Identify Fraud from Enron Email

Data Analyst Nanodegree – project 5

Lukasz Korona

The following report is an attempt to build a classifier that will help to group Enron employees based on their participation (or lack thereof) in the fraud that resulted in the collapse of the company. The project uses the dataset that characterizes specific employees by both their financial characteristics and the history of their internal, e-mailing correspondence.

1. Dataset characteristics and processing

First of all, I would like to have a look at the available data. It appears that it contains 146 records.

```
In [1]: print len(data_dict.keys())
146
```

By picking one element (as the structure of the data shows that we have a dictionary of dictionaries), I can also establish what kinds of traits are included in the dataset, as well as how many characteristics are initially in the dataset.

```
In [2]: print len(data_dict[‘LAY KENNETH L’].keys())
21

In [3]: print data_dict['LAY KENNETH L']
{'salary': 1072321, 'to_messages': 4273, 'deferral_payments': 202911, 'total_payments': 103559793,
'exercised_stock_options': 34348384, 'bonus': 7000000, 'restricted_stock': 14761694, 'shared_receipt_with_poi':
2411, 'restricted_stock_deferred': 'NaN', 'total_stock_value': 49110078, 'expenses': 99832, 'loan_advances':
81525000, 'from_messages': 36, 'other': 10359729, 'from_this_person_to_poi': 16, 'poi': True, 'director_fees':
'NaN', 'deferred_income': -300000, 'long_term_incentive': 3600000, 'email_address': 'kenneth.lay@enron.com',
'from_poi_to_this_person': 123}
```

As we can see, there are 21 characteristics in the initial dataset. Most of them (except for *‘poi’* and *‘email_address’*) are of integer type, numbers without decimal places. The *‘email_address’* is of string type, while *‘poi’* is Boolean and can only have two values, either True or False. The *‘poi’* characteristic will be used as a label for the algorithm built and fine-tuned within this project.

A few e-mail features also show the frequency of contact with Persons Of Interest, either as a sender, receiver or a mutual receiver. Two more characteristics were added at this

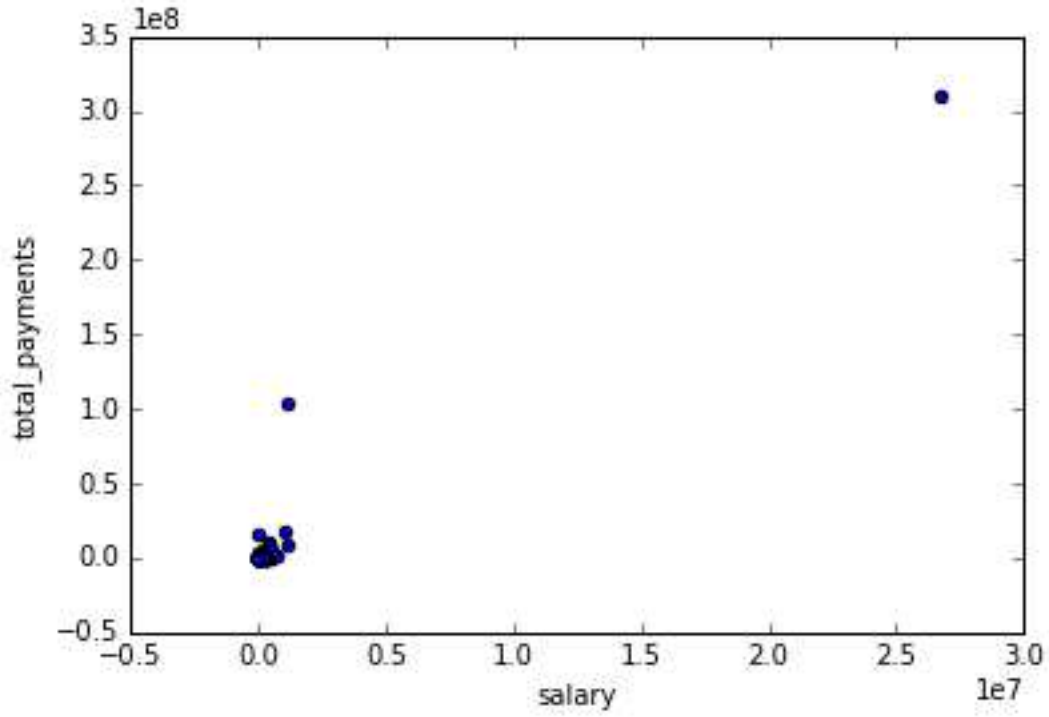
point; although the number of messages to and from POIs are valuable pieces of information, they ignore the relative scale of corresponding with potential culprits. Percentages of messages to POIs and from POIs were added to be checked for outliers.

