Machine Learning Final Project – Enron Data

Mario Bonilla
Udacity Data Analysis Nanodegree
P5 – Intro to Machine Learning

Contents

Introduction

Enron Corporation (former New York Stock Exchange ticker symbol ENE) was an American energy, commodities, and services company based in Houston, Texas.

Before its bankruptcy on December 2, 2001, Enron employed approximately 20,000 staff and was one of the world's major electricity, natural gas, communications, and pulp and paper companies, with claimed revenues of nearly $111 billion during 2000. Fortune magazine named Enron "America's Most Innovative Company" for six consecutive years.

At the end of 2001, it was revealed that its reported financial condition was sustained by an institutionalized, systematic, and creatively planned accounting fraud, known since as the Enron scandal. Enron has since become a well-known example of willful corporate fraud and corruption. The scandal also brought into question the accounting practices and activities of many corporations in the United States.

The **aim of this project is to build a prediction model to identify POI** based on financial and email data publicly available after the Enron scandal. POI is a person of interest, usually top or middle management; someone who was either indicted for fraud, settled the issue with the government, or voluntarily testified in exchange for immunity.

For the project, a database containing data about 146 executives who worked at Enron was used. Mostly, **Scikit Learn** methods were used (Decision Tree, PCA, apart from numerous other tested) as well as the Numpy library (Python)

We will have first a glimpse at some aspects of the database; then we will process the features, perform an algorithm and check the results.

Dataset overview

There are 146 executives in the Enron Dataset (number of data points).

Their names are:

```
['METTS MARK', 'BAXTER JOHN C', 'ELLIOTT STEVEN', 'CORDES WILLIAM R', 'H
ANNON KEVIN P', 'MORDAUNT KRISTINA M', 'MEYER ROCKFORD G', 'MCMAHON JEFF
REY', 'HORTON STANLEY C', 'PIPER GREGORY F', 'HUMPHREY GENE E', 'UMANOFF
 ADAM S', 'BLACHMAN JEREMY M', 'SUNDE MARTIN', 'GIBBS DANA R', 'LOWRY CH
ARLES P', 'COLWELL WESLEY', 'MULLER MARK S', 'JACKSON CHARLENE R', 'WEST
FAHL RICHARD K', 'WALTERS GARETH W',…and more…up to 146.
```

From which 18 are POIs (as define before):

Their names are:

```
['CAUSEY RICHARD A', 'LAY KENNETH L', 'FASTOW ANDREW S', 'YEAGER F SCOTT
', 'BOWEN JR RAYMOND M', 'HANNON KEVIN P', 'RICE KENNETH D', 'CALGER CHR
ISTOPHER F', 'KOPPER MICHAEL J', 'COLWELL WESLEY', 'DELAINEY DAVID W', '
HIRKO JOSEPH', 'BELDEN TIMOTHY N', 'SHELBY REX', 'KOENIG MARK E', 'RIEKE
R PAULA H', 'GLISAN JR BEN F', 'SKILLING JEFFREY K']
```

Also, there are 21 features per person, with financial as well as email related information.

To show this, we have chosen the `Boss` data, Kenneth Lay:

```
{'salary': 1072321, 'to_messages': 4273, 'deferral_payments': 202911, 't
otal_payments': 103559793, 'exercised_stock_options': 34348384, 'bonus':
 7000000, 'restricted_stock': 14761694, 'shared_receipt_with_poi': 2411,
 'restricted_stock_deferred': 'NaN', 'total_stock_value': 49110078, 'exp
enses': 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}
```

For instance, we can see that LAY KENNETH L took the highest payment, USD $ 103,559,793.

