Enron Fraud Analysis Using Machine Learning Algorithms

- Shrey Marwaha
6th November 2017

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 ultimate aim of this project is to analyse the **Enron Fraud dataset** and to identify the **persons of interest**, which were involved in the fraudulent activities that took place at Enron. The dataset provided was a combination of financial and email data which is publicly available, and the culpable persons were already labeled as POIs.

Objective: The objective is to design a classifier, using scikit-learn's algorithms, applying machine learning concepts such as feature engineering, parameter tuning, cross-validation and metrics based evaluation, to finalise an algorithm which does a good job of predicting the POI candidates.

A brief **exploratory analysis** is as follows :

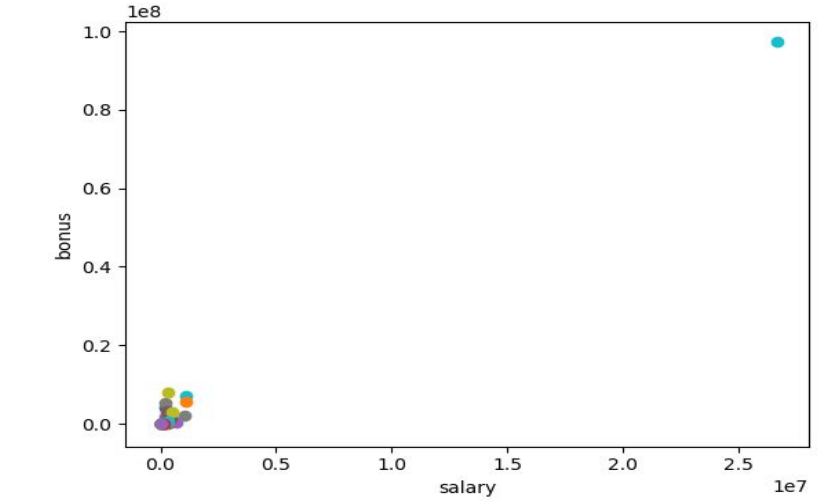
1. The dataset has **146 entries** with details of the employees with their email and financial data.
2. There are **21 features** describing each of the records for respective employees.
3. **18 records labeled as POI** are present in the dataset (~12% only).
4. A lot of the features for respective employees are labeled as NaN, for reasons beyond the scope of the project. The number of such values against each feature is as follows:

```
{'bonus': 64,'deferral_payments': 107,
 'deferred_income': 97,'director_fees': 129,
 'email_address': 35,'exercised_stock_options': 44,
 'expenses': 51,'from_messages': 60,
 'from_poi_to_this_person': 60,'from_this_person_to_poi': 60,
 'loan_advances': 142,'long_term_incentive': 80,
 'other': 53,'poi': 0,
 'restricted_stock': 36,'restricted_stock_deferred': 128,
 'salary': 51,'shared_receipt_with_poi': 60,
 'to_messages': 60,'total_payments': 21,
 'total_stock_value': 20}
```

These were changed as required to 0 for whichever feature required.

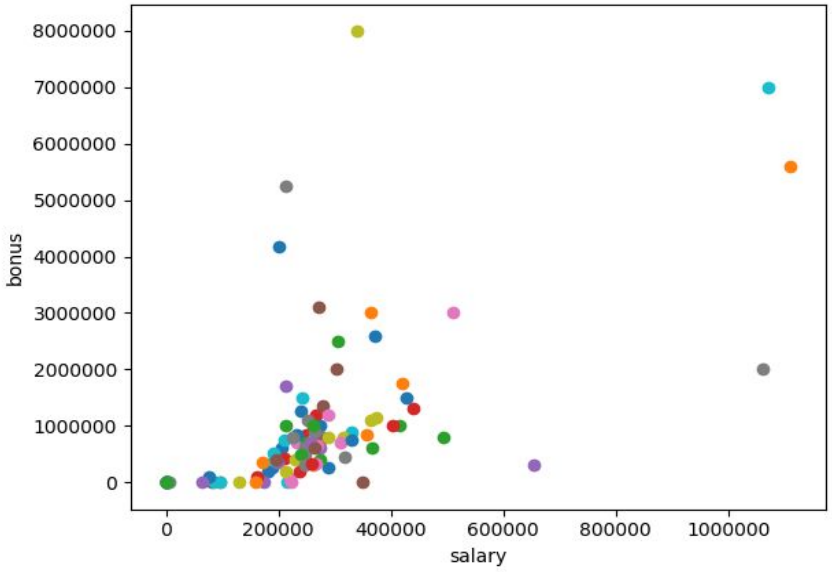
Outliers

Some graphs were also plotted between the features that initially stood out to me, such as payments, stocks, salary and bonus. This helped in outlier detection as well

