P5: Identify Fraud from Enron Email

Udacity Data Analyst Nanodegree

By: Reem Bin-Hezam

Project Overview

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

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?**

In this project, I will play detective, and use my new machine learning skills to build a person of interest (POI) identifier based on financial and email data made public as a result of the Enron scandal.

The dataset contains 146 records of previous Enron employees. It combines emails and finances features about each one of them, including a label stated whether they are POIs or not. After investigating the dataset, I found that 18 out of 146 were labeled as POI. There are actually more POIs than this number, but they were not Enron employees and I don’t have a complete information about them, so I guessed that it would be more accurate for my algorithms not to include them.

Features

The features in the data fall into three major types, namely financial features, email features and POI labels. There are a total of 21 features including the POI label.

financial features: ['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'] (all units are in US dollars)

email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is ‘email_address’, which is a text string)

POI label: [‘poi’] (boolean, represented as integer)

There were missing values for almost all the features, which shown on the below table:

Total number of NaNs for each feature:

Feature	NaNs	NaN %
total_stock_value	20	13.70%
total_payments	21	14.38%
email_address	35	23.97%
restricted_stock	36	24.66%
exercised_stock_options	44	30.14%
salary	51	34.93%
expenses	51	34.93%
other	53	36.30%
to_messages	60	41.10%
shared_receipt_with_poi	60	41.10%

Feature	NaNs	NaN %
from_messages	60	41.10%
from_this_person_to_poi	60	41.10%
from_poi_to_this_person	60	41.10%
bonus	64	43.84%
long_term_incentive	80	54.79%
deferred_income	97	66.44%
deferral_payments	107	73.29%
restricted_stock_deferred	128	87.67%
director_fees	129	88.36%
loan_advances	142	97.26%

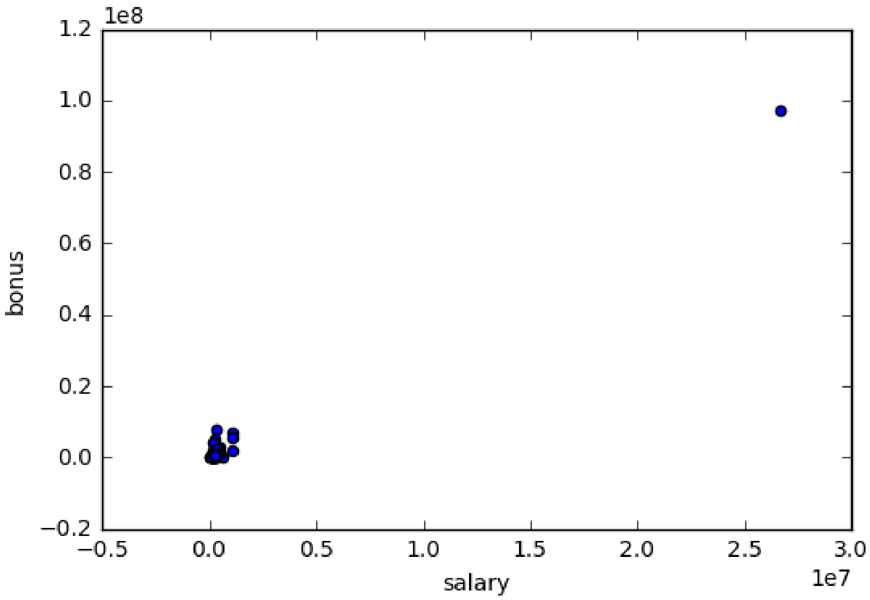
Outliers

I have identified 3 outliers which are:

- TOTAL (a spreadsheet quirk, it’s the total row for each feature column)
- THE TRAVEL AGENCY IN THE PARK (it’s not a person name)
- LOCKHART EUGENE E (100% of NaNs for all features)

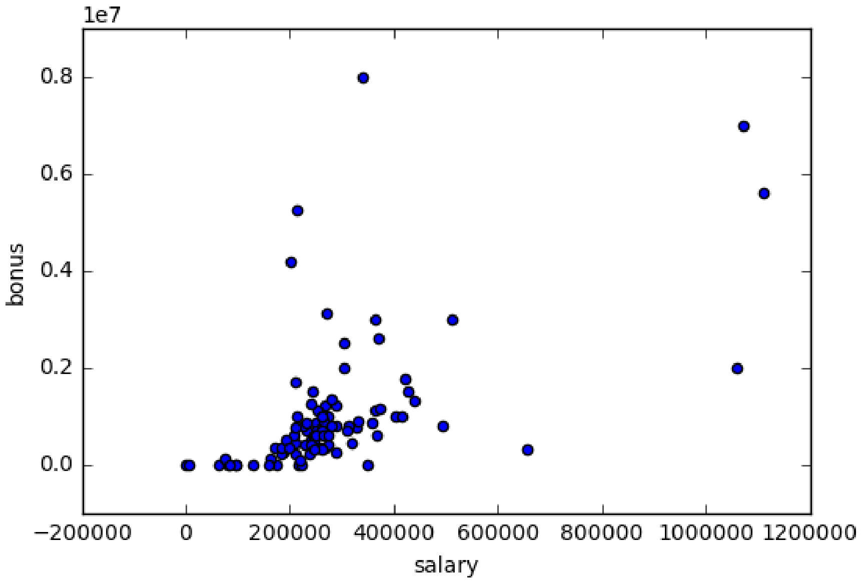
1st Outlier:

