Identify Fraud from Enron Email

Darren Chiu

Note: All the wrangling on the data and process of experimenting with the machine learning algorithms are in the 'Enron_Data_Exploration' jupyter notebook. It serves as a record of stuff that I tried and some notes throughout the process. It was not intended to be use as documentation but a reference of the thought process.

Final Model

Gaussian-Bayes classifier, with three features used - 'exercised_stock_options', 'expenses', 'deferred_income'. This model achieved a precision score of 0.55515 and recall score of 0.38000.

Model Tuning Documentation

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"]

   The goal of the project is to classify if a person is Person of Interest(POI) in the Enron email and financial details that were released to the public space.
   The dataset was entered into the public record as a result of Federal investigations. The dataset contained information about the payoff of employees and email details of Enron employees.

   After getting the data, I did some basic wrangling on the dataset. There were 146 records, with 20 features. There were 18 POIs in the original dataset, i.e. 128 non-POIs. In other words, there were only 12.33% of the observations are POIs.

   I also did an investigation on the data integrity and come up with a summary as below:

Field	Number of missing values
salary	51
deferral_payments	107
total_payments	21
loan_advances	142
bonus	64
restricted_stock_deferred	128
deferred_income	97
total_stock_value	20
expenses	51
exercised_stock_options	44
other	53
long_term_incentive	80
restricted_stock	36
director_fees	129
to_messages	60
email_address	35
from_poi_to_this_person	60
from_messages	60
from_this_person_to_poi	60
shared_receipt_with_poi	60
poi	0

After I got the data, I did some examination and found that there is an entry "TOTAL" in the dataset which obviously didn't represent any employee. I removed this record. There was a record with no data at all (LOCKHART EUGENE E), so this is also removed. After some manual check on the names, I also found that there is an entry of "THE TRAVEL AGENCY IN THE PARK" which is also a meaningless or wrong record as well.

Histograms on the financial fields are also plotted and 2 people (SKILLING JEFFREY K and LAY KENNETH L) with outlier values in salary and total_payments. However, we won't be removing them as they are both POIs and we already have an extremely small number of POIs in out dataset.

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 importance 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", "intelligently select features", "properly scale features"]

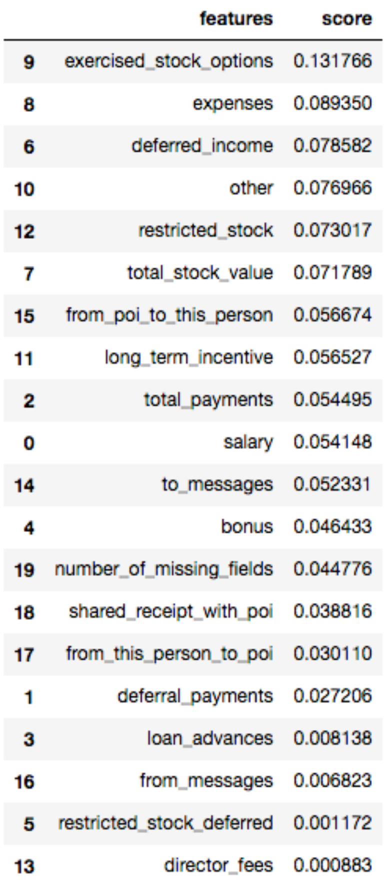
   There were 3 features being used in the final model – 'exercised_stock_options', 'expenses', 'deferred_income'. For all the selected fields, I did feature scaling using MaxMin scaling method. This is because I was planning to try out SVM algorithm and SVM is scale sensitive.

I have also try engineer my own feature that are not come with the original dataset. I noticed that people with a lot of missing fields are usually not POI. Thus, I tried created a new feature based on the number of missing fields in that records. The feature was named 'number_of_missing_fields' which equals to the number of missing fields of that person. I created the feature using the following code:

```python
for person in data_dict.keys():
    data_dict[person]['number_of_missing_fields'] = sum(1 for x in
data_dict[person].values() if x == 'NaN')
```

In terms of feature importance, it ranked 13 across other features with an feature importance score of 0.044776. As we later found that optimal number of features for SVM is 5, decision tree is 14 and Gaussian-Bayes is 3. This engineered feature was only included in the decision tree analysis. This will be explained in the next part of this report.

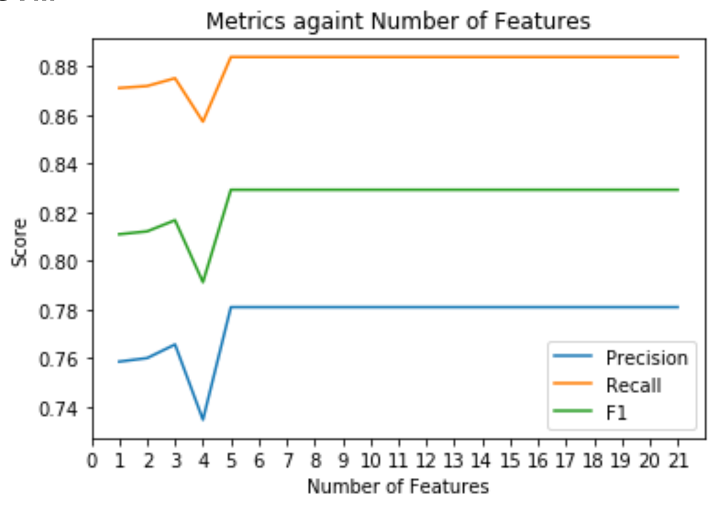
As we are not sure if our features are independent to each other, SelectKBest might not be suitable in our case. So, we will use Feature Importance from Extra Trees ensemble to decide the features to use. The features with sorted score are shown below:



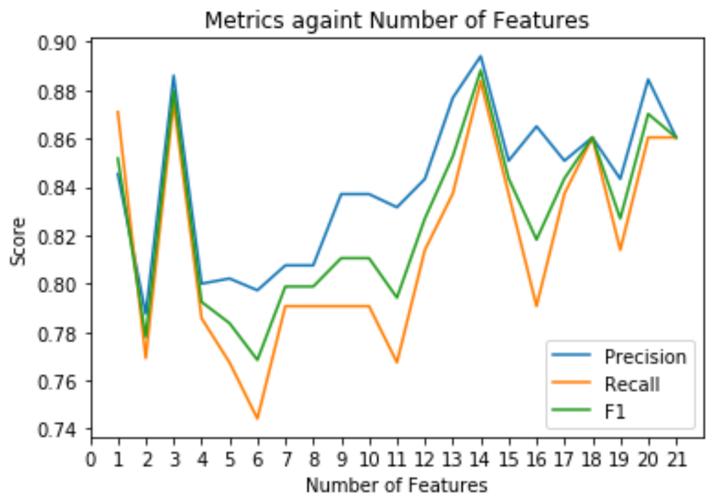
	features	score
9	exercised_stock_options	0.131766
8	expenses	0.089350
6	deferred_income	0.078582
10	other	0.076966
12	restricted_stock	0.073017
7	total_stock_value	0.071789
15	from_poi_to_this_person	0.056674
11	long_term_incentive	0.056527
2	total_payments	0.054495
0	salary	0.054148
14	to_messages	0.052331
4	bonus	0.046433
19	number_of_missing_fields	0.044776
18	shared_receipt_with_poi	0.038816
17	from_this_person_to_poi	0.030110
1	deferral_payments	0.027206
3	loan_advances	0.008138
16	from_messages	0.006823
5	restricted_stock_deferred	0.001172
13	director_fees	0.000883

We also don't know how many features should we take. Thus, I plot graphs of precision score, recall score and F1 score for all three models that I am going to compare, namely Gaussian-Bayes, decision tree and SVM.

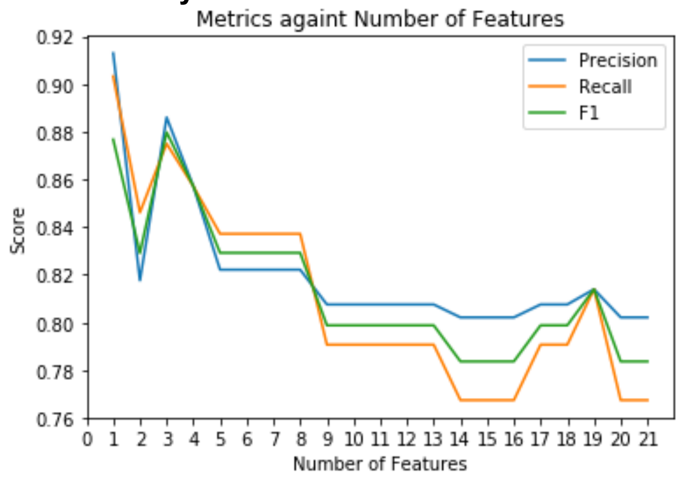
SVM



Decision Tree



Gaussian-Bayes



To conclude, 5 features would be the optimal number of features for SVM, 14 for decision tree and 3 features would be optimal for Gaussian-Bayes.

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 end up using Guassian-bayes. I tried SVM and decision tree as well. From my experiments, Guassian-bayes performed the best and decision tree performed the worst (all using the setting tuned by GridSearchCV and optimal number of features).

Below are the summary statistics of the three models using their optimal settings

For Decision tree (with 14 features)
Precision: 0.30361
Recall: 0.18900

For SVM (with 5 features)
Precision: 0.45876
Recall: 0.13350

For Gaussian-Bayes Classifier (with 3 features)
Precision: 0.55515
Recall: 0.38000

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? What parameters did you tune? (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 items: "discuss parameter tuning", "tune the algorithm"]

   Each algorithm might have more than 1 parameter to set and these are all impacting the performance of a machine learning algorithm. If we didn't tune it well, some might be causing overfitting or overgeneralization of the model which results in low precision/recall score.

   I used the GridSearchCV to tune the parameter for SVM and Decision tree. For SVM, 2 parameters were tuned, the kernel and C value. C value was tuned against the range of 1000 to 100,000, with stepping size of 1000.

   For decision tree, 5 parameters were tuned, criterion, splitter, max_features, max_depth and min_samples_split. Details of the tuning is available in the jupyter notebook.

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 items: "discuss validation", "validation strategy"]

   Validation is the process of evaluating the performance of machine learning model. Some common mistakes are using same set of data as training set and test set. This could result in an extremely high accuracy and lead to overfitting if someone keep trying to improve the model without splitting data into training and test set.

   I mostly use precision and recall evaluating my setup in the tester.py. I didn't use accuracy as a major evaluation metrics because our sample is highly skewed with a very small number of POIs. We can easily be deceived by the algorithm if it biased its predictions to non-POIs.

In tester.py, it used the StratifiedShuffleSplit to break down the data into multiple train/test set for cross validation. The reason of using StratifiedShuffleSplit is also our highly skewed dataset. Using StratifiedShuffleSplit can help getting a relatively equal size of classes in the testing set. Before I send my model to tester.py, I also printed out the precision score and recall score to get a rough understanding of the model performance.

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 common evaluation metrics are precision and recall. My final model is having an average precision of 0.55515 and recall of 0.38000.

   For precision, it means that out of all the prediction that the model classified as POI, 55.515 % of those are real POI.

   For recall, it means that out of all the prediction made for entries that should be classified as POI, 38.0% of them are classified as POI.