Some other details:

For nearly every person in the dataset, not every feature has a value:

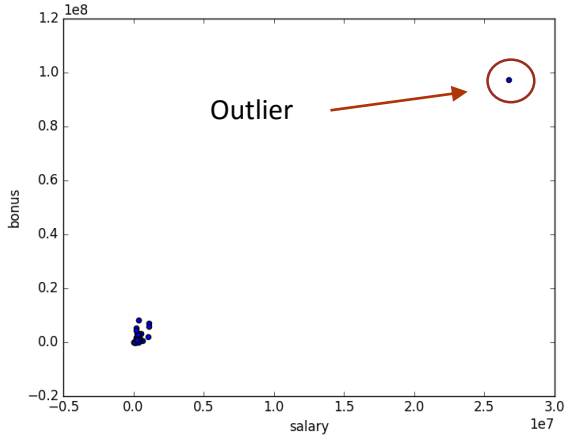
- 95 people have a quantified salary
- 111 people have an email address
- 21 people have NaN total payments, that is the 14.38 % of the total number of people (146). Those are all non-POIs, because 18 people are POIs and none of them have NaN total payments.

Cleaning the Data

Preliminary Plots, Outliers Removal

Because we have already detected **NaN** values and the can interfere with the operations to be performed, and as a general good policy to start, we will replace all the NaN values with zeroes.

We will check the data with one of the simplest set of features, the salary and bonus figures:

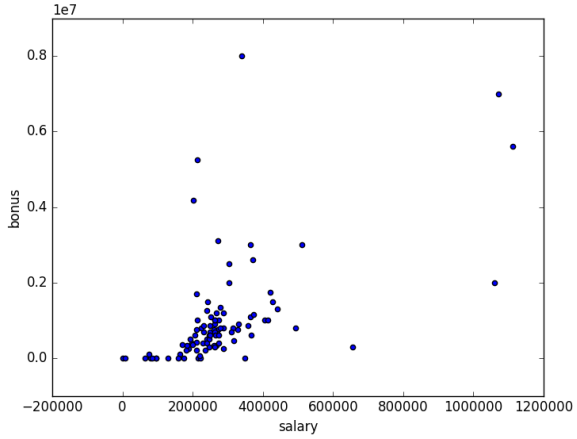


We visually detect an outlier in the upper right part of the plot. There must be someone with a salary higher than 2.5. Who?

```
The biggest outlier is: TOTAL
```

But that is not a person, but a "**total**" number (financial data statistics) so we remove it.

We plot again to see if there are more outliers using the same features:



In the plot we see several more outliers: who are they?

For example, who made at least $5 mil bonus and over $1 mil salary?

```
The new outlier: LAY KENNETH L
The new outlier: SKILLING JEFFREY K
```

But they happen to be POI persons and they contain valuable information so we leave them in the dataset.

Visually checking the dataset, we also found noticed 2 more entries that we will remove:

- THE TRAVEL AGENCY IN THE PARK: that is not a person. Although it could have played a role by for example funneling money, we are not sure about it and just to be on the safe side we remove it.
- LOCKHART EUGENE E: the data point for this person contains no data.

We ended up with a dataset containing 143 data points.

Features

We have to find the best features for our model, the ones that provide the most information. For that objective, we have manually selected a preliminary total of 12 features based on experience. The first one is “poi”.

features_list = ["poi", "bonus", "salary", "shared_receipt_with_poi", "exercised_stock_options", "long_term_incentive", "other", "restricted_stock", "restricted_stock_deferred", "salary", “total_payments", "total_stock_value"]

Building New Features

After cleaning the outliers, we have to find the best features for our model, the ones that provide the most information.

In order to achieve that, we will create 2 new features, being a result of features already present in the dataset.

- The first one is the ratio of emails from poi to a given person by the total number. "**ratio_from_poi_email**"
- The second one is the ratio of emails to poi to from given person by the total number. "**ratio_to_poi_email**"

We create them and include them in the dataset.

So, the total list of features now is:

features_list = ["poi", "bonus", "salary", "ratio_from_poi_email", "ratio_to_poi_email", "shared_receipt_with_poi", "exercised_stock_options", "long_term_incentive", "other", "restricted_stock", "restricted_stock_deferred", "salary", “total_payments", "total_stock_value"]

Feature Classification, Scaling and Classifying

Once we have this manually chosen list of features, we will apply a process of features selection. We have to bear in mind that the dataset is very small.

