Udacity Data Analyst Nanodegree
Machine Learning Project
Identifying Enron's POIs' (Person of Interest)

by Mustafa Bilal Tahir

Introduction

The Enron scandal shocked the world in 2001 and has gone down in history as one of the most glaring examples of corporate greed and lax regulatory oversight. It has led to new laws coming into play and a new way of thinking about assessing a company's financial health. The funny thing about Enron was that they were never technically hiding anything. Most of the evidence collected against them came from documents *they* disclosed. The problem was that they buried all the critical information beneath layer after layer of complex regulations and accounting garble. Malcolm Gladwell captures this duplicity really well in one of his prices titled "Open Secrets - The mystery of Enron"[1]. Consequently, one of the few good things to come of this tragic affair where thousands of employees lost their life's savings was that the Enron scandal was heavily studied and it's data very well documented by the investigators who looked into it. Our objective is to use Machine Leaning to see if, looking at the dataset, we could have predicted if an employee in the database was a Person of Interest (POI). A POI is defined as an employee who as indicted for fraud in the forthcoming investigation or was at least called on to testify.

The Dataset

The Enron dataset has the following fields for each employee:

{'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':}

The 'poi' field is the indicator which identifies the people who were POIs. There were 146 data points or employees in total in the dataset. Of these, only 18 were POIs. Looking at these numbers off the bat we can tell that our Machine Learning algorithm has a strong chance of being biased because the majority of the people in the data are not POIs.

Other differentiation points that can lead to potential issues include, for example, the values for the total payments field. The percentage of the population that has an "NaN" value (No value) for their total payments is 14.4%. However, if you just look at the POIs there is no one who has this value. This could be due to the way the data points were added (Non POIs could be added from a spreadsheet while the POIs adde din manually and with more care to have all of their information). We should be careful in how our Machine Learning Algorithm interprets this. For example if we later on added more people to the dataset by hand and also added in their total

---

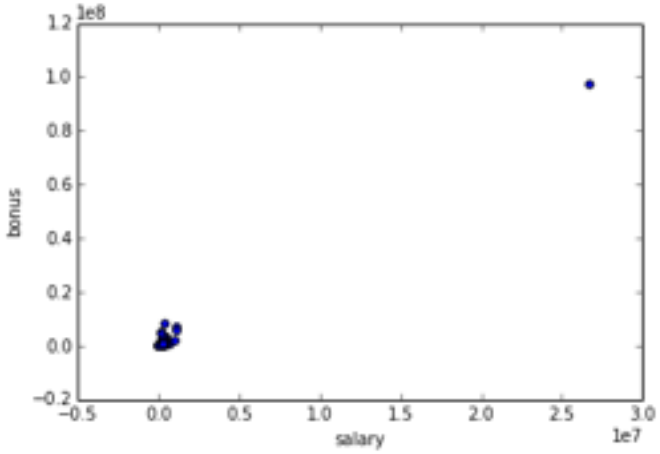
[1] http://www.newyorker.com/magazine/2007/01/08/open-secrets-3

payments information, our algorithm could falsely predict someone is a POI based on this information even though it is unlikely they are considering how small that population is compared to the whole.

Outliers

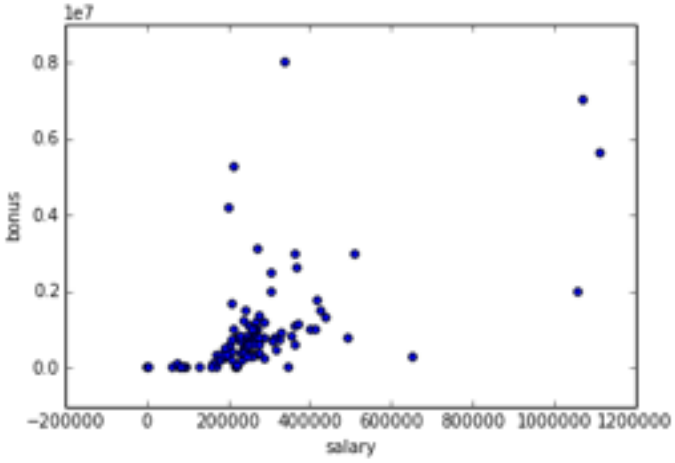
Plotting the data using the Salary and bonus fields gives us a good idea of the spread and allows us to identify an outlier immediately. considering just how far away it is, we were able to conclude that it was actually the 'Total" field mistakenly left in. We removed this data point but kept other data points which appeared to be outliers because of their high bonus and salary (Kenneth Lay and Jeff Skilling) which were legitimate data points.