Before searching for outliers, I should also look at the number of *NaN* values for each column. Using features that have many of them (by default, the function that transforms the data will turn those into zeros) will have a negative effect on the classifier and can skew the results in a very drastic way, making the classifier virtually useless. A new function was written that can be replicated for any dictionary of dictionaries. It loads such a structure and returns a list of tuples, where each key of a sub-dictionary has the number of *NaN* values assigned to it. The function produced the following results:

[('loan_advances', 142), ('director_fees', 129), ('restricted_stock_deferred', 128), ('deferral_payments', 107), ('deferred_income', 97), ('long_term_incentive', 80), ('bonus', 64), ('to_messages', 60), ('from_poi_to_this_person', 60), ('shared_receipt_with_poi', 60), ('from_messages', 60), ('%topoi', 60), ('%frompoi', 60), ('from_this_person_to_poi', 60), ('other', 53), ('expenses', 51), ('salary', 51), ('exercised_stock_options', 44), ('restricted_stock', 36), ('email_address', 35), ('total_payments', 21), ('total_stock_value', 20), ('poi', 0)]

We can see that some variables have almost only *NaN* values (e.g. *loan_advances* or *director_fees*). On the other hand the *poi* label does not have any *NaN* values.

The next step in analyzing the data was to identify and, if necessary, remove outliers. Three records were excluded. The first one was spotted by creating a scatterplot. One column *'total_payments'* had an unusual data point, shown below:



It was identified by using a different function, which produced the following results:

```
[('TOTAL', 309886585), ('LAY KENNETH L', 103559793), ('FREVERT MARK A', 17252530),
('BHATNAGAR SANJAY', 15456290), ('LAVORATO JOHN J', 10425757)]
```

We can see that the biggest value has the *'TOTAL'* key, which points at a data quirk; the record with this key should be removed.

The second one was erased manually, as the key in the dictionary did not point at a specific person, but rather a business entity: *'THE TRAVEL AGENCY IN THE PARK'*.

The third one contained only *NaN* values, as shown in the print-out below:

```
In [16]: print data_dict3['LOCKHART EUGENE E']
{'to_messages': 'NaN', 'deferral_payments': 'NaN', 'expenses': 'NaN', 'poi': False, 'deferred_income': 'NaN',
'email_address': 'NaN', 'long_term_incentive': 'NaN', 'restricted_stock_deferred': 'NaN',
'shared_receipt_with_poi': 'NaN', 'loan_advances': 'NaN', 'from_messages': 'NaN', 'other': 'NaN', '%topoi': 'NaN',
'director_fees': 'NaN', 'bonus': 'NaN', 'total_stock_value': 'NaN', 'from_poi_to_this_person': 'NaN', '%frompoi':
'NaN', 'from_this_person_to_poi': 'NaN', 'restricted_stock': 'NaN', 'salary': 'NaN', 'total_payments': 'NaN',
'exercised_stock_options': 'NaN'}
```

By removing three data quirks, the number of records was reduced to 143.

The last challenge stemming from the structure of the data is connected with its lack of balance. Only 18 records have a value *True* / 1 for the *poi* column, while the remaining people were not indicted or given immunity for their testimony.

2. Feature selection

It was first hypothesized that not all features will be useful per se; some of them might only be useful if combined with others and turned from their absolute values to their percentages (that is why two additional characteristics were built). First of all, the variables *from_messages* and *to_messages* do not intuitively give us any indication as to whether the person might be of interest to law enforcement in connection with fraudulent activities. Some jobs (for example, human resources or internal communications positions) will require a more vivid e-mail activity than others. Secondly, even if a person sends and receives many messages, it might be connected to people who have not been known to be participants of the massive fraud.