- First, we **divide** (split) the data into training and testing datasets, using the sklearn "train_test_split" method.

- Then, we **preprocess** the features using a sklearn "StandardScaler" method. Standardizing data is recommended because:
  a) the data set contains figures very different in size (from millions to figures less than one)
  b) we used Principal Components Analysis (PCA) dimension reduction and it performs better after normalizing

- Finally, we apply the sklearn "**DecisionTreeClassifier**" method in order to obtain a list of features (ranking) with the correspondent weights.

The results of the first iteration are:

```
The accuracy with Decision Tree is: 0.805555555556
precision = 0.285714285714
recall = 0.181818181818

Feature Ranking:
1 feature bonus (0.645414509087)
2 feature salary (0.118861607143)
3 feature ratio_from_poi_email (0.118861607143)
4 feature ratio_to_poi_email (0.0670251247852)
5 feature shared_receipt_with_poi (0.0498371518419)
6 feature exercised_stock_options (0.0)
7 feature long_term_incentive (0.0)
8 feature other (0.0)
9 feature restricted_stock (0.0)
10 feature restricted_stock_deferred (0.0)
11 feature total_payments (0.0)
12 feature total_stock_value (0.0)
```

The accuracy is fine, however the precision and recall are very low.

Algorithm Selection and Fine Tuning – Validation and Performance

In the code file attached to this document, one can also find (# commented) other methods used during the process of finding the best results.

We tried the MinMaxScaler and the StandardScaler, for normalizing/**scaling**. Standardizing data is recommended because:

a) the data set contains figures very different in size (from millions to figures less than one)
b) we used Principal Components Analysis (PCA) dimension reduction and it performs better after normalizing

We tried the **classification** methods DecisionTreeClassifier, Gaussian Naive Bayes, Support Vector Machines (SVM), KNeighborsClassifier, RandomForestClassifier and AdaBoostClassifier.

Also, we tried several **cross validation** methods like StratifiedShuffleSplit, ShuffleSplit, StratifiedKFold, as well as Train Split. Cross-validation is the process of splitting the data into training and testing data. In this way, the model will train on the training part of the data, and it will be validated on the testing part of the data.

A lot of people seem to consider cross validation and validation to be synonymous. But they are not. Cross validation can be used when tuning the parameters of the model. Cross validation is simply a way of generating training and validation sets for the process of parameter tuning.

Model validation is one form of testing. Validation happens during the model construction phase. Validation is done to evaluate the appropriateness of a learned model, which is a measurement of the quality of parameters used during model training. Validation happens prior to real deployment. Once the model is trained and deployed, it is possible to monitor its performance offline on a test dataset. This kind of offline batch testing does not require retraining the model. It is also possible to use one dataset for validation and another for offline testing, as long as the test set does not contain the validation set.

We formed **pipelines** with (PCA + DecisionTreeClassifier), (PCA + GaussianNB), (PCA + RandomForestClassifier) and (PCA + SVC).

We also manually tried a myriad of **different parameters** of the methods above mentioned with the goal of obtaining better results. Each algorithm has its own set of parameters. By modifying those better results can be achieved; however, there is always the risk of tuning up the model to work well with the working data and then not work that well when faced to a bigger set of new data.

That introduces us to **tuning,** which is the work perform on machine learning algorithms variable data based on some parameters which have been identified to affect system performance as evaluated by some appropriate metric (in this particular case, accuracy, precision and recall). Improved performance reveals which parameter settings are more favorable (tuned) or less favorable (un-tuned).

One of the difficulties is that learning algorithms require parameters to be set before the models can be used. Even if used "as is" (with no parameters) they take the default ones in the background. How those parameters are set depend on a whole host of factors. The goal is usually to set those parameters to optimal values that enable to complete a learning task in the best way possible.

Therefore, tuning an algorithm or machine learning technique, can be simply thought of as process which one goes through in which the parameters that impact the model are optimized in order to enable the algorithm to perform in the best way possible, for our objectives.

Translating this into common sense, tuning is essentially selecting the best parameters for an algorithm to optimize its performance given a working environment such as a specific dataset, etc. And tuning in machine learning is an automated process for doing this.

