Identify Fraud from Enron Email & Financial Compensation Metrics

by Frank Corrigan

Question Response Document

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"]

In the late 1990's, Enron executives Ken Lay and Jeffrey Skilling developed a staff of executives that, by the use of accounting loopholes, special purpose entities, and poor financial reporting, were able to hide billions of dollars in debt from failed deals and projects which eventually led to Enron's bankruptcy; the largest corporate bankruptcy in U.S. history at that time.[1] By analyzing financial compensation and email metadata for Enron employees, it is possible to detect patterns between features that identify an individual as a person of interest (poi)[2]. Due to the statistical complexity of the patterns we are looking for, it is extremely useful to employ machine learning methods for analytical model building.[3]

The dataset contains 14 financial compensation features (salary, bonus, stock value, etc.), 6 email metadata points (email address, emails sent, email received from poi, etc.), and a boolean feature declaring the individual as poi or not. The raw data contains a 'TOTAL' data point which is treated as an outlier and immediately removed. Kenneth Lay's total compensation was much more than anyone else, so he is also treated as an outlier. Robert Belfer and Sanjay Bhatnagar had some inconsistencies in their data points (sum of compensation != total_payments) so they are also removed as outliers. Lastly, the raw data also contains many "NaN" (not a number) values that are appropriately treated as 0's.

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

[1] https://en.wikipedia.org/wiki/Enron_scandal

[2] A person of interest is defined as individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

[3] http://www.sas.com/en_us/insights/analytics/machine-learning.html

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"]

My feature selection process was arduously iterative. In the final model, 9 features were used of which 3 were out of the box variables (*salary*, *shared_reciept_with_poi*, *deferred_income*) and the other 6 were hybrid variables. To illustrate, two of the strongest hybrid features in the model I called *sal_navalue_count* and *fromto_ratio*. With the raw dataset being as sparse as it was, I believed there was importance in the metadata of the data itself. Hence, I started using counts of the missing values for each observation as features. Ultimately, the number of missing values in the financial compensation data points (not including stock option data) proved valuable for prediction. For the email-related features, I intuitively knew that individuals with greater relative communication to and from poi were probably poi themselves. Hence, the *fromto_ratio* is the addition of percentage of emails received from poi and the percentage of emails sent to poi for each individual.

To pick and create features, I used a combination or decision tree feature_importances_, SelectKBest, and PCA. I started early using feature_importances_ from decision trees, but once I started using PCA I relied more heavily on SelectKBest and exploring the principle components. The scores for best 10 features were

('salary', 6.4679679843454627)
('shared_receipt_with_poi', 7.1818771442764069)
('deferred_income', 17.17552763271879)
('bonusexpsal', 9.559316692235166)
('nonsal_ratio', 12.369312712856718)
('em_ratio', 8.2832315549109197)
('total_navalue_count', 7.558259797411444)
('total_total', 4.1176785774775935)
('sal_navalue_count', 11.707644067293582)
('fromto_ratio', 11.269952240225802)

The four best scores, highlighted, are for *deferred_income* (compensation related), *nonsal_ratio* (compensation related), *sal_navalue_count* (compensation related), and *fromto_ratio* (email related). Interestingly, despite *deferred_income*'s top score, it is one of the weaker links of the principal components. Likewise, *nonsal_ratio* is even weaker as highlighted below. Removing *deferred_income* from the set decreases the final model performance, however, removing *nonsal_ratio* increases it.

Final Feature List

['salary', 'shared_receipt_with_poi', 'deferred_income', 'bonusexpsal', 'nonsal_ratio', 'em_ratio', 'total_navalue_count', 'total_total', 'sal_navalue_count', 'fromto_ratio']

Principal Components of Top 10 Best Features (using SelectKBest)

```
[[ 0.29730099  0.22052028 -0.07219128  0.14270303  0.06641305  0.53228948
  -0.41331956  0.05977197 -0.35876406  0.49653501]
 [ 0.37946421  0.04596605 -0.10449767  0.16261413  0.04120302 -0.4747261
  -0.16055398  0.25952397 -0.58448162 -0.39334377]]
```

The definitions of each feature can be found in the Appendix.

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"]

Final model = DecisionTreeClassifier from sklearn's tree module. I began model selection process with this model selection graphic from sklearn which brought me through LinearSVM, KnearestNeighbors, RandomForest, RandomForest with Adaboost. and KernelSVM (surprisingly, the individual DecisionTreeClassifier is not on this map). I also attempted to fit a LogisticRegression regression model and finally chose the DecisionTreeClassifier. In the tester.py file, the model was able to acheive 89.7% accuracy, precision_score of 0.63, recall_score of 0.57, and f1_score of 0.60. The Adaboost model did have decent evaluation performance, however, it took forever to run on the test set. The KernelSVM was also decent, but neither of these came close to the results achieved with the final DecisionTreeClassifier.

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"]

Tuning the parameters means you can alter the behavior of how your algorithm uses the given dataset to learn. Some models, such as the SVMs, really needed tuning. Without this step, the model badly overfit the training data and the evaluation performance was poor. In this specific instance, utilizing the regularization parameter (C) to penalize extreme parameter values helped the model generalize to unseen data.