Secondly, I wanted to exclude those features that have a large number of *NaN* values. The helper functions have a built-in way to change *NaN* values into zeroes, however, in some cases it might be better to get rid of them altogether due to a very significant number of *NaNs*, which can distort the results. Based on the findings from the first section and to have the best data possible, all variables that had more than 60 *NaN* values were arbitrarily excluded from the list of features; the newly-created variables remain, as the number of *NaNs* for each of them was exactly 60.

Thirdly, the ‘email_address’ variable was excluded, as it was of string type and did not intuitively give much information about the investigated person.

After taking three steps, we were left with the following list of features:

analyzed_features_list = ['poi', 'salary', 'total_payments', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'restricted_stock', 'shared_receipt_with_poi', '%frompoi', '%topoi']

The number of features was still too big relative to the size of the dataset. The first intuition was to limit the number of features to four most relevant ones, as a potential starting point, and go back to this step if this proved to be a cause of overfitting or other problems, such as low evaluation metric values. At this point, the features were no longer chosen based on NaN values or their intuitive relevance, but rather on *SelectKBest* algorithm and *feature_importances_* function of the decision tree classifier. A function was created to select a specific number of features, based on the provided dataset; as the dataset is small, the task of choosing best features was looped a hundred times, each time with a different data split random state. For each loop, selected features’ count in a resulting dictionary was updated, and the result was as follows:

{'%frompoi': 77, '%topoi': 3, 'exercised_stock_options': 90, 'expenses': 3, 'other': 2, 'restricted_stock': 27, 'salary': 86, 'shared_receipt_with_poi': 9, 'total_payments': 10, 'total_stock_value': 93}

Four features with the highest number of occurrences were included as the final features for this feature selection method, including one new feature, *‘%frompoi’*.

The second selection method was based on decision tree algorithm’s feature importances attribute. As it uses a different features selection method that is more suited to DT classifiers, it was worth noting if the results would be different. Another function was written to split the data a hundred times and get, out of all those splits, the average importance of each feature was calculated, as documented below:

{'salary': 0.04651609653469989, 'total_payments': 0.04162133930190159, '%topoi': 0.03573323677331521, '%frompoi': 0.19413475018367576, 'total_stock_value': 0.060252649979678184, 'expenses': 0.1459459487963054, 'exercised_stock_options': 0.15953048283431387, 'other': 0.13284885473573474, 'shared_receipt_with_poi': 0.13488144006171257, 'restricted_stock': 0.048535200798662484}

The results are different, as we can see that *expenses* join *exercised stock options* as financial features, whereas *%frompoi* and *shared_receipt_with_poi* are now the most important e-mail features.

3. Algorithms, cross-validation and evaluation

Two algorithms have been chosen for building a classifier and they will be compared to each other. The first one will be Gaussian Naïve Bayes and the second a decision tree classifier, trained on two different sets of parameters and fine-tuned.

1. Validation / cross-validation – importance, usage

The basic procedure in supervised machine learning algorithms is to split the data into training and testing subsets. This should enable us to "test" the algorithm on an independent (testing) set after fitting it on the training set. In general, validation should help us ensure that the algorithm, once fitted, will have good predicting power on the real-life data that has not been included in the experiment or at least in its training part.

The problems arise when the dataset is relatively small. Splitting the data just once may not help us achieve the result we wish, as we might lose some predicting capabilities. In the case of this project, we might for example put only non-POIs in the training set; our predicting power will then be non-existent with regard to people actually involved in the fraud. This is especially problematic every time we have such an unbalanced dataset, with a relatively low number of persons of interest. One other problem that may arise is called *overfitting*, where our model achieves rather poor predictive performance, in plain words, it does not generalize well from the training set. One should be especially careful about overfitting with small datasets combined with a large number of selected features.

To limit the aforementioned problems, cross-validation should be employed. There are different kinds of this technique, such as *k-fold cross-validation*, *leave-p-out cross-validation* etc., but they all can be summarized in a similar way. The dataset is split into training and testing set more than one time. Each time, the training (and so the testing) sets will contain different observations. The predictions are then averaged from multiple data splits.