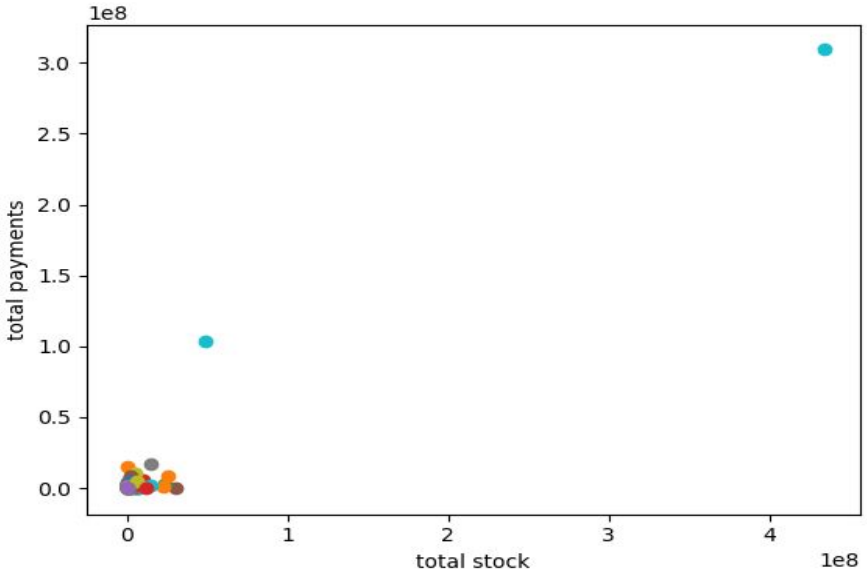


.

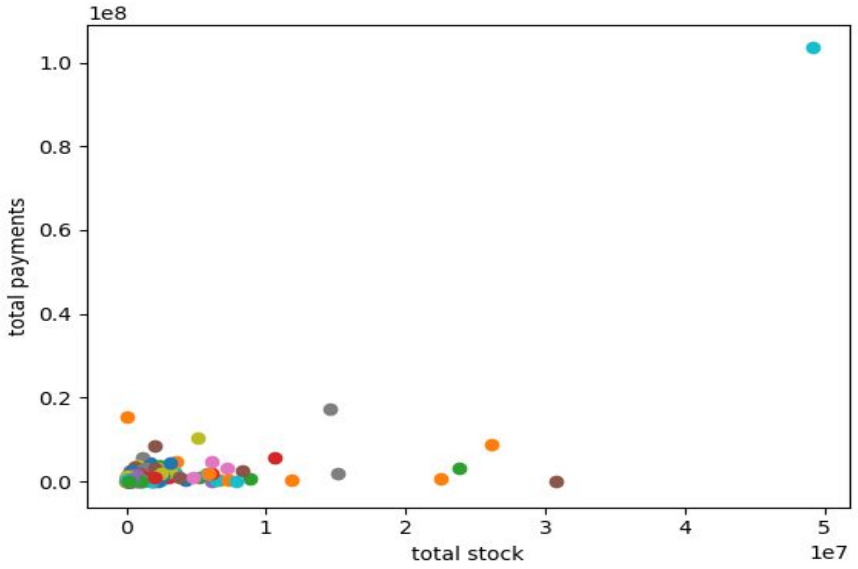
In this figure, a clear outlier is visible, which was later, through code, found out to be a spreadsheet entry of the **'TOTAL'** of all employees. This was later removed, after which the same graph looked much better. The farther points here do signify some of the POI's.



Similar was the case with total payments and total stocks.



