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 is 146 observations x 21 features. Features are split into roughly three groups - 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 removed, leaving 145 rows or observations. Of the 145 individuals, 18 (or 12.5%) are labeled as poi. This is a disproportionate

[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

distribution and will need to be addressed when selecting a cross validation method during modeling. Of the 3,045 data points, 1,352 (or 44.5%) are missing values. Yikes. Some features have a greater number of missing values than others.

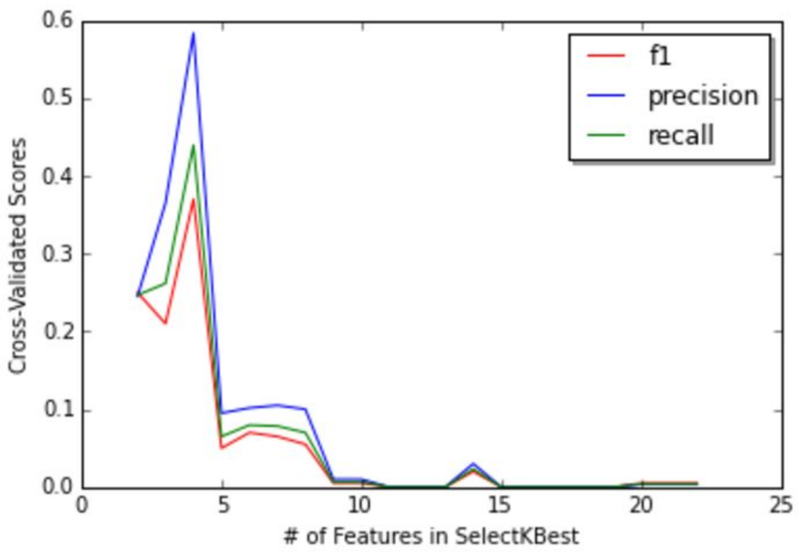
salary	51
to_messages	59
deferral_payments	107
total_payments	21
exercised_stock_options	44
bonus	64
restricted_stock	36
shared_receipt_with_poi	59
restricted_stock_deferred	128
total_stock_value	20
expenses	51
loan_advances	142
from_messages	59
other	53
from_this_person_to_poi	59
poi	0
director_fees	129
deferred_income	97
long_term_incentive	80
email_address	34
from_poi_to_this_person	59

These missing values are treated as 0. This is sensible for the financial metrics. Unfortunately, while this makes less sense for the email metrics we do not have email data on roughly 40% of the individuals so we cannot simply drop them from the dataset. Lastly in the data exploration / cleaning stage, Robert Belfer and Sanjay Bhatnagar had some inconsistencies in their data points (sum of compensation != total_payments) so these are manually adjusted using the *enron61702insiderpay.pdf* file.

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"]

During the feature selection process, I found that performance of the algorithms was better with scaled data -- and StandardScaler (mean = 0, sd = 1) was better than MinMaxScaler (scales data between 0 and 1) for most algorithms. While I did try an abundance of engineered features, the final model contains only 3 features --- exercised_stock_options, total_stock_value, and bonus --- none of which were engineered and no email features were used. Two features I attempted to engineer were *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. Unfortunately, while this feature was marginally important with decision trees, decision trees were not my final model and therefore this was not used. 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. Surprisingly, this feature also does not make it to the final model.

```
<matplotlib.text.Text at 0x110c55ad0>
```



SelectKBest was ultimately used to pick features. I systematically tested the best features one by one with each algorithm to find the combination of features with the best performance for each model judged by recall, precision, and f1 scores. I ordered every feature by SelectKBest score, then fit a knn model with only the first feature and recorded the scores. Then, I fit another model with the first two features and recorded the scores. Then, I fit another model with the first three features and so on until I had a plot of features vs performance for all features which is shown to the right. I repeated this process for 5 other algorithms and elaborate more on this in the next section.

