Enron - Machine Learning Project

Introduction

In this project I am going to use **Machine Learning** to try to identify Persons of Interest by analysing financial and email data from some of the Enron employees. The idea is to get a list of people that could be also POI and then maybe involved in Enron fraud case.

There were many employees in the company, so getting a list with enough accuracy could help as a starting point for a more detailed investigation of those people that the process identify as possible POI.

In this particular case I have lot of data, from financial data to thousand of emails sent and received by Enron employees. Reading, investigating and identifying each element one by one would take too much time and be an error prone process. Machine Learning can help by finding patterns in the data that could be used to identify POIs. Since there is no 1-1 relationship between a feature and the possibility of being a POI, I would need to use different algorithms and processes to get that list by training.

Understanding the Dataset and Question

Cleaning invalid data

I used two different methods to try to find data that is not valid. First one, we find the person with the max total_payments. It was **TOTAL,** that is just a sum of all values.

Also checked for top 5 people with NaN values, and one of them was **THE TRAVEL AGENCY IN THE PARK,** also not a person so removed too.

Data exploration

The original datasets includes 146 people but as explained before two were removed. All the info shown from now on only takes in account the **valid 143 people**. This is done to avoid filling the document with noise (all the statistics and information before and after the removal).

For each person we have this list of features:

- **salary**
- to_messages
- **deferral_payments**
- **total_payments**
- **exercised_stock_options**
- **bonus**
- **restricted_stock**
- shared_receipt_with_poi
- **restricted_stock_deferred**
- **total_stock_value**
- **expenses**
- **loan_advances**
- from_messages
- **other**
- from_this_person_to_poi
- poi
- **director_fees**
- **deferred_income**
- **long_term_incentive**
- email_address
- from_poi_to_this_person

Being those in **bold** financial features and all others email related features.

poi is an exception, since it just identifies if the person is POI or not. **From all those 145 people, just 18 are classified as POI.** That is 12%, very low number. I will have to pay special attention when training and testing our Machine Learning algorithm because we don't have many POI examples to use.

Not all features have data for all employees. There are cases where for some people the value of a feature could "**NaN**", that means not a number or in other words "**we don't have data**".

Number of NaN for each feature:

- salary: 50
- to_messages: 58
- **deferral_payments: 106**
- total_payments: 21
- loan_advances: 141
- bonus: 63
- email_address: 33
- **restricted_stock_deferred: 127**
- total_stock_value: 19
- shared_receipt_with_poi: 58
- long_term_incentive: 79
- exercised_stock_options: 43
- from_messages: 58
- other: 53
- from_poi_to_this_person: 58
- from_this_person_to_poi: 58
- poi: 0
- deferred_income: 96
- expenses: 50
- restricted_stock: 35
- **director_fees: 128**

Being the ones in **bold** the top 3 features without information for most of the employees. All are financial data.

Those with many NaN values seems to be related with employees on top of the company (stocks, director fees...). The intuition says that those on top of the company could be potential POI. Let's see from those few people with no NaN in those features, how many are POI. The idea is to try to find a correlation manually:

How many employees that has director_fees are POI?
There are 0 POI with director_fees.

How many employees that has restricted_stock_deferred are POI?
There are 0 POI with restricted_stock_deferred.

How many employees that has deferral_payments are POI?
There are 5 POI with deferral_payments.

The data analysis shows that my intuition was not correct, none of the employees with director_fees and restricted_stock_deferred are initially labeled as POI. In the case of deferral_payments, from the 39 employees with information 12,8% are POIs. In summary, seems there is no clear correlation.

This also leads to a important questions:

What features are all "NaN" for POI people?

- restricted_stock_deferred
- director_fees

So, from all POI there are two features that include no information at all for them, probably making these features useless in the learning phase. As I explained before, the number of POIs is pretty low, so I need to maximise the POI data we pass to the algorithm. I will investigate this further in next sections.

What features have information from all POI people?

- total_payments
- total_stock_value
- expenses
- other
- poi
- email_address

I can ignore **poi** and **email_address**, but other three could be interesting to take in account since they have data for **all** our POI. Taking in account that there is not much data to analyse from POI, I need to investigate with more details those that give us more information about them.

From the data I have now, seems that **total_payments**, **total_stock_value** and **expenses** could be good features to investigate. For these reasons:

- All POI have data from those features.
- The number of NaN is pretty low across all the people. The worst one is expenses, where 34% of the times include no info, but at same time we have 100% from POIs. I will add also **salary** and **bonus** that is usually one of the most important factors to find who has important roles in a company.

