Using Machine Learning to Find Fraud from Enron Email and Financial Data

Meixian Chen

Project overview

The Enron email database consists of over real 600,000 emails from 158 employees, mostly are senior managers of the Enron Corporation. It was acquired by the Federal Energy Regulatory Commission during its investigation after the company's bankrupt, which was the consequence of willful corporate fraud and corruption. The Enron email database is a precious resource for social studies as well as artificial intelligence research. (https://en.wikipedia.org/wiki/Enron_Corpus)

The original data is available in https://www.cs.cmu.edu/~./enron/.

Explore the dataset

After cleaning and preprocess, the data we have in this project is as following:
It contains 146 points, each point represents a person labeled whether he/she is a person-of-interest (POI).

18 out of 146 persons are POIs.

For each data point, there are 14 financial features, such as ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'],

and 6 email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']

146 persons with 20 features, that makes 2920 fields, and 1358 of the fields (46.5%) are missing values.
deferral_payment, load_adavances, and restrict_stock_deferred are the features with the most missing values (reps. 73.3%, 97.3%, and 87.7% of the values are missing).

Outliers

LOCKHART EUGENE E is not a POI and all his features are missing values, I remove him from the data data.

THE TRAVEL AGENCY IN THE PARK is not a real person, I remove this points as well.

I plot the points by using some combination of features, the distribution of the data looks normal except one point, ‘TOTAL’, which has a very large “total_payment” values comparing to the other points. It is an outlier and I remove it from the data.

Features Selection and Optimization

First (manual features selection), I manually eliminate the features which contains too many missing values. They are deferral_payment, load_adavances, and restrict_stock_deferred mentioned above.

Second (features generation), the percentage of emails from/to/shared a POI shows better how a person related with POIs than the absolute email number. Hence I generate three new features in the following way:

percent_from_poi_to_this_person = from_poi_to_this_person/to_messages
percent_from_this_person_to_poi = from_this_person_yo_poi/from_messages
percent_shared_receipt = shared_receipt_with_poi/to_messages

Now my features_list contains:

```
['poi','salary','total_payments',  'bonus',
    'deferred_income', 'total_stock_value', 'expenses',
    'exercised_stock_options', 'long_term_incentive',
    'restricted_stock', 'director_fees',
    'percent_from_this_person_to_poi',
    'percent_from_poi_to_this_person', 'percent_shared_receipt']
```

Third (features scaling), the email features in the features_list range from 0 and 1, while the financial data have a much larger range (up to millions). In order to achieve a good performance of the classification, I apply min_max_scaler to scale all the features to a range of [0,1].

Fourth(intelligent features selection), I use SelectKBest method to choose the best k features. After the fitting the model with the dataset, we have the scores for all the features (the higher score a feature has, the more powerful it tends to be in predicting if a person is a POI.).

salary 18.575703268
total_payments 8.86672153711
bonus 21.0600017075
deferred_income 11.5955476597
total_stock_value 24.4676540475
expenses 6.23420114051
exercised_stock_options 25.0975415287
long_term_incentive 10.0724545294
restricted_stock 9.34670079105

director_fees 2.10765594328
percent_from_this_person_to_poi 16.6417070705
percent_from_poi_to_this_person 3.21076191697
percent_shared_receipt 9.29623187148

Salary, bonus, stock and how often a person write a POI tend to be the most relevant features in predicting a POI.

To test and tune the algorithm, I explore different values of k (5~12) to evaluate the predicting accuracy. Interestingly, for different classifier, the best parameter varies. We could see the result in the later part of this report.

Fifth (reduce features dimension), I apply PCA to reduce the features dimension in order to achieve better performance of the latter steps. I explore different values of n_components of PCA, (2~5), test and tune them in different combination with the k parameter of SelectKBest method.

Evaluation metric and Validation strategy

I use the scores of Precise and Recall, as well as their harmonic mean F1 to evaluate the classifier method. In this context of this project, Precise is the percentage value of real POIs out of the all POIs the classifier predicts to be; and Recall is the percentage value of real POIs that classifier predicts to be out of all the POIs in the dataset.

