Udacity Course: Data Science II

Luan Fernandes
São José dos Campos, São Paulo, Brazil

Enron Scandal

Final Project Report

May 14th 2019

Introduction

In this project, we are going to use the power of machine learning to try to predict who was involved in the famous Enron' Scandal. This scandal refers to a series of unethical practices, mostly from managers, including changes in the balance sheett of the company to change its performance. These practices were later discovered and led the company to its bankruptcy, and many senior managers were found guilty in some degree.

Our goal here is to try to predict who was involved in this schema looking at individual attributes that may reveal their roles in the scandal. For instance, for a matter of fact, people involved in these bad practices earned a lot more money than regular employees. Then, we can use a machine learning classification model in order to understand this implied relationship.

Data Exploration

The dataset contains 146 data points, from only 18 people have "poi" allocation (the classed are highly non balanced). No missing values in the entries, but outliers were investigated in the following way:.

1. Salary: there is a value of 26.704.229,0 USD that seems an outlier, standing very far from the others in boxplot view. Investigating the person associated with this data point, we noticed that it says 'TOTAL', which points out to be the sum of all salaries in the Enron dataset, which we confirmed in the "enron61702insiderpay.pdf" document. After that removal, we can still see maybe 4 outliers in boxplot view, but investigation showed that

these points are real persons, like the “POI” LAY KENNETH L (Enron’s CEO from 1995 to 2001) and SKILLING JEFFREY K (Enron’s after Kenneth), so we definitely should not remove them from the dataset.

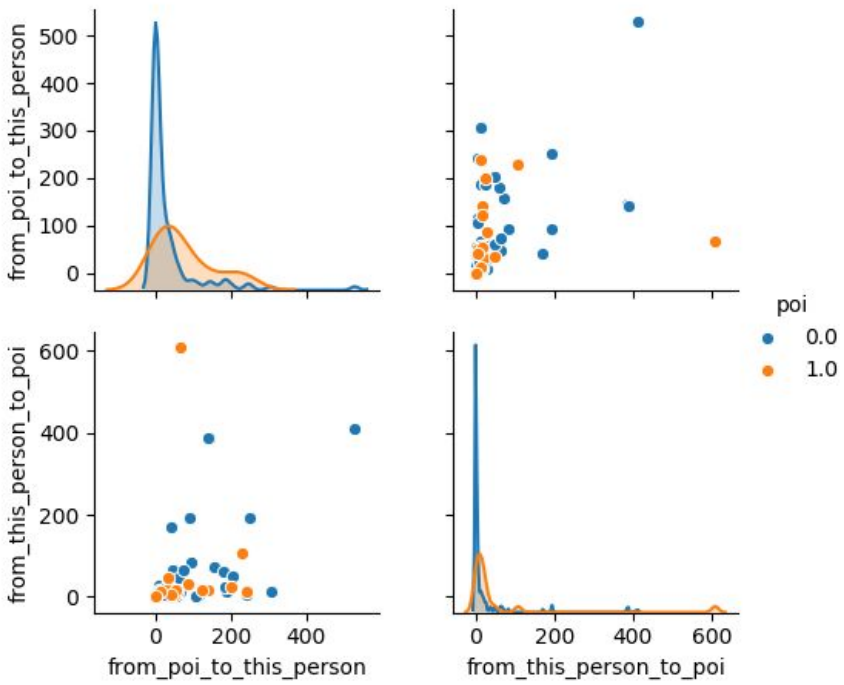
2. total_payments: looks like it shows an outlier too, but investigation also shows that this point is Kenneth Lay too. While all other data points had total_payments below 20 millions USD, with most of these below 1 million USD, Lay had over 100 million USD in total payments. Same happens for bonus feature: highest values correspond to real cases.

Features Selection

Between features that show emails information, it’s clear that 'from_poi_to_this_person' and 'from_this_person_to_poi' are the principal ones, because they show the relationship between a person in the dataset and a POI. In the image below (already without the outlier), we see that a POI does not used to email other POI as much as the non POI did, which raises a hypothesis that POI could use other medias to chat information, specially about the scandal (this hypothesis is not explored here). The “shared_receipt_with_poi” also can show a relationship with poi, so we kept these 3 email features.

