Identify Fraud from Enron Email

Jerome Vergueiro Vonk

April 13, 2018

1 Goal

The goal of this project is to identify, using machine learning, *Enron* employees who may have commited fraud based on the public Enron financial and email dataset. Machine learning is useful in this context because there are several classification algorithms that can "learn" from the data and then find what class a sample belongs to.

Quick overview of the data:

- There are 146 people on this dataset
- Of those, 18 are considered *Persons of Interest*. This is a hand-generated list of individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.
- There are 21 features on the dataset, which fall into three major types: financial features (salary, bonus, stock options, etc), email features (number of emails sent/received to other POIs, etc) and POI labels.

There were some problematic data points which were removed before starting the analysis:

- Someone named Eugene E Lockhart had missing values for all features but the POI label. Since he was not a person of interest himself, I discarded him.
- One data point was named "THE TRAVEL AGENCY IN THE PARK". It was identified by manually looking at the names for all the people in the dataset. Since it was not a person, it was removed from the database before analysis.
- There was one data point named "TOTAL". This was a sum of every column from the financial table. It was identified because it produced outliers in scatter plots of financial data. Since it is not a person, it was removed.

This is a snippet of the code that found a data point missing values for all features:

```
for person in enron_data:
        valid = False
        for feature, feature_value in enron_data[person].iteritems():
                if feature_value != 'NaN' and feature != 'poi':
                        valid = True
                        break
        if valid == False:
                print 'Found someone with NaN for all values: ', person
```

2 Features

2.1 New features

I created two new features from existing ones. They are:

- **fraction_from_poi**: The number of messages sent from a POI to this person, divided by the total number of messages this person received.
- **fraction_to_poi**: The number of messages sent from this person to a POI, divided by the total number of messages this person sent.

Sending (or receiving) a message to (from) a person of interest does not make one a person of interest himself, but perhaps if someone communicates with a POI too often, then he could be a possible POI as well.

One of these created variables proved to be effective in identifying POIs and was consider by the selection algorithm as the fifth best, as we can see in the next section. Since the final list of important features, shown in section 4, is composed by six features, **fraction_to_poi** made it into the final list, proving it has **a positive impact on precision and recall**.

2.2 Feature selection

Considering all features but e-mail address as candidates, I ran the **SelectKBest** univariate feature selection algorithm, provided by *Scikit-learn*, using *F-test* as the scoring function. The picture below shows the score along with the percentage of missing values for each feature.

Feature	F-Value	Missing %	Feature	F-Value	Missing %
exercised_stock_options	24.81507973	0.301	expenses	6.09417331	0.349
total_stock_value	24.18289868	0.137	from_poi_to_this_person	5.24344971	0.411
bonus	20.79225205	0.438	other	4.18747751	0.363
salary	18.28968404	0.349	fraction_from_poi	3.12809175	0.411
fraction_to_poi	16.40971255	0.411	from_this_person_to_poi	2.38261211	0.411
deferred_income	11.45847658	0.664	director_fees	2.1263278	0.884
long_term_incentive	9.92218601	0.548	to_messages	1.64634113	0.411
restricted_stock	9.21281062	0.247	deferral_payments	0.22461127	0.733
total_payments	8.77277773	0.144	from_messages	0.16970095	0.411
shared_receipt_with_poi	8.58942073	0.411	restricted_stock_deferred	0.06549965	0.877
loan_advances	7.18405566	0.973			

We can clearly see that some of the features wouldn't help at all in the classification. This is the case, for example, of *director_fees* and *restricted_stock_deferred*, in which about 88% of the data points had missing values.

This was only a first stage of feature selection. With the features selected in this process, I later tried six different classifiers to see which one would lead to better results. I kept in mind that, after choosing one classifier, I could run the selection algorithm again in an exhaustive way to optimize the results. Also, some algorithms, like *Decision Trees* and *Random Forest*, have a maximum number of features as parameter, so there could be another selection on the process.

For this stage, I decided to go with the default value of 10 features. Of the chosen, eight come from the financial data and two come from the email data, one of them (*fraction_to_poi*) being a feature created by me.

3 Algorithms

I tried six classification algorithms with minimal parameter tuning. The only exception on this list was SVC, which showed very poor scores with linear kernel and default parameters. Because of not being affine transformation invariant, it was also the only algorithm that showed better results (regarding precision and recall) after feature scaling (*Scikit-learn*'s **MinMaxScaler** was used). These are the performances obtained:

Algorithm	sklearn method	Feature Scaling?	Accuracy	Precision	Recall	F1	F2
Naive Bayes	GaussianNB	No	0.836	0.366	0.314	0.338	0.323
SVM	SVC (RBF kernel, C=100000)	Yes	0.784	0.195	0.200	0.198	0.199
Decision Tree	DecisionTreeClassifier	No	0.824	0.331	0.311	0.321	0.315
K Nearest Neighbours	KNeighborsClassifier	No	0.867	0.503	0.245	0.330	0.273
AdaBoost	AdaBoostClassifier	No	0.838	0.364	0.284	0.319	0.297
Random Forest	RandomForestClassifier	No	0.865	0.470	0.124	0.196	0.145

Out of the box, **KNearestNeighbours** showed the best score regarding accuracy, but also showed imbalance between precision and recall. Both **Naive Bayes** and **Decision Tree** achieved better than 0.3 precision and recall, which is what we are looking for in this analysis.

