Identifying Fraud from Enron Email

Ada Lee

November 1, 2015

Dataset and Question

Data Exploration

Enron was one of the largest companies in U.S. in 2000, and it collapsed into bankruptcy due to corporate fraud. In this project I will try to build machine learning algorithms to predict weather a person in the company took part in the fraud (POI or non-POI). There are 146 data points with 20 features and one label (`poi`) in our dataset. 18 data points (12.3%) are POI and 128 (87.7%) are non-POI. There are many missing values in the dataset, for example `loan_advances`, `director_fees`, `restricted_stock_deferred`, `deferral_payments`, `deferred_income` and `long_term_incentive` have more than 50% of missing values.

feature	count of missing values	percent of missing values
loan_advances	142	97.3%
director_fees	129	88.4%
restricted_stock_deferred	128	87.7%
deferral_payments	107	73.3%
deferred_income	97	66.4%
long_term_incentive	80	54.8%
bonus	64	43.8%
to_messages	60	41.1%
shared_receipt_with_poi	60	41.1%
from_messages	60	41.1%
from_this_person_to_poi	60	41.1%
from_poi_to_this_person	60	41.1%
other	53	36.3%
salary	51	34.9%
expenses	51	34.9%
exercised_stock_options	44	30.1%
restricted_stock	36	24.7%
email_address	35	24.0%
total_payments	21	14.4%
total_stock_value	20	13.7%

Outlier

There is one outlier in the data set, whose name is "TOTAL", "TOTAL" is not a person name and its financial feature value is total of all people's financial value, other value is missing, poi is False. I just remove this outlier.

Add new feature

I add new feature `bonus_vs_total_payments` which means percent of bonus over total payments of an Enron insider. The formula is `bonus_vs_total_payments` = `bonus` / `total_payments`. I think that the larger percent of bonus over total payments of an Enron insider, the more probable that this Enron insider take part in POI. Adding this feature do help improve performance of Naive Bayes algorithm, and it does not improve performance of Decision Tree.

Pick and Tune algorithms

We would like to fit some machine learning models and evaluate their performance. We see that this dataset is biased, we could still get 87.7% accuracy if we predict all data to be non-POI, so only accuracy would not be good evaluation of prediction performance. We are more concerned to recognize POI, that is when a person is POI, we want the probability to recognize them is high. If we predict all people as POI, then all people that is POI will be recognized as POI, we do not want that, we also want that if a person is predicted as POI, the probability that the person is POI is high. Therefore we need to use both recall and precision to evaluate machine learning performance in this dataset.

$$accuracy = \frac{\text{number of people that are correctly predicted as POI or non-POI}}{\text{number of all people in the dataset}}$$

$$recall = \frac{\text{number of people that are predicted as POI and they are actually POI}}{\text{number of people are actually POI}}$$

$$precision = \frac{\text{number of people that are predicted as POI and they are actually POI}}{\text{number of people that are predicted as POI}}$$

To avoid overfitting, we need to split the dataset into training dataset and testing dataset, and I make sure that percent of POI in both training and testing dataset is same by using `StratifiedShuffleSplit`. Since we only have 145 datasets after removing outlier "TOTAL", our dataset is small. To avoid that our performance is due to chance, I use 1000-fold cross-validation. In other words, for every machine learning algorithm I randomly split the data into training and testing dataset 1000 times, and fit algorithm and calculate performance 1000 times, and get average performance as performance of the algorithm.

Naive Bayes (GaussianNB)

Use SelectKBest

1. Initially choose all 19 features, create an empty table
2. While there are still remaining features
   2.1 fit a naive bayes model (`naive_bayes.GaussianNB`) using remaining features
   2.2 if both recall and precision is greater than 35%, add features and performance to the table
   2.3 use `SelectKBest` to remove the worst feature
3. Output the table

features	accuracy	recall	precision
exercised_stock_options,deferred_income,total_stock_value,bonus, restricted_stock,long_term_incentive,salary	0.8467	0.384	0.4566
exercised_stock_options,deferred_income,total_stock_value,bonus, long_term_incentive,salary	0.8471	0.37	0.4568
exercised_stock_options,deferred_income,total_stock_value,bonus,salary	0.8546	0.3805	0.4888
exercised_stock_options,total_stock_value,bonus	0.843	0.351	0.4858

From the table we see that best features for Naive Bayes are `exercised_stock_options`, `deferred_income`, total_stock_value, ‘bonus and salary. We get 38.1% recall and 48.9% precision.

Adding new feature bonus_vs_total_payments and use SelectKBest

features	accuracy	recall	precision
exercised_stock_options,deferred_income,total_stock_value,bonus, restricted_stock,long_term_incentive,salary,bonus_vs_total_payments	0.8381	0.352	0.4203
exercised_stock_options,deferred_income,total_stock_value,bonus, long_term_incentive,salary,bonus_vs_total_payments	0.8437	0.3515	0.441
exercised_stock_options,total_stock_value,bonus,bonus_vs_total_payments	0.8522	0.391	0.5266
exercised_stock_options,total_stock_value,bonus	0.843	0.351	0.4858

From the table we see that after adding new feature bonus_vs_total_payments, best features for Naive Bayes are exercised_stock_options, total_stock_value, bonus, bonus_vs_total_payments. We get 39.1% recall and 52.7% precision, so adding new feature bonus_vs_total_payments do really help improve Naive Bayes algorithm.