Outliers

Let's analyse features in more detail. I are going to check the difference between the average and the max value:

Statistics for salary:

MAX percentage over AVG: 391%

Statistics for total_payments:

MAX percentage over AVG: **3948**%

Statistics for total_stock_value:

MAX percentage over AVG: **1465**%

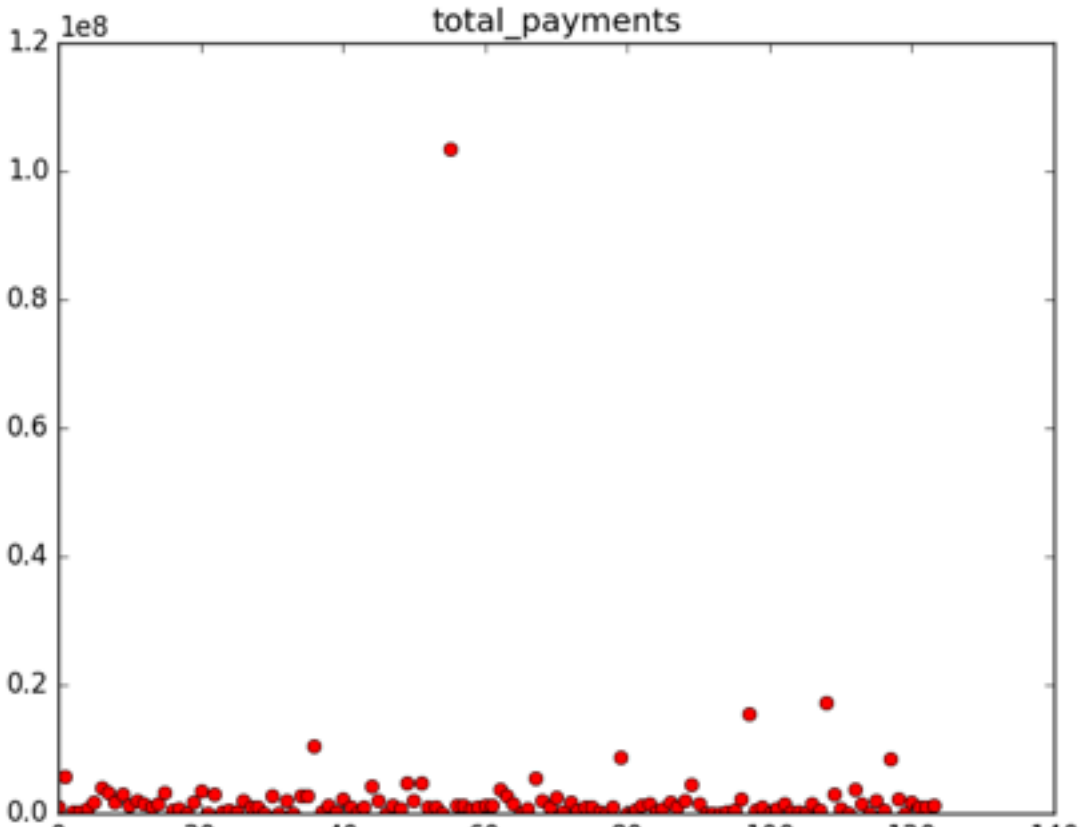
Statistics for expenses:

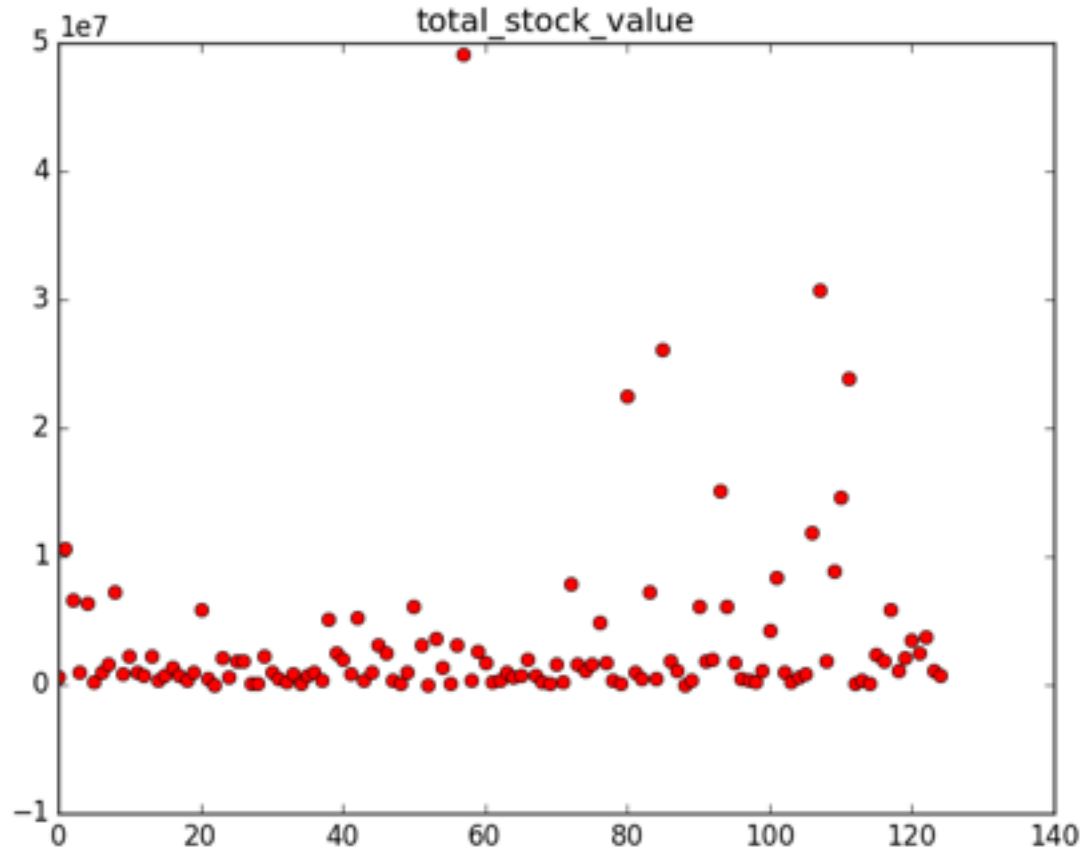
MAX percentage over AVG: 422%

Statistics for bonus:

MAX percentage over AVG: 666%

There are clear outliers in **total_payments** and **total_stock_values** where the difference between the max and the min is more than 1000%. Let's take a look:





So, there is at least one clear outlier in each feature. **Who are those outliers?**:

Person with max total payment is: **LAY KENNETH L**
Person with max total_stock_value is: **LAY KENNETH L**

LAY KENNETH L was the CEO of the company and one of the highest paid CEO in America, so not only an outlier in Enron. Sentencing was going to happen on September 11th 2006, but he died on July 5th 2006. He is one of the POI. Taking in account he was the CEO and one of the small group of 18 POI I am not going to remove it because it is valid and important data point.

Optimize Feature Selection/Engineering

Create new features

From email related features there are two that would make sense to merge in order create a new one. **from_this_person_to_poi** and **from_poi_to_this_person**. A person that receives lots of emails from POI could be as important as a person that send lots emails to POI. Those are two very related features that by merging them we balance their importance. So, in case there are Person A and Person B:

Person A:
from_this_person_to_poi: 1
from_poi_to_this_person: 10

Person B:
from_this_person_to_poi: 10
from_poi_to_this_person: 1

email_exchange_with_poi will be the result of from_this_person_to_poi + from_poi_to_this_person for each person. This new feature will balance the importance of the two previous ones, so Person A and Person B have the same changes of being POI.

As I explained in previous section, from all financial features there were two (restricted_stock_deferred and director_fees) that had no info at all from our POI. Taking in account that the number of POI we have is pretty small, we need to maximise the data we learn from them, so I am going to ignore those two features.

Also in previous section I found some features that provided information for all POI. Following the same idea of maximising the data we get from POI, **total_payments**, **total_stock_value** and **expense** will be included**.** Since **salary** and **bonus** is something that affect all the employees in a company, those two will be included two.

Finally, my own **email_exchange_with_poi will be included.** Will also add "**to_messages**". It is a feature we haven't manually checked in detail but our Lasso Regression should test it.

Intelligently select features

In total, the number of features is 7. In order to find which ones describe better who is a POI I am going to use Lasso Regression to identify those features that could be helpful. This method will help to reduce the number of features making the Machine Learning process faster and with less overfitting. This is the result:

Feature	Lasso Coefficient
total_payments	-5.72309789e-09
total_stock_value	1.69715208e-08
expenses	**8.00841247e-07**
salary	1.29194211e-07
bonus	**7.09070870e-08**
to_messages	-1.59799164e-05
email_exchange_with_poi	**5.74164316e-05**

Looks like **expenses**, **bonus** and **email_exchange_with_poi** are the ones that could help to find our POI, so those are the three features that will be used.

Properly scale feature

This is the information stored on those features:

Statistics for expenses:
MAX: 228763
MIN: 148
AVG: 54192
MAX percentage over AVG: 422%

Statistics for bonus:
MAX: 8000000
MIN: 70000
AVG: 1201773
MAX percentage over AVG: 666%

Statistics for from_poi_to_this_person:

MAX: 528
MIN: 0
AVG: 65
MAX percentage over AVG: 814%

The difference between two financial features are pretty significant. The max value goes from 228K to 8M and the avg. also shows great difference. But in this particular case both are financial data (dollars) that represents the income for each person on each category. Therefore, if we scale it we could lose some information. If someone receives 8M dollar bonus, it is something that looks pretty important and that could lead to find a POI. Therefore, **the logic says that features should be used as they are right now without any scaling.**

The final project shown here doesn't include any scaling, but I actually tested everything with **scale** and **MinMaxScaler**. SVC should be the one to show important improvements, but after testing it, there is no real difference between using scaling and not using it. Same happens with Decision Trees. Final scores remain mostly the same.

