**POI IDENTIFICATION FROM ENRON DATASET USING ML**

**SOHAN SAMANTA**

**INTRODUCTION**

Enron corporation was an American Energy company based in Houston, Texas. It was formed in 1985 by Kenneth Lay after merging with Houston Natural gas and InterNorth. By the use of accounting loop holes and poor financial data, some members of the company hid millions of dollars in debt from failed deals and projects. The scandal eventually led to the bankruptcy of Enron Corporation. Many executives at Enron were indicted with a variety of charges and some were even later imprisoned.

In this project, we will try to play detective and build a program that will be able to identify persons of interest from email and financial data made public by the authorities. By person of interest we refer to any employee or separate individual who was either involved directly or indirectly in the Enron fraud or had some knowledge about it but chose to hide it and were later indicted for it by the government.

This project flow can be divided into 4 major parts:

1. Data Set
2. Feature Selection
3. Picking and tuning an algorithm
4. Validation and Evaluation

**THE ENRON DATA SET**

The data set is comprised of two major subdivisions of information – financial information and email data.

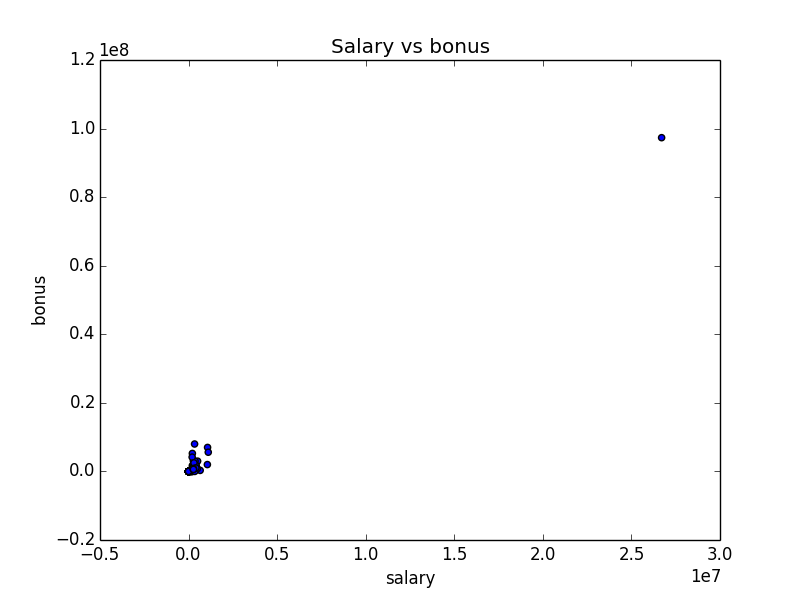
The data is stored in a pickle file in a dictionary of dictionary format. The first thing I am interested in after downloading is the structure and length of the data. Since each item corresponds to one person who could be a potential POI (person of interest), the length essentially gives us a number of potential suspects.

An example of the way the data is stored is provided below:

{'METTS MARK': {'salary': 365788, 'to\_messages': 807, 'deferral\_payments': 'NaN', …….},

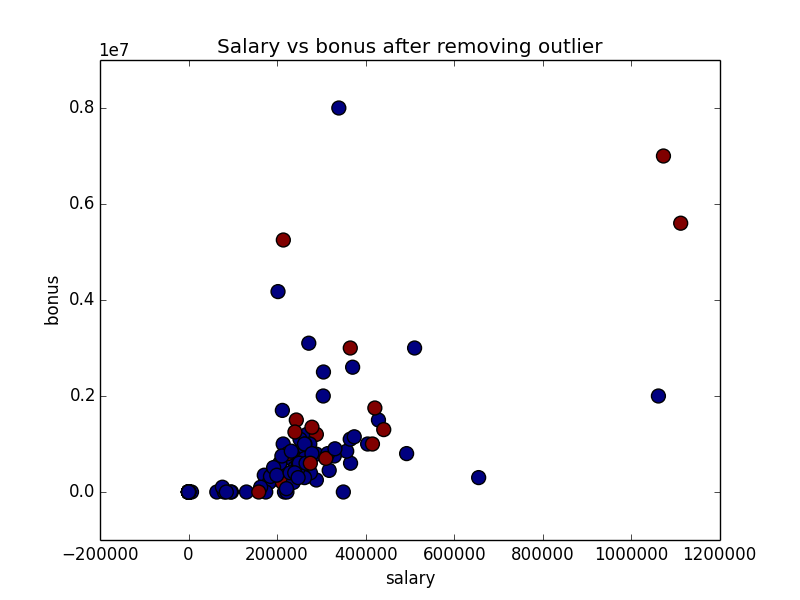
'BAXTER JOHN C': {'salary': 267102, 'to\_messages': ………….}, , , }

