Identifying Fraud from Enron Emails Dataset

by Joy Lal Chattaraj

Case Study : 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.

Short Questions

Q1. 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?

The goal of this project was to utilize the financial and email data from Enron to build a predictive that could identify whether an individual could be considered a "person of interest" (POI). The Enron corpus - publicly maed by US Federal Energy Regulatory Comission during its investgiation of Enron, which comprised of email and financial data of 146 people (records), most of which are senior management of Enron. The corpus is widely used for various machine learning problem and although it has been labeled aready, the value is the potential application for similar cases in other companies or spam filtering application. The dataset contained 146 records with 1 labeled feature (POI), 14 financial features, 6 email feature. Within these record, 18 were labeled as a "Person Of Interest" (POI).

While trying to find outliers, I used the `matplotlib` library to plot the points on a scatter plot for fields `salary` and `bonus`. I found an outlier whose value was way higher than others. Upon further inestigation I figured out that the key for it was `TOTAL` which wasn't a valid data point for our dataset. I could figure out another outlier with an unusually long name when I was scanning through the list of keys, it was `THE TRAVEL AGENCY IN THE PARK`.Finally I could figure out another outlier `LOCKHART EUGENE E` that contained only NaN's i.e. it didn't give us any information.

So, Outliers detected are:

- `TOTAL` action taken : key popped out from dictionary as it wasn't a valid datapoint
- `THE TRAVEL AGENCY IN THE PARK` : again I popped the key out of the dict for the same reason
- `LOCKHART EUGENE E` : popped out, didn't contain any data

Q2. 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. In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances 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.

First of all I used `scikit-learn's MinMaxScaler` to scale the features, as many of the financial and email features varied over different ranges, it was important to scale them so that they are valued evenly. There are a few more advantages of feature scaling : 1. It makes your training faster. 2. It prevents you from getting stuck in local optima.

However a few algorithms like decison trees and random forest are not effected by feature scaling.

I looked at a list of NaN's for the dataset as a lot of info was missing. Here is a list of what I found

Feature Name	Number of NaN's
POI	0
SALARY	50
TO_MESSAGES	58
DEFERRAL_PAYMENTS	106
TOTAL_PAYMENTS	21
EXCERCISED STOCK OPTIONS	43
BONUS	63
RESTRICTED STOCK	35
SHARED RECEIPT WITH POI	58
RESTRICTED STOCK DEFERRED	127
TOTAL STOCK VALUE	19
EXPENSES	50
LOAN ADVANCES	141
FROM MESSAGES	58
OTHER	53
FROM THIS PERSON TO POI	58
DIRECTOR FEES	128
DEFERRED INCOME	96
LONG TERM INCENTIVE	79
FROM THIS PERSON TO POI	58

Since the `Loan Advances` feature dosen't give us much information (141 out of 144 are NaN), so I decided to remove it straight away.

Then I used the `SelectKBest` library from sklearn to find the top 12 features according to their scores. Here is a list of scores of the top features

Feature	Score
EXERCISED STOCK OPTIONS	24.81
TOTAL STOCK VALUE	24.18
BONUS	20.79
SALARY	18.29
FRACTION FROM POI	16.409
DEFERRED INCOME	11.45
LONG TERM INCENTIVE	9.9221
TOTAL PAYMENTS	8.7727
SHARED RECEIPT WOTH POI	8.589
LOAN ADVANCES	7.18
EXPENSES	6.09

Feature	Score
OTHER	4.28
FRACTION TO POI	3.12
DIRECTOR FEES	2.12

After this, the scores were really low (below 0.5). Initially I had selected 15 of the best features, but after running the code multiple times on different values of K , I could figure out that 12 was the number of features best suited for my algorithm and I could ignore rest of the features with a lower score. Below are the results:

Number of features vs classifier score

Algorithm	Number of Features	Recall Score	Precision Score
Decision Tree Classifier	15	0.38782	0.47450
Decision Tree Classifier	14	0.38735	0.47450
Decision Tree Classifier	13	0.38839	0.47500
Decision Tree Classifier	12	0.40024	0.50350
Decision Tree Classifier	11	0.41979	0.45400
Decision Tree Classifier	10	0.41955	0.45500
GaussianNB	10	0.36639	0.31400
GaussianNB	12	0.32430	0.31100
Random Forest	10	0.39343	0.23350
Random Forest	12	0.40413	0.21700
Adaboost	10	0.45157	0.32400
Adaboost	12	0.40959	0.31600

The features `faction_from_poi` and `fraction_to_poi` were created by me. The features `to_poi` and `from_poi` didnot make much sense, So I decided to make these features relative, i.e. what fraction of a person's email were sent to a poi and were from a poi. This would neutralize the effect of number of emails sent by user. Suppose a person sends a 100 emails per day sends 5 emails to a poi and another person sends 5 emails per day sent 2 mails to poi. Now earlier the first person would be considered more important but now the second person has a greater chance to be a poi as 40% of his/her mails are sent to a poi. It was a similar case with the feature `fraction from poi`.