Pick and Tune an Algorithm

Parameter tuning

Each algorithm has different parameters that can change the way they work. That means that the result can be changed and improved using parameters that suit better the data that is being investigated. If parameters are not modified the algorithm would run with default values that could or couldn't be good enough for our dataset. So, parameter tuning is important to get the best possible result from our machine learning algorithm.

In this particular project we are going to try **Support Vector Machines** and **Decision Trees**. SVM is very versatile and has plenty of parameters to choose from being **Gamma** and **C** the most important ones as stated in the documentation.

http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#example-svm-plot-rbf-parameters-py

"Proper choice of C and gamma is critical to the SVM's performance."

In the case of Decision Trees I will tune **min_samples_split** and **min_samples_leaf** to decide in which particular point the split is done in the three.

In next section I present a list of parameters to test and which combination provides the best result. I have automated the test using **GridSearchCV.**

Support Vector Machines

The parameters to test are:

{'C': [1, 10, 100, 1000, 10000], 'gamma': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}

Being the best result:

{'C': 1, 'gamma': 0.0001}

Running our classifier again with those parameters we get the exact same score of 0.8.

Decision Trees

Parameters to test are:

{'min_samples_split' : [1, 2, 3, 4, 5, 10, 15, 20, 25, 50, 100], 'min_samples_leaf' : [1, 2, 3, 4, 5, 10, 15, 20, 25, 50, 100]}

Being the best result:

{'min_samples_split': 100, 'min_samples_leaf': 1}

Ironically, the score I receive is lower, 0.8 and also lower precision and recall (more about this in next section).

So, looks like **Decision Trees** with default values provide the best score.

Validate and Evaluate

Evaluation Metrics

In previous section we saw that the algorithm with best score was Decision Trees using default parameters. In order to evaluate the performance of the algorithm I am going to use evaluation metrics: **precision** and **recall**.

There are different reasons to use evaluation metrics instead of the score provided by the classifier. The most important one is that the classifier score (or accuracy) is not ideal for skewed classes like the one we are investigating now, where the number of POI is pretty small compared with all the people we have in the dataset.

Second reason is that we really want to try to identify possible new POIs, even if later checks confirm that they were not really involved. So, false positives in this particular project are not a problem and that is something we can handle. These scores takes false positives and false negatives into account. Metrics will be:

Precision: True Positive / (True Positive + False Positive)
Recall: True Positive / (True Positive + False Negative)

Both are presented as ratios from 0 to 1 being 0 the worse value and 1 the best.

For precision, a ratio of 1 means that every person labeled as POI is indeed POI.
For recall, a ratio of 1 means that every known POI person was labeled as as POI.

During our firsts test I find that our Decision Tree with default parameters get these scores:

Precision: 0.666666666667
Recall: 0.285714285714

So, 66% of the time the algorithm labeled someone as POI it was correct. And only 28% of known POI were identified as POI.

Recall is pretty close to what we expect in our final result (0.3) but still not there. I need to check the validation strategy and see how we split the data between training and test.

Validation

Validation is really important in Machine Learning, because we need to train our classifier but also test it to know if it is working as expected. It is important to find, based on the data we have and the labels, how much data use for testing and how much for training. The idea is to find the exact cut point that maximises the performance. It is important to not lie to ourself. A pretty common error is to use same data for training and testing, that will show pretty good (and difficult to believe) results.

This is already coded:

```
from sklearn.cross_validation import train_test_split
features_train, features_test, labels_train, labels_test = \
```

```
    train_test_split(features, labels, test_size=0.3, random_state=42)
```

It is using train_test_split, using 30% of our features/label combination for tests. A simple test, increasing the test_size to 0.4 shows that we get worse results:

Precision: 0.4
Recall: 0.25

We don't have many POI data to train our classifier, so reducing the training points by adding them to the test data it is not going to help. We reduce the test_size to 0.2. Things looks much better:

Precision: 0.5
Recall: 0.6

In order to improve it even more, I have added **stratify=labels** to the list of parameters. So data will split in a stratified fashion using the labels. After doing this, the scores we get are even better:

Precision: 0.75
Recall: 0.75

Reducing it to 0.1 really doesn't improve it that much, so 0.2 looks like the best split point based in the metrics received.

Algorithm Performance

Using tester.py we get the following results:

```
DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
     Accuracy: 0.79367    Precision: 0.37681    Recall: 0.36400  F1:
0.37030    F2: 0.36649
     Total predictions: 12000   True positives:  728  False positives:
1204  False negatives: 1272 True negatives: 8796
```