The length of the dictionary is 146. Hence we have 146 people to consider. Whenever dealing with a large set of data, one should always be wary of outliers. The simplest way to check for outliers in this case would be to plot a simple bonus vs salary scatter plot. Any outlier with abnormal value would be at once visible.



So, we do find one outlier. A little search inside the data dictionary reveals that this outlier is the ‘TOTAL’ value at the end of the list. We go through the list of names once manually, and find one wrong entry which does not belong to any person. It is named ‘THE TRAVEL AGENCY IN THE PARK’. We also check if there are any entries with all values missing. And sure enough, we find an entry of a person called ‘LOCKHART EUGENE E’.

So we promptly remove all outliers and re-plot the data.



Now, the data looks more realistic, with a few large values. I have also colored the persons of interest with red. This gives us an idea of how the POIs and non POIs are distributed.

We also take this time to look at how many poi and non-poi person are present in the list. We find the following:

1. Number of poi = 18
2. Number of non-poi = 125

**FEATURE SELECTION**

Feature selection is as important as the selection of the algorithm that these features will be used within. While good features can improve the training and prediction of an algorithm, not all algorithms will give the same results with the same set of features.

But before we start working with features, let us look at some of the important properties of the features:

1. There are a total of 21 features.
2. Some features have a high number of missing or 0 values. The top three features with a very high number of null values are:
   * Loan Advances (140 null values)
   * Director Fees (127 null values)
   * Restricted Stock Deferred (126 null values)

In this step, what we essentially endeavor to do is lay a ground work for our features and algorithms. We start by selecting a wide range of features, which we intuitively feel may be of some importance and then use a set of algorithms to test them on. This step is essentially a bit of trial and error, as we don’t know what parameters would give us good results, nor do we have any idea as to which algorithm will be best suited for this data set.

While we use most of the features from the original data set, we also create a few features on our own like the ratio\_of\_to\_poi\_messages and salary square to include some added non linearity within the feature set. We end up having 28 distinct features, as listed below.

List of all features:

'poi', 'salary', 'deferral\_payments', 'total\_payments', 'loan\_advances', 'bonus', 'restricted\_stock\_deferred', 'deferred\_income', 'total\_stock\_value', 'expenses', 'exercised\_stock\_options', 'other', 'long\_term\_incentive', 'restricted\_stock', 'director\_fees', 'to\_messages', 'from\_poi\_to\_this\_person', 'from\_messages', 'from\_this\_person\_to\_poi', 'shared\_receipt\_with\_poi', 'ratio\_of\_to\_poi\_messages', 'ratio\_of\_from\_poi\_messages', 'total\_poi\_messages', 'salary\_square', 'bonus\_square', 'total\_stock\_value\_square', 'incentive\_cube', 'bonus\_salary\_ratio'

Now, for any algorithm, 28 features are an overkill. Thus, we use the Principal Component Analysis to bring the number of features down. The help of the pipeline feature of Python is taken to add the PCA feature to each and every classifier. Since we are also using Grid Search in tandem with PCA and pipeline, we initially set a range of parameter values for the classifiers and the PCA to test each of their performances and then choose the best ones based upon their performances.

The performance can be summarized by the performance metrics we are most interested in, and is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SL No | Algorithm | Accuracy | Precision | Recall |
| 1 | Decision Tree | 81.06 | 29.73 | 30.8 |
| 2 | Linear SVC | 79.41 | 26.71 | 31.2 |
| 3 | Ada Boost | 83.78 | 27.5 | 13.25 |
| 4 | Random Forest | 85.78 | 41.97 | 17.25 |
| 5 | Logistic Regression | 82.66 | 33.33 | 30 |

For this project, we set out with a target of around 30% precision and recall and a high value of accuracy. From the above table we find that both Decision Tree and Logistic Regression provide pretty good results.