After the outlier removal, the graph looked like this:



One point does like an outlier, but is actually a POI, **Kenneth Lay.** The graph explains the unusually high payments and stock that he held.
Points with no data were also removed through the code. This was done by checking the total_payments, total_stock and director_fees features, as they presented a consolidated information regarding the employee financial data.

- **TOTAL** : Was an extreme outlier for financial data.
- **THE TRAVEL AGENCY IN THE PARK** : Record did not represent an Enron employee.
- **LOCKHART EUGENE E** : Record contained no useful data.

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. (You do not necessarily have to use it in the final analysis, only engineer and test 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.

4 new features were engineered from the given feature set, as shown as follows:

```
# Feature 1
total_wealth = total_stock_value + total_payments

# Feature 2
poi_incoming = from_poi_to_this_person / to_messages

# Feature 3
poi_outgoing = from_this_person_to_poi / from_messages

# Feature 4
poi_interaction = (total_poi_interaction / total_interaction) +
(shared_receipt_with_poi / max_shared_receipt_value)

total_poi_interaction = from_poi_to_this_person + from_this_person_to_poi

total_interaction = to_messages + from_messages
```

Total wealth is a feature which encompasses the financial data pretty well, at least the features where we expect the differences between POIs and non-POIs. The other three features were engineered as the original features used in them didn't really have any significance alone, with very low scores in SelectKBest. The **poi_interaction** features is particularly strong as the both ratios of total interaction with POIs and shared receipts in conversations with POIs intuitively point towards the likelihood of being a POI themselves.

Feature selection process was an iterative one. I initially selected some features that I thought were best to be used. This involved intuition to the extent of removing text fields with no relevance, and fields with extremely high NaN values.
Initial feature list that I decided to use, after above said process, was as follows:

```
['poi', 'salary', 'bonus', 'total_payments',
'total_stock_value','exercised_stock_options', 'restricted_stock',
'deferred_income','long_term_incentive', 'shared_receipt_with_poi',
'director_fees', 'from_messages', 'from_poi_to_this_person',
'from_this_person_to_poi','to_messages']
```

After checking the scores, I removed the last 5 features in the above list. The updated list was tested on a classifier which returned the following scores:

Accuracy: 0.79100 Precision: 0.25108 Recall: 0.23350, which was pretty low.

With the addition of the new engineered features, and removal of some old ones, the new feature set with best metrics was :

```
['poi', 'exercised_stock_options', 'deferred_income',
 'shared_receipt_with_poi','poi_outgoing', 'poi_incoming',
 'total_wealth', 'poi_interaction']

Accuracy: 0.85727   Precision: 0.45347   Recall: 0.34350
```

But ultimately, upon reducing another feature by score (poi_incoming), I achieved the best metrics, which were as follows:

Accuracy : 0.86853 ***Precision: 0.50894*** ***Recall : 0.0.39850***

*T*he scoring was done through **SelectKBest** algorithm. During this process, the **MinMaxScaler** was used as well**,** as most of the features were financial and hence very large in number, while the email related ones weren't.

(Red -> Removed, Green -> Chosen in final list, NA -> Doesn't exist)

Feature	Feature Scores (Iter-1)	Feature Scores (Iter-2)	Feature Scores (Iter-3)
exercised_stock_options	24.81	24.81	**24.81**
total_stock_value	24.18	24.18	COMBINED (total_wealth)
total_payments	8.77	8.77	
bonus	20.79	20.79	Accounted in total_wealth
salary	18.28	18.28	
deferred_income	11.45	11.45	11.45
long_term_incentive	9.92	9.92	NA
restricted_stock	9.21	9.21	NA
shared_receipt_with_poi	8.58	8.58	8.58
from_poi_to_this_person	**5.24**	NA	NA
from_this_person_to_poi	**2.38**	NA	NA
director_fees	**2.12**	NA	NA
from_messages	**1.64**	NA	NA
to_messages	**0.16**	NA	NA
total_wealth	NA	NA	16.99
poi_incoming	NA	NA	**3.12**
poi_outgoing	NA	NA	16.40
poi_interaction	NA	NA	10.30

Hence the final list used was :

```
['poi_outgoing', 'total_wealth',
 'shared_receipt_with_poi', 'exercised_stock_options',
 'poi_interaction', 'deferred_income']
```

Clearly, the impact of adding new features, as well as removing redundant ones, is visible through the better metrics, as well as good feature scores of the newly engineered features.