Finally, the StandardScaler plus the train_test_split along with DecisionTreeClassifier and/or the (PCA + DecisionTreeClassifier) were the two "finalist". We chose those ones because they are easy to understand, implement and return good results.

Some advantages of decision trees are:

- Simple to understand and to interpret.
- Requires little data preparation. Other techniques often require data normalization (we did that for the PCA process and because the data was heterogeneous in size).

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting.
- Decision tree learners create biased trees if some classes dominate over others.

To improve the results, we chose 6 features (trial and error fine tuning) and we also change some parameter in the sklearn "DecisionTreeClassifier" (again, trial and error fine tuning).

- features_list = ["poi", "bonus", "salary", "ratio_from_poi_email", "ratio_to_poi_email", "shared_receipt_with_poi", "total_stock_value"]
- clf = tree. DecisionTreeClassifier(min_samples_split=10)

The results are now much better:

```
Decision Tree Classifier after modification:
The accuracy is: 0.848484848485
The Precision is = 0.333333333333
The Recall is 0.666666666667
```

That gives, when run locally against "tester.py" the following results:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=No
ne,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
       Accuracy: 0.82429     Precision: 0.36857    Recall: 0.32250
       F1: 0.34400    F2: 0.33077
       Total predictions: 14000     True positives:  645  False positive
s: 1105 False negatives: 1355 True negatives: 10895
```

We would like to mention that we tried another method, a pipeline with PCA and DecisionTreeClassifier:

```
Pipeline with PCA and DecisionTreeClassifier:
The accuracy is: 0.757575757576
The Precision is = 0.222222222222
The Recall is 0.666666666667
```

And in this case, although the accuracy is not too bad and the recall is fine, the precision is somehow low.

But, when run locally against "tester.py" the results are surprisingly good, pretty close to the first method:

```
Pipeline(steps=[('PCA', PCA(copy=True, components=6, whiten=False)), ('c
lassification', DecisionTreeClassifier(class_weight=None, criterion='ent
ropy', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=1, min_weight_fraction_leaf=0.0,
            presort=False, random_state=0, splitter='best'))])
       Accuracy: 0.81893     Precision: 0.35809    Recall: 0.33750
       F1: 0.34749    F2: 0.34143
       Total predictions: 14000     True positives:  675  False positive
s: 1210 False negatives: 1325 True negatives: 10790
```

Anyway we choose the "direct" DecisionTreeClassifier method because the results are slightly better (although very similar, especially the F1).

Conclusions

The algorithm used in this project throws a good accuracy when tested with the whole dataset (tester.py). The figure is 0.81 and that means that about 80 % of the POIs were corrected labelled as POI (number of items in a class labelled correctly divided by all items in that class).

The recall is the probability of predicting a POI being actually a POI. The figure is not that great; about 1/3 of the time will predict it correctly. Mathematically, the true positives divided by the sum of (true positives + false negatives).

The precision is the probability of having predicted someone is a POI, what is the chance of actually being a POI. In this case, about 2/3 of the times will be false alarms. Mathematically, the true positives divided by the sum of (true + false) positives.

Finally, the F1 score is a weighted average of recall and precision.

It is possible to improve this figures:

- having more data to train (and test).
- better tuning the parameters at every step:
    - better feature selection mix,
    - as well as adding new features (ratios…etc.)
    - perform machine learning in the email text, searching for insights, patterns…
    - modifying method´s parameters until the optimal mix.

References

- Scikit- Learn, used extensively, by the hundreds: http://scikit-learn.org/stable/user_guide.html
- Stack Overflow, used extensively: http://www.stackoverflow.com
- GitHub, used extensively: http://www.github.com
- Udacity Forum, used extensively: https://discussions.udacity.com/c/nd002-p5-intro-to-machine-learning
- Natural Language Toolkit: http://www.nltk.org/
- Stemmers: http://www.nltk.org/howto/stem.html
- Wikipedia (Enron, several machine learning articles): http://www.wikipedia.org