4 Parameter tuning

It is crucial, in every machine learning process, to find an optimal set of values for the parameters that a particular algorithm accepts. It is important to emphasize that the default parameters are not always the best option, and that optimized performance can be achieved by understanding the data you are working with so that you can infer what parameters, when tunned, can present better results.

For example, the **Support Vector Machines** algorithm, with parameters 'linear' for *kernel* and C = 1 showed very poor performance regarding precision and recall:

Algorithm	sklearn method	Accuracy	Precision	Recall
SVM	SVC (linear kernel, C = 1)	0.862	0.013	0.001

The **Naive Bayes** does not have parameters to be tuned, but we can run an exhaustive feature selection to see if we can do better than the scores we obtained in last section. That was done by varying the k parameter of **SelectKBest** in the range from two to twenty one, i.e, all the features available. The obtained results follow. I highlighted the maximum value for accuracy, precision and recall.

Features	Accuracy	Precision	Recall
2	0.841	0.469	0.268
3	0.843	0.486	0.351
4	0.847	0.503	0.323
5	0.856	0.495	0.327
6	0.861	0.516	0.386
7	0.854	0.487	0.380
8	0.854	0.486	0.396
9	0.841	0.384	0.317
10	0.836	0.366	0.314
11	0.822	0.326	0.312
12	0.822	0.324	0.311
13	0.822	0.325	0.311
14	0.814	0.304	0.310
15	0.814	0.304	0.310
16	0.816	0.310	0.310
17	0.778	0.248	0.329
18	0.771	0.230	0.306
19	0.770	0.214	0.271
20	0.772	0.215	0.268
21	0.739	0.226	0.395

We observe that the maximum accuracy and precision scores were obtained with the six best features selected, while the best recall score was obtained when eight features were considered. The score for **Naive Bayes** with six features turned out to be the best score I got testing different classifiers and parameters. It is also relevant to observe how adding more features indiscriminately will only result in worse scores.

Therefore, the optimal set of features found was consisted by: **exercised_stock_options, total_stock_value, bonus, salary, fraction_to_poi and deferred_income**

Decision Tree, in the other hand, does have some parameters that we can tune. Taking advantage of the exhaustive search that **GridSearchCV** provides, I tuned the following parameters:

- **criterion** - The function to measure the quality of a split. Tested with values *'gini'* and *'entropy'*.
- **min_samples_split**: - The minimum number of samples required to split an internal node. Tested with values '2' and '10'.
- **max_features**: - The number of features to consider when looking for the best split. Tested with values '6', '8' and '10'.

The best estimator, as chosen by **GridSearchCV**, selected the parameters as: *criterion* = *'gini'*, *min_samples_split* = 2, *max_features* = 6. The scores were: Accuracy = 0.84213, Precision = 0.40108, Recall = 0.37300.

5 Validation

Validation is the process of deciding whether the numerical results obtained are acceptable as descriptions of the data. One classic mistake is to train and test the machine learning algorithm with the same data. What happens in this case it that you would obtain an illusory high evaluation score because the model was overfitted, meaning it fits too well to this particular data but would fail to predict future observations.

Given that the dataset for this project is considerably small, we wouldn't get good results by simply splitting the train and test data on some ratio (say, 70%/30%). I've used a cross-validation technique available in *Scikit-learn* called **StratifiedShuffleSplit**. It shuffles the data, splits the data into train and test in the 90%/10% ratio and repeats this process the desired number of times (in this case, it was 1000 operations).

6 Evaluation

There are different evaluation metrics that can be applied in machine learning, but in this project we are aiming to maximize the precision and recall scores. Precision answers the question: "How many selected items are relevant?" and recall answers the question: "How many revelant items are selected".

For the best classifier (**NaiveBayes** applied on the six best features) I obtained the following scores:

- **Accuracy**: 0.861
- **Precision**: 0.516
- **Recall**: 0.396

What these numbers say about performance in this analysis is:

- With respect to accuracy, the algorithm correctly classified a person 86.1% of the time.
- Regarding precision, when the algorithm says that someone is a *person of interest*, there's 51.6% of chance that the person was involved in fraud.
- Concerning recall, we affirm that 39.6% of the *persons of interest* have been identified by the algorithm.