Use PCA for dimension reduction

I would try to using 2 to 19 PCA components to fit naive bayes, and find best algorithms. Below is result.

principle components	accuracy	recall	precision
2	0.8736	0.284	0.5504
3	0.8735	0.284	0.5493
4	0.8645	0.2665	0.485
5	0.86	0.236	0.4521
6	0.8607	0.236	0.4569
7	0.8504	0.2365	0.3975
8	0.8535	0.3515	0.4386
9	0.8549	0.3515	0.4441
10	0.8537	0.351	0.439
11	0.8645	0.351	0.4885
12	0.8625	0.351	0.4789
13	0.8493	0.291	0.4087
14	0.8213	0.3245	0.3279
15	0.8172	0.335	0.3218
16	0.8041	0.3365	0.2947
17	0.7985	0.344	0.2868
18	0.7981	0.3295	0.2808
19	0.8017	0.4065	0.3126

From the table we see that best result is using 11 principal components, we get 35.1% recall and 48.8% precision.

Use Feature Scaling, then use PCA

First I do min-max feature scale, then I choose 2 to 19 principal components, and fit a naive bayes.

principal components	accuracy	recall	precision
2	0.8538	0.334	0.4369
3	0.8501	0.334	0.4215
4	0.8215	0.335	0.332
5	0.8315	0.385	0.3723
6	0.8265	0.387	0.36
7	0.8311	0.435	0.3826
8	0.8342	0.439	0.3914
9	0.8308	0.426	0.38
10	0.8279	0.4155	0.3705
11	0.8181	0.417	0.3481
12	0.8274	0.419	0.37
13	0.8133	0.41	0.3359
14	0.8066	0.411	0.323
15	0.8017	0.403	0.3117
16	0.8005	0.397	0.3078
17	0.7991	0.397	0.3053
18	0.7918	0.4025	0.2945
19	0.7855	0.4205	0.2901

Add min-max scale, the best result now is using 8 principal components, with 43.9% recall and 39.1% precision.

Decision Tree

I would like to fit a decision tree as best as I can. Feature scaling does not affect decision tree, so I want do feature scaling in this part. Below is how I do feature selection. (Since the result is random, so I run 50 times, and get 10 results where recall+precision > 1).

1. Initially choose all 19 features
2. while there are still remaining features
   2.1 use remaining features to fit a new decision tree model [`DecisionTreeClassifier()`] and calculate Gini importance, recall, precision and accuracy
   2.2 if some features have 0 Gini importance, remove these features
   2.3 if all features have Gini importance larger than 1, remove 1 feature with lowest Gini importance
3. Output best features, accuracy, recall and precision with largest recall + precision

features	accuracy	recall	precision
deferred_income,exercised_stock_options,expenses	0.8593	0.502	0.5076
deferred_income,exercised_stock_options,expenses	0.8609	0.506	0.5132
deferred_income,exercised_stock_options,expenses	0.8619	0.5095	0.5167
exercised_stock_options,expenses,from_this_person_to_poi,other	0.8681	0.4925	0.5421
exercised_stock_options,expenses,from_this_person_to_poi,other	0.8665	0.4905	0.5358
exercised_stock_options,expenses,from_this_person_to_poi,long_term_incentive	0.8578	0.5055	0.5022
deferred_income,exercised_stock_options,expenses	0.8606	0.5055	0.5124
deferred_income,exercised_stock_options,expenses	0.8603	0.4985	0.5113
deferred_income,exercised_stock_options,expenses	0.8596	0.5055	0.5088
deferred_income,exercised_stock_options,expenses	0.8609	0.5015	0.5133

Among 10 best feature combinations that I found, 6 of them is "deferred__income,exercised__stock__options,expenses", so best features I find for random forest is `deferred_income`, `exercised_stock_options` and `expenses`. From the table we see that best recall and precision is both around 50%.

Tune some parameters of decision tree.

Machine learning algorithms are parameterized and modification of those parameters can influence the outcome of the learning process. Tuning a machine learning algorithm is important because it help us to find better algorithm. In this section I would like to tune decision tree with selected features `deferred_income`, `exercised_stock_options` and `expenses`. I would tune `criterion`, `splitter`, `max_features`, `min_samples_split`. Their meanings are:

- `criterion` : string, optional (default="gini")
  The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- `splitter` : string, optional (default="best")
  The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.
- `max_features` : int, float, string or None, optional (default=None)
  The number of features to consider when looking for the best split:
  - If int, then consider `max_features` features at each split.
  - If float, then `max_features` is a percentage and `int(max_features * n_features)` features are considered at each split.
  - If "auto", then `max_features=sqrt(n_features)`.
  - If "sqrt", then `max_features=sqrt(n_features)`.
  - If "log2", then `max_features=log2(n_features)`.
  - If None, then `max_features=n_features`.
- `min_samples_split` : int, optional (default=2) The minimum number of samples required to split an internal node.

Choices of them in my experiment are show below.

`criterion`: "gini", "entropy"
`splitter`: "best", "random"
`max_features`: "auto", "sqrt", "log2", None
`min_samples_split`: 2, 3, 4, . . . , 18, 19, 20

There would be 304 results ($2 \times 2 \times 4 \times 19 = 304$). We only show results with both recall and decision larger than 0.45.

criterion	splitter	max__features	min__samples__split	accuracy	recall	precision
gini	best		2	0.8606	0.503	0.5122
gini	best		3	0.8679	0.511	0.5396
gini	best		4	0.8611	0.4855	0.5146
entropy	best		2	0.8566	0.4745	0.4982
entropy	best		3	0.8581	0.4575	0.5039
entropy	best		4	0.8523	0.457	0.4821

From the table we see that when criterion=“gini”, splitter=“best”, max_features=None, min_samples_split=2 or 3, we get best performance. From the table we see min_samples_split = 3 would be a little better than min_samples_split = 2, but it may be due to chance, so I want to keep all parameters with its default values (criterion=“gini”, splitter=“best”, max_features=None, min_samples_split=2).

Final algorithm

In summary, best algorithm that I find is decision tree (all parameters default) using features deferred_income, exercised_stock_options and expenses.