About the financial features, it not difficult to see, intuitively, that “salary”, “total_payments”, “bonus” are the most likely related with a chance to be a poi (poi earned much more money due to the scandal). After some investigation, we kept only some features: 'salary', 'total_payments', 'bonus', 'total_stock_value', 'exercised_stock_options', 'long_term_incentive' and 'director_fees'. At first, we used more features in the pipeline for the best model, but later we removed less important features whenever they did not diminished out scoring.

.



New Features

We also created a new feature to test in our model, which is the ratio between “total_payments” and “salary”, in order to try to detect big differences between the current salary and all the money paid to someone. However, this new feature had no impact in the best classifier found, probably because it is a combination that can be detected by the original classifier, when looking to both features separately.

Pipeline Structure

In order to find the best model, we chose to build a pipeline using the previous features selected (we’ve fit the pipeline for more financial features, but then we removed some that did not lower our scores). This pipeline is based on the following steps:

- Scalar;
- PCA;
- Classifier.

For the scalar, we chose StandardScaler(), a very common one, but for PCA and Classifier we've tried several options. The classifiers analyzed were: Gaussian Naive Bayes, Decision Trees, Random Forests and Support Vector Machines.

For the pipeline models, we also used GridSearchCV for cross-validation, for which we could pass a variety of parameters to tune our classifiers. As we are going to see, the best classifier found was the **Naive Bayes**. The following topics resume the best parameters found for each classifier under this pipeline, as well as the main scores for these best parameters (all of them referencing the same features chosen) when we used tester.py to test the classifier:

1. SVM
    best params:
        PCA_components: 4
        SVM_C: 0.001
        SVM_gamma: 0,1
    tester.py results
        accuracy: 0.86870
        precision: 0.11012
        recall: 0.00671
        F1: 0.01265
        F2: 0.00826

2. NB
    best params:
        PCA_components: 4
        NB__priors: None
    tester.py results
        accuracy: 0.85648
        precision: 0.40277
        recall: 0.30045
        F1: 0.34417
        F2: 0.31654

3. Decision Tree (DT)
    best params:
        PCA_components: 2
        DT_criterion: 'entropy', DT_splitter: 'random'
    tester.py results
        accuracy: 0.80955
        precision: 0.24802
        recall: 0.25567
        F1: 0.25179
        F2: 0.02541

4. Random Forest Classifier (RFC)
    best params:
        PCA_components: 2
        RFC_criterion: 'entropy'
    tester.py results
        accuracy: 0.83195
        precision: 0.02531
        recall: 0.17461
        F1: 0.20665
        F2: 0.18616

We see that for the NB classifier, we could only set options for “priors” parameter, but for the other ones we could vary some parameters, such as “criterion” for Decision Trees and Random Forest, or “C” and “gamma” for the SVM. The tuning of classifiers is important in order to avoid overfitting or underfitting, so that step is very relevant.

Validation

In the section before, the validation that generated the score metrics was done using the tester.py script, even though a prior validation is done in poi_id.py. Both validations have the main ideia: it is fundamental to separate the training set from the testing set. If one does not do that, he incurs in a big risk of biasing the model a lot, in a sense that it will have a great performance for the dataset studied but will not be able to predict well new data. The separation step avoid this, forcing the data to really learn patterns of training set that also apply to the test set, and then the model will predict better new data.

Particularly, the tester.py uses StratifiedShuffleSplit() to separate 30% of the data to testing, and for training set it creates 1000 folds of same size were the percentages of each class are maintained on each fold.

Evaluation Metrics

Finally, as seen in the Pipeline Structure section, we obtained a performance of 0.40 for precision, 0.30 for recall and 0.86 for accuracy. While the last shows a got fit for the whole model, we did get recall and precision over 0.30 (and smaller values for other classifiers). The precision show us the ability of the model not to label as positive a sample that is negative, and the recall shows the ability.

In the context of Enron's emails, we rather prefer a good recall over precision, because we can guarantee that we are not labeling a true poi as negative. A false poi, labeled by the clf as poi, can be taken out from investigation after later parts in the process, but we do not want to "lose" a true poi. to find all positives samples, or not labeling as negative a sample that is positive. Then, we got the classifier with the best accuracy and recall.