Q3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

After trying 5 algorithms, I finally decided to use the `DecisionTreeClassifer` because of the higher recall score. This meant that in a given sample we could predict (correctly) a greater percentage of poi's present in the sample. It was important not to miss many poi's although we could afford to cross examine a few innocent ones. I also tried the following algorithms:

- `Gaussian Naive Bayes` : I got good results without much effort but the result of decision tree was better
- `Support Vector Machine Classifier` : I couldn't find any success here most of the times I got a score of 0.0 (in terms of both precision and recall)
- `Random forest Classifier` : The result was satisfactory but the recall was a bit on the lower side and run time was really high

- `AdaBoost Classifier` : Good result (precision and recall above 0.3) but decision tree did a better job in lesser time

I have included the scores of all the alogrithms mentioned above in the final section of this document.

Q4. 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? .

Tuning the alogrithm is an essential part of Machine Learning. It refers to adjusting the parameters of your algorithm to best fit your data, produce best results with the least effort possible. I not tuned properly the algorithm or classifier may produce an overfit result which is good for the training data but when you try to test it on a different one it fails to produce satisfactory predictions. I is also better to tune your algorithm to use minimum number of iterations to reduce the run time.

While fine tuning my `DecisionTreeClassifier` is first started using `Gridsearchcv` to figure out the best parameters for `criterion`, `max_deth` and `min_samples_split` but it took some time and I wasn't able to get a fair idea on how the algorithm varied the results on changing each parameter. It was then I decided to do the task manually and started with altering `criterion`. I found that using entropy as a criterion produced better results i.e it was better to use information gain of the data points to form the tree. Then I altered the `max_depth` from values 2-30, finally I found that it was the value 2 that gave the best results. I tried tuning other parameters too but that didn't help much so I decided to stop at that point.

Q5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is the process of testing your predictions from the model on the basis of certain metrics and that the model generalizes with the rest of the dataset. One of the classic mistakes if to use the same training data for testing/validation which usually produces a very high accuracy thus it gives us the feeling that our algorithm is doing good whereas we don't know what is the reality.Another mistake is when you divide the dataset into two parts on of which has only one of the labels and the other part has the other set of labels, such classifers aren't well trained and perform exceptionally well on training data whereas fail drastically on the test data.

For Validation I used 30% of the data as test data and trained my classifer on the remaining 70% using `train_test_split`. Then I used the `recall_score`, `precison_score` and `accuracy_score` from sklearn to test its performance.

I also ran the `tester.py` which uses `StratifiedShuffleSplit` to split the data, on my results with (K fold = 1000 times) to get an average of the results.

Q6. 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.

Since the number of POI's is far less than the total number of people in the datset, accuracy can't be used as a measure for evaluation. Thus we consider `recall_score` and `precision_score` for evaluation.

Algorithm	Precision Score	Recall Score	F1 Score
GaussianNB	0.36639	0.31400	0.33818
svm.SVC	-	-	-
DecisionTreeClassifier	0.40008	0.50350	0.45
RandomForestClassifier	0.39343	0.23350	0.29307
AdaBoostClassifier	0.45157	0.32400	0.37729

Thus, I ended up using `DecisionTreeClassifier` as it had better scores than others. In common terms the `DecisionTreeClassifier` can identify 50.3% of the poi's present in the dataset correctly (recall score), i.e. if 100 poi's were present in the dataset it would identify 50 of them . The precision score says that out of every 100 people it says are poi, 39 of them would actually be poi whereas rest of them would be innocent. Since not missing a poi should be our priority we can afford to label a few non-poi's as poi if we can acheive a greater recall score. That is what I have done, I allowed the precision to lower when I saw an increase in recall score. This would ensure that we miss lesser number poi's in a given sample.

The F1 score is a measure that gives us the best of both, i.e. it maintains a balance between recall and precision.