Final Feature List
```
[  'exercised_stock_options',  'total_stock_value',  'bonus' ]
```

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”]

KNeighborsClassifier() was selected. Through a systematically iterative processes, I chose the model that gave me the best f1_score. Comparison in the table below shows the difference between using/not-using SelectKBest, scaling/not-scaling features with both StandardScaler() and MinMaxScaler(), using/not-using engineered features (fe in the table) in the SelectKBest process, and using/not-using PCA(). Highlighted in yellow are some of the highest individual values and averages in each table.

f1 table with NO SelectKBest method

	no fe, no scaling	no fe, standardScale	no fe, MinMaxScale	fe, standardScale	fe, standardScale, pca
knn	0.32	0.00	0.06	0.01	0.01
lr	0.17	0.22	0.10	0.18	0.18
bayes	0.25	0.26	0.31	0.34	0.28
kernel	divByZero Error	divByZero Error	divByZero Error	divByZero Error	divByZero Error
dectree	0.22	0.22	0.22	0.28	0.23
average	0.24	0.18	0.17	0.20	0.18
avg. avg.					0.19

f1 table with SelectKBest method

	no fe, no scaling	no fe, standardScale	no fe, MinMaxScale	fe, standardScale	fe, standardScale, pca
knn	0.37	0.51	0.41	0.51	0.51
lr	0.18	0.43	0.17	0.43	0.43
bayes	0.43	0.43	0.43	0.41	0.37
kernel	divByZero Error	0.43	0.22	0.43	0.43
dectree	0.37	0.38	0.28	0.37	0.44
average	0.34	0.436	0.30	0.430	0.436
avg. avg.					0.39

First, utilizing the SelectKBest method consistently yields better performance than simply using all the features in the dataset. Second, scaling the feature set using StandardScaler() greatly improves performance on most algorithms. Using the engineered features in the SelectKBest process does not seem to improve performance nor does PCA except with the decision tree classifier. As shown in the second table, the knn algorithm is clearly the best algorithm which is used as the final model.

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. You can see in the tables above that the f1_score rises from 0.43 to 0.47 after tuning the parameters for svc. Similarly, the aggregate average score rises from 0.39 to 0.42 after using the GridSearchCV methodology.

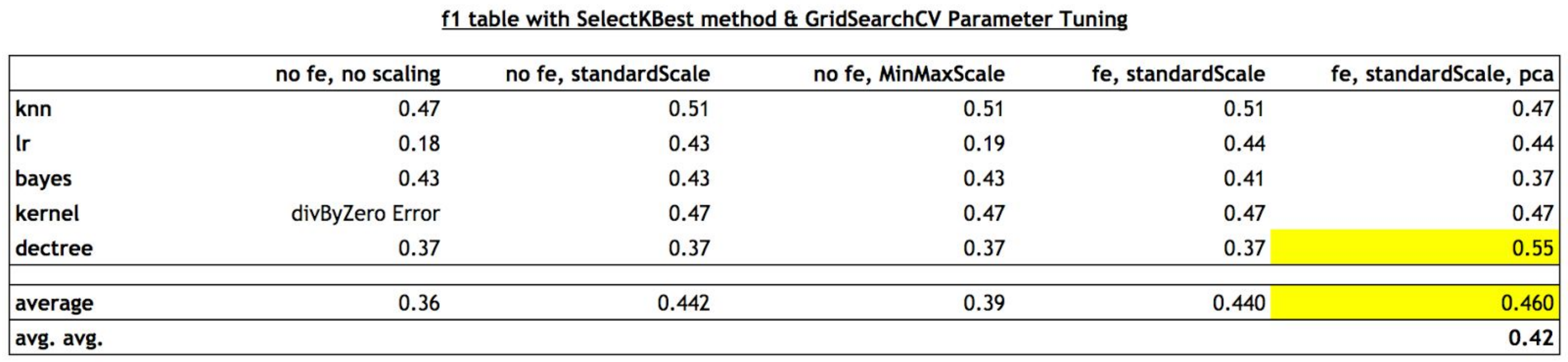
Although tuning the parameters for the final model -- a KneighborsClassifier() -- didn't have a significant effect I did search for the parameters that maximized precision rather than f1 for reasons explained in Question #6. To identify the optimal parameters, I leveraged sklearn's GridSearchCV from the grid_search module with a StratifiedShuffleSplit cross-validation method. I used the StratifiedShuffleSplit rather than say, a simple Kfold, due to the unbalanced number of pois in the dataset. The StratifiedShuffleSplit returns stratified randomized folds and the folds are made by *preserving* the percentage of samples for each class[4].

