Project: Identity Fraud from Enron e-mail

Jeffrey Ryu
28 September 2017

Goal

The goal of the project is to use the public Enron records, including detailed financial data for top executive as well as tens of thousands of emails, to identify the person of interest (poi) by utilizing the machine learning techniques learned in the lesson. For this project, poi is defined as a person who was indicted, settled without admitting guilt or testified in exchange for immunity. Following data will be used to help identify the poi.

- Financial data has salary and other compensation information, such as bonus, restricted stock and director fees.
- Email data has information such as number of emails from poi to this person and number of emails from this person to poi.
- There is also a poi labels data, which has lists of employees and a Boolean variable to show whether they are poi.

There are 146 rows and 22 features in the initial data set. While 35 people were listed as poi in the 'poi_names.txt, only 18 people were flagged as poi of the 146 rows. The other 17 people were not part of this data set.

Outliers

- One of the rows in the initial data set is for person 'TOTAL'. Instead of being a record for an individual, this is a sum of all of the financials for those with financial records. For example, the salary for 'TOTAL' is $26.7 million while the next highest is $1.1 million. This is clearly an outlier and is removed.
- Another row is for "THE TRAVEL AGENCY IN THE PARK' with total payments of $362,096. According to the footnote (j), this is a travel agency that was paid for business-related travel made by Enron employees and is co-owned by the sister of Enron's former chairman. I will remove this also because this is not a person.
- 'LOCKHART, EUGENE E' had NaN values for all the features. This means that he was not being paid nor sent e-mails from Enron email address. He also did not have an Enron email address. Perhaps he is an ex-employee who had left the company. After giving some thoughts, I decided not to remove him because despite not being paid or having sent an email, he had a potential to be poi if he was related to the company.

I also observed features data by plotting feature value versus poi value as described below in the "Initial features" section. I decided not to throw out any other data based on these observations. After removing the two outliers, we are left with 144 rows of data.

NaN treatment

Another thing I noticed from the data was that there is a lot of 'NaN'. The starter code had 'NaN' replaced with zero. There are a few methods to handle 'NaN'; they can be replaced with zero, average, with the most common occurrences, or left as it is. As I worked through the project, it made sense to replace the 'NaN with zeroes as most of these values would have been zero. For example, 'NaN' values in director fees is likely from that person not receiving any director fee, rather than being an input error. Hence, it made sense to replace 'NaN' with zero.

Initial features

I started out with all the available features, since I did not know which ones are stronger features than others. For each feature, I made a scatter plot of feature value versus poi value to visualize the data. For example, I made scatter plots of salary versus poi, deferral payments versus poi, total payments versus poi, and so on. On the first run, I removed the data if the value of 'NaN' was found. Two of the features ('restricted_stock_deferred' and 'director_fees') only had 'poi' values of zero when all 'NaN' were removed. After looking through all the scatter plots, I went over every features list and made sure that the zero made sense for all the features. I then re-plotted all the charts with 'NaN' replaced with zeroes.

New features

I made several new features as well.

- First engineered feature is the ratio of long term incentive to total payments. Long term incentive is a compensation that will occur down the road. My rationale was that the higher the ratio of the longer term payments to their total will incentivize a person to perpetuate the fraud.
- Second feature is the ratio of restricted stock to total stock value. Similar to the long term incentive, restricted stock is also a compensation that will occur down the road. The rationale is the same as above.
- Third feature is the ratio of bonus to total payments. It may be that poi may have higher ratio of bonus relative to their total pay.
- Fourth feature is the ratio of e-mail from poi to e-mail to poi. Poi may send more e-mails to other poi than non-poi relative to e-mails they receive.
- Fifth feature is the ratio of e-mails sent from poi to total emails sent. This ratio may be higher for poi.
- Sixth feature is the ratio of e-mails received from poi to total e-mails received. Similarly as the fifth engineered feature, this ratio may be higher for poi.

Parameter tuning

I decided to test out three different algorithms: Gaussian Naïve Bayes, Decision Tree and AdaBoost. For Gaussian Naïve Bayes, there was no parameter to tune. However, Decision Tree and AdaBoost had several parameters that could be tuned. By tuning the parameter through a various combinations, we can try to come up with better performing model. GridSearchCV helps achieve this by running through various combinations of parameters that were specified and returning the mean scores of those combinations.

For ranking the scores, I chose to optimize the F1 score which is a weighted average of precision and recall, rather than accuracy. This was because classes are imbalanced, i.e. there is only a few poi in the data set.

Validation

Instead of simple split into training and test data set, we can split data into training, validation and test data. We then select the model with the highest score in the validation set. Validation is important because a model that overfit the training data would be selected without the validation set. I used the StratifiedShuffleSplit as the cross-validation, because there were only a few poi in the data set. This allows me to have similar proportion of poi in each data set. Otherwise, it is likely that a split would leave one or more of the data set with all non-poi, i.e., all the result are expected to be zero regardless of features.

Decision Tree algorithm

For the first algorithm, I used the the DecisionTreeClassifier with StratifiedShuffleSplit and SelectKBest. I ran this in GridSearchCV using following parameters. Given that there is a very few poi in the data set, perhaps a few features can have sizable impacts on the result. Thus, I more lower range values than higher ones.

