P5: Identify Fraud from Enron Email

Michail Kovanis

Understanding the dataset and question

The objective of this project was to explore the dataset of emails by Enron employees and executives and to use machine learning to identify persons of interest (POI) who were involved in the fraud. Machine learning is a powerful tool that, when used appropriately, can help uncover hidden patterns in data and make predictions, thus it was useful for completing this project's objective.

The dataset contains as keys the names of the Enron employees and chairmen and consists of 16 financial features (salary, bonus etc.), 6 email features (email addresses, number of emails sent to a poi, etc.) and a POI label. Out of 146 keys there are two which don't refer to a person ("TOTAL", "THE TRAVEL AGENCY IN THE PARK"). Every feature does not contain data on every person. The 'total_payments' and 'restricted_stock' contain the most data (125 and 110 data points respectively) and the "loan_advances" and "director_fees" contain the least (4 and 17 respectively). POIs are marked with 1's and non-POIs with 0's and this information is known for every person in the dataset. Moreover, the "TOTAL" key obviously contains only outliers and will not be considered in the analysis. Finally, almost all financial features seem to have at least one outlier value (apart from "director_fees" and "bonus"). For example, the maximum value of 'restricted_stock_deferred' is about 350 times higher than the previous one. These values were retained in the final analysis, because I managed to achieve the minimum precision and recall values required without removing them.

Optimize Feature Selection/Engineering

I selected to use the following features: 'salary', 'bonus', 'exercised_stock_options', 'long_term_incentive', 'from_this_person_to_poi', 'from_poi_to_this_person' and 'shared_receipt_with_poi'. In addition, I used the sum of salary and bonuses for a person as a new feature called 'earnings'. The 'earnings' feature might be useful in the case that a POI was compensated for his/her involvement with a high bonus instead of a salary or the opposite. I was also planning to use the 'total_stock_value' but it has an almost linear relationship with 'exercised_stock_options', thus I chose to use only the second. The rest of the financial features were selected because they show the wealth of each person, which I expect to be correlated with one's involvement in the fraud. Finally, I used the number of emails received from/sent to/shared with a POI as a proportion of total emails sent or received by each person.

I used SelectKBest to select the k = 3 best features to feed to the classifiers. The feature scores for each feature were the following:

Feature	Salary	Bonus	Earnings	Exercised stock options	Long term incentive	% from this person to poi	% from poi to this person	% shared receipt with poi
Score	14.58	17.33	18.74	21.15	7.99	13.41	2.10	6.60

The three best features were the ‘exercised_stock_options’, 'earnings' and 'bonus'. These features were used to fit to all the classifiers.

Pick and tune an algorithm & Validate and evaluate

I used five different algorithms; the naïve Bayes, decision tree, random forest, adaboost and k nearest neighbours. I created the respective classifiers, scaled the features (necessary for k-nearest neighbours and to improve running time for the rest), fitted the data for all the previously mentioned features and used the tester.py script to assess and compare their performance regarding accuracy, precision and recall. In this dataset an algorithm could theoretically produce high accuracy without identifying any POI, just because the majority of labels are equal to 0. For that reason it was very useful to use precision and recall; the first to measure the proportion of true POI occurrences among all those flagged as POIs by the algorithm and the second to measure the proportion of true POI occurrences among all those correctly flagged as POIs and those that should have been flagged but weren't. In his case I was interested to identify as many POIs as possible and thus recall is an especially useful metric for that.

In addition, the results should be validated in order to avoid cases that the classifier overfits to the training dataset and doesn't produce results that can be generalised to other datasets of the same type. For this reason I split the data into a training and a testing dataset. This way I trained the classifiers to a portion of the complete dataset and used the remaining datapoints to test the results. I used stratified splitting (test_size=0.1) to ensure that both sets will include POIs and non-POIs. Below, I present the results for accuracy, precision and recall of all algorithms in their default parameter settings as given by tester.py.

Metric	Naive Bayes	Decision Tree	Random Forest	AdaBoost	K Nearest Neighbours
Accuracy	0.824	0.792	0.840	0.805	0.869
Precision	0.396	0.316	0.441	0.330	0.702
Recall	0.271	0.306	0.139	0.261	0.254

In the default setting only the Decision tree achieved both precision and recall higher than 0.3, whereas k nearest neighbours had the highest accuracy. However, I wanted to see whether I could achieve higher performance by changing the parameters of the classifiers. Algorithms might overfit or underfit if the classifiers are not appropriately tuned. Thus tuning the classifiers can reveal an algorithm's best performance and avoid fitting issues.

Below, I present the different parameter settings I tried for all algorithms (apart from Naive Bayes) and the results for the best performing setting.

	Decision Tree	**Random Forest**	**AdaBoost**	**K Nearest Neighbours**

Par/ters tried	min_samples_split=[2,4]	n_estimators=[10,20]
		min_samples_split=[2,4]
			learning_rate=[1,2]
			n_estimators=[50, 100]
				n_neighbors=[1,2,5]
				p=[1, 2, 5]

Best par/ters	min_samples_split=2	n_estimators=20
		min_samples_split=4
			learning_rate=1
			n_estimators=50
				n_neighbors=1
				p=2

Accuracy	0.792	0.839	0.805	0.804
Precision	0.316	0.438	0.330	0.372
Recall	0.306	0.161	0.261	0.400

From all the results it is apparent that the k nearest neighbours algorithm produced the best results in terms of precision and recall.

References

Enron case: https://en.wikipedia.org/wiki/Enron

Enron dataset: https://www.cs.cmu.edu/~./enron/

Sklearn documentation: http://scikit-learn.org/stable/documentation.html

Ravel: http://stackoverflow.com/questions/34165731/a-column-vector-y-was-passed-when-a-1d-array-was-expected