For this dataset, I prefer a higher Recall score, which mean a classifier is good to find out suspicious candidates, and then further investigation can be run on the candidates to decide if it is a real POI.

For validation strategy, I use cross validation to evaluate the model. It is important to partition the dataset into training and testing parts. We fit the model with training data and in order to fairly evaluate a model, we use the test data, which the model hasn't seen, to check its predicting ability. Without separating training and testing data, a model tends to tell a better performance than what it can actually do on new data.

I use StratifiedShuffleSplit cross-validation method. In this project, the dataset we are dealing with are small (143 persons). For small dataset, we use k-fold cross-validation (ShufflesSplit) to make the validation more robust. The dataset is also imbalanced (Out of 143 persons, only 18 are POIs). Dividing the dataset into train and test parts, sometime we may unluckily end up with some part with almost none instance of the minority class. "Stratification" is the process of rearranging the data as to ensure each fold is a good representative of the whole (cited from https://stats.stackexchange.com/questions/49540/understanding-stratified-cross-validation). It keeps the percentage of the target class as close as possible to the one we have in the complete dataset.

The First try of different classifiers

I try the following classifier: GaussianNB, AdaBoostClassifier, SVC, NearestCentroid, and RandomForestClassifier.

Without tuning the parameter, their performances are as following:

GaussianNB:
Accuracy: 0.83873	Precision: 0.40974	Recall: 0.47550	F1: 0.44018

AdaBoostClassifier
Accuracy: 0.86267	Precision: 0.48193	Recall: 0.40000	F1: 0.43716

SVC
Precision or recall may be undefined due to a lack of true positive predicitons
(And SVC is very hard to tuning to get a reasonable performance for this dataset)

NearestCentroid

```
Accuracy: 0.83447   Precision: 0.35046   Recall: 0.28300       F1: 0.31314

RandomForestClassifier:
Accuracy: 0.86667   Precision: 0.50000   Recall: 0.18000       F1: 0.26471
```

Among the above classifier, AdaBoostClassifier and RandomForestClassifier are very slow in passing the test from tester.py.

I decide to tuning the parameters of the algorithm which use GaussianNB and NearestCentroid.

Tune the model

Tuning a machine learning model means to try different parameter values from a reasonable range and select the ones which have the best prediction results. For many classifier algorithm, it requires parameter tuning to build a better model for specific dataset.

It is possible to over- or under- tuning a model. Over-tuning a model makes it overfit for certain dataset, but may probably fail to predict a different dataset. Under-tuning means we didn’t explore the best model.

We should use cross-verification method to evaluate the tuned model: fit a tuned model on training data, and evaluate it on a separate testing data. We tune a classifier aiming prediction scores on top of test data. The reason is the same why we do cross-verification.

I use GridSearchCV to automatically explore different combination of values from the parameter grid, and use the one that gain the highest “f1” score in the cross validation.

The parameter grid contains different k values for SelectKBest method, n_components values for PCA, and different metric for NearestCentroid.

For GaussianNB, the best parameters are SelectKBest_k = 12 and pca_n_components = 5.
The performance evaluated by tester.py is
Accuracy: 0.84073	Precision: 0.38219
Recall: 0.31550	F1: 0.34566

Surprisingly, GaussianNB works better alone (Accuracy: 0.83873 Precision: 0.40974	Recall: 0.47550	F1: 0.44018) than combining with SelectKBest and pca_n_components!

For NearestCentroid, the best parameters are SelectKBest_k = 5, pca_n_components = 3, and NearestCentroid_metric = ‘euclidean’.

The performance evaluated by tester.py is
Accuracy: 0.80133	Precision: 0.32336
Recall: 0.44850	F1: 0.37579

(Both Precision and Recall are higher than the requirement (0.3) for this project)

Note, here the score function I use to tune the model is ‘precision’. If I use ‘f1’ as the score function, I manage to get a very high ‘recall’ score with NearestCentroid when SelectKBest_k = 5, pca_n_components = 3, and NearestCentroid_metric = ‘manhattan’.
Accuracy: 0.71087	Precision: 0.28015
Recall: 0.74450	F1: 0.40711
However, the precision score is lower than the requirement 0.3.