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

The final algorithm that I used was **Adaboost Classifier**. Along with that I tried *Naive Bayes* and *Decision Tree* Classifiers. The initial performance between these three without any tuning was as follows:

Algorithm	Accuracy	Precision	Recall
Naive Bayes	0.86953	0.51603	0.34600
Decision Tree	0.83547	0.36689	0.32250
AdaBoost	0.86853	0.50894	0.0.39850

I decided not to go with Naive Bayes because the number of POIs in the dataset was very low, and hence having a higher accuracy was most probably due to correct prediction of the negatives.
So finally, I decided to use Adaboost Classifier, by seeing the Precision and Recall scores, as Accuracy is not very reliable in case of our dataset, which has very low POIs (positives).

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?

Parameter tuning in Machine Learning involves finding the appropriate values of the various parameters available for an algorithm, to achieve best possible performance, while maintaining a reasonable computational time. Fundamentally, it is just changing some variables to optimize a model.
If not tuned, let alone tune properly, we are forced to use the default values, which may lead us to miss out on model performance and accuracy, and increase the time taken. Tuning the model wrong on the other hand, may simply break your algorithm, or cause it to take a lot of time, leading to wastage of time and resources.
And lastly, each dataset requires different tuning parameters to obtain the best fit and predictions. So using the default value is just the beginners way. For better performance, tuning is necessary.

For my algorithms, I used **GridSearchCV** alongside the concept of **Pipeline**, to get the best out of my choices. Initially, I used a **Min-Max Scaler**, PCA, and then the Classifier in this order. But with the addition of PCA, the performance got degraded for both Decision tree and Adaboost, so it was removed during tuning.
Other parameters were supplied as a dictionary, to the GridSearchCV instance, and it returned the best parameters through the ***best_params_*** and the classifier as ***best_estimator_***

Final chosen parameters obtained through GridSearchCV are:

```
Chosen parameters :

{'clf__learning_rate': 1, 'clf__algorithm': 'SAMME',
'clf__random_state': 42, 'clf__n_estimators': 80}
```

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 our model against previously unseen data, and then evaluating various metrics to gauge its performance. Various metrics can be used for evaluation, and it depends on the kind of data which metric should be used.

This is where a mistake can be made. If the same data that has been used for training (while fitting the classifier) is used for validation, it will lead to **Overfitting**, which is a classic mistake. Due to this, the model which will perform well on already seen data will have low ability to adapt to unseen data, hence perform poorly.

The model in this project is validated using the provided testing script, which makes use of a cross validation method called **StratifiedShuffleSplit** to split the data into testing and training datasets. It returns the indices for testing and training data points separately. The main feature here is **Stratification**, seeking to ensure that each class is (approximately) equally represented across each test fold and training fold. This is especially important here since POI class is very small in number in our dataset, which is an imbalanced one. Stratified split ensures a percentage of each class in the test and training sets, which is close to original. A direct split would lead to biased results.

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.

The two metrics used were **Precision and Recall.**
Precision describes the ratio of how often our model correctly identifies a positive label, (POI = 1), to the total times it guesses a positive label(all POI = 1 predictions). This includes correct positive predictions plus wrong positive predictions.

Recall on the other hand describes the ratio of correct positive labels identified to all the records flagged as POIs. This includes correct positive predictions plus wrongly identified negative labels (predicted as non-POI, when it was originally a POI)

.

Final scores were as follows:

Precision : ***0.51128***
Recall : ***0.37400***

```
Accuracy: 0.86887     Precision: 0.51128
Recall: 0.37400  F1: 0.43200     F2: 0.39522
Total predictions: 15000   True positives:  748
False positives:  715 False negatives: 1252
True negatives: 12285
```

Conclusion

Most algorithms prove to be accurate and have high performance in balanced datasets, which this one is definitely not. Due to a very small number of positives (people who are POIs), getting a better metric with the most optimal scores is quite tough and hence the model becomes complex.

References

- https://machinelearningmastery.com
- http://scikit-learn.org/stable/index.html
- https://stackoverflow.com/
- http://www.ritchieng.com/machine-learning-ensemble-of-learners-adaboost