art and a little brute force (trying as many combinations as possible). With every model the parameters effect the output differently and as a data analyst you must try all of the different combinations to understand how changing the parameter has an effect on the performance.

Validation Strategy

The building of the final model involved separating the dataset into training and test datasets. This means that we hold part of the data out of the training routine for the model. Then use this held out section of the dataset to validate the model. This helps to ensure that the model isn't under or over-fitted.

The way in which the dataset is split is important because without making sure that any point in the dataset has an equal chance of being part of the training set and the test set, we are open to bias in the final model. Cross validation is the strategy that was employed for this model, which helps guard against unintentionally over-fitting to the test set. The method used is called StratifiedShuffleSplits, which mixes up the data and created 1000 different sets of the data, (this number is user defined). This method also ensures the correct percentage of labels in the test and training sets.

Evaluation Metrics

For this model I focused on the precision, recall and F1 scores as I tested features and tuned the model. I felt that accuracy in this case was a poor choice because by simply marking no one as a "poi" the accuracy score. The precision measured what percentage of all employees the model identified as a "poi" were actually poi's, where recall measured the percentage of poi's identified by the model versus all poi's in the dataset. The F1 score is the combination of the precision and recall, which help you understand an overall perspective of the performance of the model.

The GradientBoosting algorithm had the best performance and with the 14 different runs the average scores for precision, recall and F1 were .529, .401 and .456 respectively. The goal of this project was to achieve a precision and recall score above .3, which my model has performed. All though my score surpassed the requirement I feel a better scored could have been achieved with more experience and the introduction of more features. Another possible feature added could be the text learning in accordance to stock movement. The people at enron were highly motivated by the stock price of their company (My understanding is the previous day's closing price was posted for all to see in the company). Therefore I would think that there may be some emails flying around (especially poi's) about the stock on particularly good or bad days. Given more time, I would love to explore this possibility.

Summary

This project was a lot of fun and made the process of exploring and learning about machine learning easier. I can really see the value of a team approach to model building especially when it comes to feature engineering, feature scaling, algorithm selection and algorithm tuning. Having like-minded individuals with different backgrounds and experience could really help with the process of model building through machine learning. I'm exciting to see where this new knowledge can lead and help uncover in this new world of ubiquitous data.