The scatterplot visualization of the bonus and salary noticeably shows the biggest outlier on the dataset:



When I go back to the outlier entry in the original data source of Enron financial data table I found that this entry was for the Total raw which was automatically calculated by the spreadsheet! So definitely this was an outlier that should be removed.

However, after I removed it and plot the scatter plot once again, I noticed about 4 more outliers, as below:



As I noticed there are still about 4 outliers, such as LAY KENNETH L and SKILLING JEFFREY K for instance. But these are valid values of Enron's biggest bosses, and definitely they are POIs.

2nd Outlier: I noticed that the last entry on the Enron financial data table was not actually a person, so this might be a typo and I removed it.

3rd Outlier: While I was exploring the dataset, I noticed that LOCKHART EUGENE E contains NaNs for all the features, so this is a useless data point and It should be removed. Besides that, when we convert the NaNs to Zeros, this might affect the results of our classifiers if it was not removed.

Those were the most three noticeable outliers that I could find on the dataset. After removing them, the size of our data become 143 persons only.

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.**

Dealing with Missing Values

NaNs value is handled by the featureFormat() function and is replaced by 0s. However, this could still lead to inaccurate results therefore not all features are going to be included and only the best of them is going to be used in the algorithms. Obviously, features with large number of NaNs are not going to be included.

New Features

I created and engineered three new features (two email feature, and a finance feature):

1. **to_poi_ratio:** the percentage of the number of email messages sent from this person to POI to the total number of email messages sent from this person.
2. **from_poi_ratio:** the percentage of the number of email messages sent from POI to this person to the total number of email messages sent from this person.
3. **bonus_to_salary_ratio:** The ratio of the bonus to the salary of a person.

The 1st and 2nd features were inspired from lesson's Quiz: Feature Selection. As it is obvious, the percentage of the emails sent from or received to POI to the total number of messages leads to more accurate results than just the count of these emails.

The 3rd new feature was inspired from searching about the Enron Fraud case and the fact that it might lead to useful insight. However, keeping or removing these features was kept to the decision of the algorithm that supports getting feature importance I would use.

Selected Features

I used SelectKBest functions in order to select the best 10 features. Before applying the function, I removed the email_address features from the features list since it is the only string feature, and it has no added meaning to the dataset. The other two features I removed were from_this_person_to_poi, and from_poi_to_this_person, since they were replaced by the new features I have just created.

The result was as follow:

#	Feature	Score
1	exercised_stock_options	24.82
2	total_stock_value	24.18
3	bonus	20.79
4	salary	18.29
5	to_poi_ratio	16.41
6	deferred_income	11.46
7	bonus_to_salary_ratio	10.78
8	long_term_incentive	9.92
9	restricted_stock	9.21
10	total_payments	8.77

I scaled features using MinMaxScaler(), because some algorithms such as SVM and k-means needs scaled features.

3. **What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?**

I have tried more than 15 classifiers combinations, and ended up with Gaussian Naïve Bays with its default parameters, using cross validation (Precision: 0.38355 and Recall: 0.31700).

The below table shows the top 5 best results sorted by F1 and F2 values:

Algorithm	Accuracy	Precision	Recall	F1	F2
GaussianNB(priors=None)	0. 84100	0. 38355	0. 31700	0. 34711	0. 32840
KNeighborsClassifier	0. 87073	0. 54880	0. 17150	0. 26133	0. 19884
RandomForestClassifier	0. 85473	0. 39433	0. 16700	0. 23463	0. 18876
KMeans	0. 85533	0. 31360	0. 07150	0. 11645	0. 08456
DecisionTreeClassifier	0. 84353	0. 14807	0. 03650	0. 05856	0. 04298

However, I found a better solution at a github repository for a LogisticRegression classifier with the following parameters:

Pipeline(steps=[('scaler', StandardScaler()),

('classifier', LogisticRegression(tol = 0.001, C = 10**-8, penalty = 'l2', random_state = 42))])

And when I tested it on my data, I get the following results:

Accuracy: 0.84047 Precision: 0.41080 Recall: 0.45250 F1: 0.43064 F2: 0.44350

Although these were better than the GaussianNB ones, but this classifier was not covered on the lessons and I am not very sure about the parameters effects, so I didn't include it as my final results.