In the case of the tester function contained in the project, we use Stratified Shuffle Split. This technique is particularly useful for the dataset that we have to deal with – small and unbalanced. The splitting procedure is iterated one thousand times (not all splits will always be different), and each time the class frequencies, so the proportions of POIs and non-POIs, will be approximately preserved. By using this particular cross-validation technique, we overcome two problems. First of all, we iterate many times to get a reasonable number of different splits and then average the performance from them. This is crucial in terms of the relatively small size of the dataset. Secondly, we ensure that we will not experience any splits that contain only non-POIs for training. If we have 15% of POIs in a dataset, we will also have 15% of them in the training set and 15% in the testing set. The use of Stratified Shuffle Split is more correct in comparison to other splitting techniques, as many of them will not preserve the class frequency while splitting the data.

2. Gaussian NB

Gaussian Naïve Bayes is a very simple classifier that is almost impossible to tune because of no parameters that can be set (except for prior probabilities). The algorithm has been checked with the helper function, for both sets of chosen features, and the results are as follows:
For the set of features select with *SelectKBest*:

Accuracy: 0.85414 Precision: 0.48171 Recall: 0.27650 F1: 0.35133 F2: 0.30225
Total predictions: 14000 True positives: 553 False positives: 595 False negatives: 1447 True negatives: 11405

For the set of features select with *feature_importances_*:

Accuracy: 0.84736 Precision: 0.44381 Recall: 0.27050 F1: 0.33613 F2: 0.29342
Total predictions: 14000 True positives: 541 False positives: 678 False negatives: 1459 True negatives: 11322

We can see that both feature selections gives us almost acceptable values for both precision and recall (precision being over 0.44 and recall getting over 0.27). However, a second algorithm should be looked at to have a comparison to Gaussian Naïve Bayes.

3. Decision trees

Decision trees give much more freedom in terms of tuning the algorithm. Tuning the algorithm can help us achieve better performance/evaluation metrics. In some cases it also helps us speed up the calculation process (for example in support vector machines).

Two parameters have been chosen for tuning: *max_depth* and *min_samples_leaf*. In order to best tune it, a GridSearchCV was used. In order to retrieve the most useful parameter values, the dataset had to be split into train and test parts. Each data split could give different parameter values – as a result, the function using GridSearchCV was run four hundred times (each consecutive time with an initial random state different from *42* in the initial code). In this way, the most frequent parameter values were chosen, as documented below:

For the set of features select with *SelectKBest*:

{'max_depth': {2: 99, 3: 91, 4: 103, 5: 104, 6: 103}, 'min_samples_leaf': {1: 213, 2: 91, 3: 108, 4: 88}}

For the set of features select with *feature_importances_*:

{'max_depth': {2: 125, 3: 134, 4: 108, 5: 72, 6: 61}, 'min_samples_leaf': {1: 203, 2: 104, 3: 102, 4: 91}}

We can see that maximal depth for features selected with *SelectKBest* is not conclusive, as four values were quite frequent. The highest F1 score was found for *max_depth* = 3:

Accuracy: 0.84657 Precision: 0.40842 Recall: 0.16500 F1: 0.23504 F2: 0.18733
Total predictions: 14000 True positives: 330 False positives: 478 False negatives: 1670 True negatives: 11522

It was far off from the desirable result of both precision and recall larger than .3. However, with the second set of features, I found that the results are much better. Two most frequently occuring values of *max_depth* were checked, and the result for *max_depth* = 3 was finally satisfying:

Accuracy: 0.84714 Precision: **0.45501** Recall: **0.35400** F1: 0.39820 F2: 0.37045
Total predictions: 14000 True positives: 708 False positives: 848 False negatives: 1292 True negatives: 11152

In the context of the project, in around 45.5% of times, people who were classified as Persons of Interest were in fact labelled correctly – this piece of information is given to us by precision. In other words, out of 1556 predictions showing that the person is a POI, the algorithm made the right choice in 708 of them.

Additionally, 35.4% of all instances where a person was an actual POI were labelled correctly. This piece of information is given to us by recall. In other words, out of 2000 instances where the person was in fact a POI, the algorithm showed 708 of them were POIs.

Finally, accuracy shows us that in almost 85% of cases, a correct label was given (POI or non-POI).