The three parameters I tried to automatically tune were weights (which influences how each point is weighted in relation to another), # neighbors (I like to think of this as the number of groups in the field that are either poi or non-poi. Groups 1 and 3 may be poi, while Groups 2, 4, and 5 are non-poi), and p (which defines which distance measure to use --- manhattan or euclidean). The final model and parameter set…

```
('Model name: ', 'knn')
('Best Features: ', ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus'])
Pipeline(steps=[('sc',         StandardScaler(copy=True,         with_mean=True,         with_std=True)),         ('clf',
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=5, p=1, weights='uniform'))])
{'clf__weights': 'uniform', 'clf__p': 1, 'clf__n_neighbors': 5}
```

And below is the same table from Question #3 after using the GridSearchCV methodology. Note that despite the DecisionTreeClassifier()'s highest f1 score after fe, scaling, and pca I opted to stick with the KneighborsClassifier() as the final model because it had a similar f1 score but much better precision score. Again, in Question #6 I elaborate on why this is important to me.

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



f1 table with SelectKBest method & GridSearchCV Parameter Tuning

	no fe, no scaling	no fe, standardScale	no fe, MinMaxScale	fe, standardScale	fe, standardScale, pca
knn	0.47	0.51	0.51	0.51	0.47
lr	0.18	0.43	0.19	0.44	0.44
bayes	0.43	0.43	0.43	0.41	0.37
kernel	divByZero Error	0.47	0.47	0.47	0.47
dectree	0.37	0.37	0.37	0.37	0.55
average	0.36	0.442	0.39	0.440	0.460
avg. avg.					0.42

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 due to the small size of the dataset I used the entire thing to build and validate a model before bringing it to the test set in tester.py. I used StratifiedShuffleSplit cross validation with 1,000 iterations so I could be confident that the results I was seeing locally would be the same as when I ran the tester.py file. It took some time, but eventually I stopped attempting to maximize the score I was getting in the tester.py file since I realized I was training on the test set… and this would lead to bad generalization to unseen data.

So, I used the StratifiedShuffleSplit cross-validation method with 1,000 iterations 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 precision, recall, and f1 are 0.87, 0.37, and 0.51 respectively. If I tweak the parameters (in the tester.py file) I can increase these results slightly, 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 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 89.2%. This author recently made the claim, 'let your algorithm do 90% of the work and let a human do the last 10%.' However, due to the accuracy paradox[5] this is not an appropriate measure for evaluation. Precision, recall, and f1 score are better suited. A 0.85 precision_score means that when our model identifies an individual as a poi, this is correct 85% of the time. Blackstone's formulation rings true for me here, 'better that ten guilty persons escape than that one innocent suffer[6].' According to our metric, if we identify 10 poi, this means that we likely have 10 - 8.5 = 1.5 type I errors or 1.5 people we identified as poi are actually not poi. To reiterate, if we identify 8.5 poi of the 10 actual poi, our precision score is 8.5 / 10 = 0.85. A higher precision score is more desirable than a higher recall score (assuming a trade-off exists between the two).

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 14 poi in our dataset and we have identified 5 of them as poi. This means there are 14 - 5 = 9 individuals that have elluded suspicion. Our recall score is then 5 / 14 = 0.35. In the event that we were more concerned with conviction over justice, we may chose the DecisionTreeClassifier() which had a higher recall score than precision score.

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

[5] http://dataskeptic.com/epnotes/the-accuracy-paradox.php
[6] https://en.wikipedia.org/wiki/Blackstone%27s_formulation

- For better understand validation and gridsearchcv -- https://www.youtube.com/watch?v=Gol_qOgRqfA
- For feature selection ideas -- http://machinelearningmastery.com/an-introduction-to-feature-selection/
- Ongoing Data Science education - http://dataskeptic.com/
- Ongoing Data Science education - https://www.udacity.com/

Feature Definitions

```
[  'exercised_stock_options',  'total_stock_value',  'bonus' ]
```

- *total_stock_value*: (base) Total value of restricted and unrestricted stock awarded to individual
- *exercised_stock_options*: (base) Monetary amount of stock options that were 'cashed in' by individual
- *bonus*: (base) Total financial incentive awarded to individual based on 'annual performance.'