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?**

Tuning parameters of an algorithm means: trying to adjust them in order to get the best performance and results. Most algorithms come with default parameters values but this doesn't always give best results. Tuning these parameters could be made either manually (which is exhausting and not practical, or automatically using different methods such as GridSearchCV and piplines.

As my first picked algorithm does not needs any tuning, I made it with the other selected algorithms such as KNeighborsClassifier, KMeans, DecisionTreeClassifier.

For example:

DecisionTreeClassifier:

- GridSearchCV Parameters: parameters = {'min_samples_split':[40,60, 100]}
- Best Parameters: {'min_samples_split': 100}

KNeighborsClassifier:

- GridSearchCV Parameters: parameters = {'leaf_size':[20,30,50], 'n_neighbors':[3,5, 10], 'weights':['uniform', 'distance']}
- Best Parameters:{'leaf_size': 20, 'n_neighbors': 10, 'weights': 'uniform'}

The selected parameters differ from the default ones, meaning that if we kept their default we may not get better results.

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 a way to make sure of our algorithm performance. This is made by separating our data into training and testing sets. Which means trained the algorithm on a subset of the data, and checks its performance on another subset.

A classic mistake is to train and test the algorithm on the same dataset, and by doing this you cannot predict how the algorithm will act when it deals with a new data that has not been seen before.

There are several types of validation, works differently with different type of datasets. A common way is to separate your data into a fixed training and testing sets prior to training, but this might not be possible on relatively small dataset, such as the case on the enron dataset (with 143 records only). Another issue for the enron dataset is the imbalanced class problem, where the number of POI is very smaller than the Not-POI (18 out of 143 only). In this case the cross validation with shuffling is a better choice.

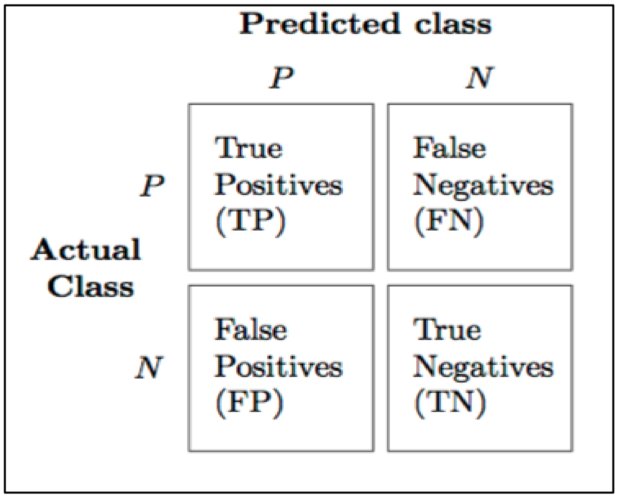
Cross validation is a way to partition the dataset into k different bins and then separates the data into one testing bin and k-1 training bins in a k-fold times, get different k results and take their average.

I used K-Fold CrossValidation (with folds = 1000) and a random_state = 42 (in order to get the same result for each run).

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"]

At the beginning, you might think that the accuracy is the most important metrics, but in fact it's not, because different mistakes yield to different results.

A two very important metrics are precision and recall. Precision is a ratio of how many selected items are relevant, while recall is the ratio of how many selected items are relevant.



		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

$$PRE = \frac{TP}{TP+FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN+TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

Depends on the problem type you might concern more on either precision or recall, or sometimes you need a balance between them, in this case you would concern about the F1 score.

At the case of Enron data:

If the algorithm doesn't have great precision, but does have good recall, it means that, nearly every time a POI shows up in the test set, It would able to identify him or her. The cost of this is that It sometimes get some false positives, where non-POIs get flagged.

If the algorithm doesn't have great recall, but it does have good precision, it means that whenever a POI gets flagged in the test set, It would be known with a lot of confidence that it's very likely to be a real POI and not a false alarm. On the other hand, the price it pays for this is that it sometimes misses real POIs, since it's effectively reluctant to pull the trigger on edge cases.

If the algorithm has a really great F1 score, this is the best of both worlds. Both false positive and false negative rates are low, which means that it can identify POI's reliably and accurately. If the algorithm finds a POI then the person is almost certainly a POI, and if the identifier does not flag someone, then they are almost certainly not a POI.

What I think for this case is that during investigations, a high recall is more important. On the other hand, for court order, being more accurate on identifying POI correctly with a high precision is more important.

For our case, our Naïve Bays model scores were:

Accuracy: 84.10% Precision: 38.35% Recall: 31.70% F1: 34.71%

Total predictions: 15000 TP: 634 FP: 1019 FN: 1366 TN: 11981

References

1. Udacity forums
2. Slack UConnect discussions
3. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#sklearn.ensemble.AdaBoostClassifier
4. http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
5. https://sebastianraschka.com/faq/docs/multiclass-metric.html