The question is can we do better? Also, we have used PCA to combine our features. Should we also include a few original features to our list? Would that be better?

**PICKING AND TUNING AN ALGORTIHM**

We have seen that Decision Tree and Logistic Regression seem to give better results than the other classifiers we have selected. So, we will remove the others and go forward with only these two, to try and improve our results.

Before we start tuning, I am forced to reflect upon my initial decision of having a big list of features. The way these classifiers work, is that they are trained by a fraction of our data set, and then tested on the remaining data. One issue that I can think of, can arise, from having a large set of features is a case of over fitting. Over fitting would essentially decrease the generalization ability of our algorithm, and thus when we test the algorithm using Stratified Shuffle Split, it would fail to give proper results.

This can actually be observed by using only a single data set, without shuffle split, and using it to predict the results. The results are significantly higher than the one shown in our table above using shuffle split. (In my program I have used 2 evaluation functions to test this. One uses the train test data to evaluate, the other uses a stratified shuffle split)

In light of the above, I reduce my features list to 15 features, which are as follows:

'poi', 'salary', 'bonus', 'total\_stock\_value', 'long\_term\_incentive', 'from\_poi\_to\_this\_person',  
'from\_this\_person\_to\_poi', 'shared\_receipt\_with\_poi', 'ratio\_of\_to\_poi\_messages', 'ratio\_of\_from\_poi\_messages',  
'salary\_square', 'bonus\_square', 'total\_stock\_value\_square', 'incentive\_cube', 'bonus\_salary\_ratio'

This gives certain interesting results, and they are summarized as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SL No | Algorithm | Accuracy | Precision | Recall |
| 1 | Decision Tree | 80.99 | 33.5 | 33.5 |
| 2 | Logistic Regression | 81.71 | 33.27 | 27.85 |

The performance of Decision Tree has certainly improved, while that of Logistic Regression has decreased. But we are not much concerned about the decrease in the performance of Logistic Regression, because the increase in the performance of DT is much better than all the performances till now.

Going forward, let us just concentrate on DT and its features.

An important function in python that we can use for looking at the relative importance between features is the **“feature\_importances\_”** function. We use this to decrease our features list further and settle on the following list:

'poi', 'salary', 'bonus', 'total\_stock\_value', 'from\_poi\_to\_this\_person',  
'from\_this\_person\_to\_poi', 'shared\_receipt\_with\_poi', 'ratio\_of\_to\_poi\_messages',  
'salary\_square', 'incentive\_cube'

This list improves the performance of our Decision Tree even further:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SL No | Algorithm | Accuracy | Precision | Recall |
| 1 | Decision Tree | 81.64 | 35.62 | 35.48 |

We do not use PCA any more as the number of features are now under a manageable limit, and using PCA at this point would only decrease our performance further.

**NOTE:** Manually playing with the features can probably give better results in any one of the performance parameters at the cost of another. For example, setting the feature list to the following:

'poi', 'ratio\_of\_to\_poi\_messages', 'ratio\_of\_from\_poi\_messages','bonus\_salary\_ratio', 'salary'

provides a precision and recall of 38%, but decreases the accuracy to 77%. However, for an overall balance I am content with the feature list and performance results we got in the previous step.

**VALIDATION AND EVALUATION**

Based on the above analysis, we finally settle on a Decision Tree with the following parameters:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

presort=False, random\_state=47, splitter='best')

The above parameters were decided upon after running a list of parameters through Grid Search in the initial stages of this project.

It gives us a satisfactory performance with 81% accuracy, 35% precision and 35% recall.

**CONCLUSION**

In this project we have given an equal weightage to the precision and recall values and not just based our decision on Accuracy. The reason behind this is that accuracy only gives us an idea as to how many predictions are correct. But recall and precision gives us an even deeper understanding based on the correct predictions of not only the true positives but also the false negatives as percentages of false positives and true negatives.

A high value of accuracy, combined with decent values of precision and recall makes our algorithm truly robust.