Estimator	parameters	Values
SelectKBest	K	[1, 2, 3, 4, 5, 6, 8, 10, 15, 20]
DecisionTreeClassfier	criterion	['gini', 'entropy']
DecisionTreeClassfier	Splitter	[‘random’]

Out of all the possible combinations, the highest mean score was 0.2810 for the following parameter: {'dtc__criterion': 'gini', 'kbest__k': 10, 'dtc__splitter': 'random'}.

Following 10 features were selected by the SelectKBest for the parameter with the highest mean score.

Feature	Score
exercised_stock_options	25.10
total_stock_value	24.47
Bonus	21.06
Ratio_bonus_to_total	20.99
Salary	18.58
Ratio_lt_incentive_to_tot_payments	14.01
Deferred income	11.60
Long_term_incentive	10.07
Restricted_stock	9.35
Total_payments	8.87

Running through the tester.py, precision, recall and f1 scores were 0.3002, 0.3080 and 0.3041, respectively. (Note to the reviewer: given that the splitter is random, it is possible that the run will give precision and recall values below 0.30 on some runs. I have tried many times and more than half the runs will return both values greater than 0.30.)

Decision Tree generally does not require scaling, so I moved onto next algorithm.

Gaussian Naïve Bayes algorithm

Similarly, I ran GridSearchCV with StratifiedShuffleSplit using Gaussian Naïve Bayes and SelectKBest with parameters just varying k values of [1, 2, 3, 4, 5, 6, 8, 10, 15, 20] for SelectKBest. The best mean score was slightly higher at 0.328 with a SelectKBest k of 10. Below are the 10 features selected and their scores.

Feature	**Score**
exercised_stock_options	25.10
total_stock_value	24.47
Bonus	21.06
Ratio_bonus_to_total	20.99
Salary	18.58
Ratio_lt_incentive_to_tot_payments	14.01
Deferred income	11.60
Long_term_incentive	10.07
Restricted_stock	9.35
Total_payments	8.87

Running through the tester.py, precision, recall and f1 scores were 0.2260, 0.3950 and 0.2875, respectively.

Naïve Bayes does also generally does not require feature scaling, so I moved on to the next algorithm.

AdaBoost algorithm

For AdaBoost algorithm, I used the following parameters:

estimator	**parameters**	**Values**
SelectKBest	k	[1, 2, 4, 6, 10, 20]
AdaBoostClassifier	n_estimators	[10,100,200]
AdaBoostClassifier	max_depth	['SAMME', 'SAMME.R']
AdaBoostClassifier	random_state	[42]

The highest mean score was 0.2570 with the following parameters: {'kbest__k': 20, 'abc__n_estimators': 100, 'abc__random_state': 42, 'abc__algorithm': 'SAMME.R'}. AdaBoost has selected 10 additional features on top of what the Decision Tree chose.

Running through the tester.py, precision, recall and f1 scores were 0.3735, 0.2855 and 0.3236, respectively.

Logistic regression

For logistic regression, I used the following parameters:

Estimator	**parameters**	**Values**
SelectKBest	k	[1, 2, 3, 4,5, 6,8, 10,15, 20]
Logistic_regression	C	[1.0, 2.0]

The highest mean score was 0.1780 with the following parameters: {'kbest__k': 15, 'lr__C': 1.0}. Logistic regression has selected 15 features.

Running through the tester.py, precision, recall and f1 scores were 0.1838, 0.1790 and 0.1814, respectively.

<u>Evaluation metrics</u>

F1 score is a weighted average of precision and recall. Precision is the probability of that a person is poi given that the model predicts him or her as poi. Recall is the probability that a person is poi given that he or she is actually as poi. These scores are preferred over the accuracy because classes are imbalanced, i.e. there are many non-poi to poi. Thus, a model that overly predicts a person to be non-poi can have high accuracy, but not high precision or recall. Therefore, F1 score was optimized in the parameter tuning instead of the accuracy.

The F1 scores were fairly close for the three algorithms, with AdaBoost performing slightly better than the others with F1 score of 0.3236. AdaBoost score was enhanced by its high precision score of 0.3735. Decision tree algorithm had more consistent precision and recall score, as they both were north of 0.30.

	Precision	**Recall**	**F1**	**k**
DTC	0.3002	0.3080	0.3041	10
GNB	0.2260	0.3950	0.2875	10
AdaBoost	0.3735	0.2855	0.3236	20
Logistic regression	0.1838	0.1790	0.1814	15

(Note to the reviewer: given that the splitter is random, it is possible that the run will give precision and recall values below 0.30 on some runs. I have tried many times and more than half the runs will return both values greater than 0.30.)

<u>Final remark</u>

This was a truly challenging project to say the least. I struggled through many steps along the way, googling and re-reading multiple documentations, articles, videos and notes. I only realized now that the project requests of references that I have used. Unfortunately, I did not keep track of all the websites and videos that I have visited because there were at least a few hundreds of them. I think every single one of them helped me out a bit by bit.

I also utilized the Udacity's Live Help frequently if I was stuck for a long period of time. Live Help was extremely helpful. Almost every time after using it, I felt like I should have gone to the Live Help sooner than struggling so much.

While it has been challenging, I am looking forward to learning more about machine learning!