Without parameter tuning on our DecisionTreeClassifier, the F1 score fell from the achieved 0.60 to 0.40, not a short ways away. To identify the optimal parameters, I leveraged sklearn's GridSearchCV from the grid_search module with a StratifiedShuffleSplit cross-validation method. The three parameters I was attempting to tune, according to optimal accuracy, were the *criterion* to calculate information gain ('gini' or 'entropy'), *min_samples_split* which dictates the # of observations required to created another level (or leaf) of the decision tree, and whiten for the PCA which, if True, would 'ensure uncorrelated outputs with unit component-wise variances[4].' After removing outliers, selecting a feature set, scaling the data, and performing PCA to determine 2 principal components, the optimal parameters were criterion = 'gini', whiten = True, and min_samples_split = 60.

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"]

We want to ensure that we build a model that will generalize well to unseen data. Utilizing a validation strategy for model selection can help. For instance, in this project I broke the local dataset into a training and validation set, built a model on the training set, and tested the model on the validation set. When I felt confident that the predictions made on the local validation set were sufficient, I brought my model to the tester.py file where the model would make predictions on 15,000 unseen observations and evaluation metrics were calculated. To some extent, I think I cheated because I repeatedly ran the tester.py file while adjusting parameters and I beleive this is the essence of bad modeling (coach, please comment!).

So, I used the StratifiedShuffleSplit cross-validation method within my GridSearchCV while selecting a model. I set the classifier random_state = 42, created a parameter set, and fed the Pipeline into GridSearchCV. I examined the scores in each fold and selected the model parameters using clf.best_params_. I used these params in poi_id.py and when I ran the tester.py file. The rounded results for accuracy, precision, recall, and f1 are 0.90, 0.63, 0.57, and 0.60 respectively. If I tweak the parameters (in the tester.py file) I can increase these results, but I am quite confident now that that would defeat the purpose of using cross validation and having a model that would generalize well to yet another unseen dataset.

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"]

While the evaluation metrics of this model exceed the minimum requirements according the the

---

[4] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

grading rubric, I am still torn about the validity for assessing the model. The accuracy score, which is the percent of observations labeled correctly, is a healthy 89.7%. This author recently made the claim, 'let your algorithm do 90% of the work and let a human do the last 10%.' So 89.7% is sufficient. A 0.63 precision_score means that when our model identifies an individual as a poi, this is correct 63% of the time. I am not exactly comfortable accusing individuals of corporate fraud with only a 60% rate. Blackstone's formulation rings true for me here, 'better that ten guilty persons escape than that one innocent suffer[5].' According to our metric, if we identify 10 poi, this means that we likely have 10 - 6 = 4 type I errors or 4 people we identified as poi are actually not poi. To reiterate, if we identify 6 poi of the 10 actual poi, our precision score is 6 / 10 = 0.60.

Recall_score, on the other hand, tells us what percentage of poi we are erroneously letting 'go free.' These are our type II errors. For instance, imagine there are actually 15 poi in our dataset and we have identified 10 of them as poi. This means there are 15 - 10 = 5 individuals that have elluded suspicion. Our recall score is then 10 / 15 = 0.66.

Appendix

Reference List

- Udacity starter code for project -- https://github.com/udacity/ud120-projects
- Read this book as a supplement to the course -- https://www.packtpub.com/big-data-and-business-intelligence/python-machine-learning
- For scaling features -- http://napitupulu-jon.appspot.com/posts/feature-scaling-ud120.html
- Blog post on executing machine learning project --- from exploratory analysis to model parameter tuning -- https://jmetzen.github.io/2015-01-29/ml_advice.html
- Seaborn for visualization -- http://stanford.edu/~mwaskom/software/seaborn/index.html
- Sklearn documentation -- http://scikit-learn.org/stable/documentation.html
- For better understand validation and gridsearchcv -- https://www.youtube.com/watch?v=GoI_qOgRqfA
- For feature selection ideas -- http://machinelearningmastery.com/an-introduction-to-feature-selection/

Feature Definitions

['salary', 'shared_receipt_with_poi', 'deferred_income', 'bonusexpsal', 'em_ratio', 'total_navalue_count', 'total_total', 'sal_navalue_count', 'fromto_ratio']

- *salary*: (base) Reflects items such as base salary, executive cash allowances, and benefits payments.

[5] https://en.wikipedia.org/wiki/Blackstone%27s_formulation

- *shared_receipt_with_poi*: (base) The number of emails that this individual was copied on that were sent to a known poi.
- *deferred_income*: (base) Reflects voluntary executive deferrals of salary, annual cash incentives, and long-term cash incentives as well as cash fees deferred by non-employee directors under a deferred compensation arrangement. May also reflect deferrals under a stock option or phantom stock unit in lieu of cash arrangement.
- *bonusexpsal*: (hybrid) Addition of bonus, expenses, and salary.
- *em_ratio*: (hybrid) Ratio of poi-related emails to total emails. Calculated as (from_poi_to_this_person + from_this_person_to_poi + shared_receipt_with_poi) / (from_messages + sent_messages).
- *total_navalue_count*: (hybrid) Sum of “NaN” for each individual across both financial and email metrics.
- *total_total*: (hybrid) Total financial compensation. Sum of *total_payments* and *total_stock_value.*
- *sal_navalue_count*: Sum of “NaN” for each individual for only financial payments (excludes stock information).
- *fromto_ratio*: Ratio of poi-related emails to total emails excluding shared_reciept_with_poi. Calculated as (from_poi_to_this_person / sent_messages) + (from_this_person_to_poi / from_messages).