Data Plot with Outlier



Data Plot without Outlier

Using a



As we can see the data looks much better in the second plot once the outlier is removed.

Features Selection

The features were all ranked for importance using a Decision Tree Classifier to fit the model (the selection of which is explained below). Here is the list:

('from_messages', 0.3715676466061259)
('other', 0.23924335886780096)
('total_stock_value', 0.18186401613367895)
('director_fees', 0.096994141937962117)
('poi', 0.063046192259675379)
('shared_receipt_with_poi', 0.047284644194756538)
('salary', 0.0)
('to_messages', 0.0)
('deferral_payments', 0.0)
('total_payments', 0.0)
('exercised_stock_options', 0.0)
('bonus', 0.0)
('restricted_stock', 0.0)
('restricted_stock_deferred', 0.0)
('expenses', 0.0)
('loan_advances', 0.0)
('from_this_person_to_poi', 0.0)
('deferred_income', 0.0)
('long_term_incentive', 0.0)
('from_poi_to_this_person', 0.0)

Using this, we can eliminate the all the features which have a 0 value. The poi is the label so we have to keep that. We also replace 'shared_receipt_with_poi' with 'expenses' based on trial and error as the fit is much better without the inclusion of this feature.

Two new features were created and tested as well:

1. The fraction of all emails to a person that were sent from a person of interest
2. The fraction of all emails that a person sent that were addressed to persons of interest

The logic for using these metrics was that employees with a larger proportion of their email correspondence to and from with a POI would themselves likely be a POI as well. Reading in these new features and re-testing, our list of features that had a non-zero importance becomes:

('from_poi_to_this_person_frac', 0.22412918115876462)
('from_this_person_to_poi_frac', 0.22412918115876462)
('total_stock_value', 0.2030169268933314)
('restricted_stock_deferred', 0.15131086142322089)
('from_messages', 0.12207361729122496)
('salary', 0.10807918673087208)
('shared_receipt_with_poi', 0.063046192259675379)
('bonus', 0.047284644194756538)
('other', 0.047284644194756538)
('total_payments', 0.033774745853397542)

The two new features show up in our list of most important features so it does appear that these features add value to the algorithm.

We further refined our features by taking out the least important features such as ‘total_payments’ and ‘other’ etc., and re-running and manually optimizing the recall and precision metrics (see definition below). Our final features list was:

features_list = ['poi', 'total_stock_value', 'from_poi_to_this_person_frac',
'from_this_person_to_poi_frac', 'bonus', 'salary','shared_receipt_with_poi',
'restricted_stock_deferred']

We did not do any feature scaling as it was not necessary with our algorithm (see below).

Model Selection

For our model performance, we are mainly going to focus on 3 metrics: accuracy, precision and recall. Accuracy is the overall fit of the model i.e. how much of the data can the model accurately predict. Precision and recall are also common evaluation metrics used to judge the performance of machine learning algorithms. Precision is simply the probability that a label that is identified as say, a ‘positive’, is actually a positive; while recall is the probability that a positive label will be identified by our model.

We compared 3 models initially without any parameter tuning: Gaussian Naive Bayes, Decision Tree, Adaboost (SVM was not included because it was taking too long to train the data). Here are the corresponding values of the 3 classifiers at their default settings:

Classifier	Accuracy	Precision	Recall
GaussianNB	0.26671	0.16305	1.0000
AdaBoost	0.79521	0.20847	0.15500
Decision Tree	0.82143	0.37876	0.39050

The Decision Tree and Adaboost handily won out against the Naive Bayes model based on the fit, and the Decision Tree classifier had the highest precision and recall and was therefore ultimately selected.

The precision and recall were further improved after tuning the parameters[2]. Each algorithm has some default settings that can be changed to improve performance. The parameters and default values of a Decision Tree Classifier are:

```
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,
        random_state=None, splitter='best')
```

Parameter tuning is important as it can result in the algorithm giving a much higher performance (see below for examples on how the metrics can change based on different settings).

GridSearchCV was used to tune the parameters of the classifier but the recommended settings seemed to optimize precision over recall and so ultimately the parameters chosen were further refined using trial and error. The best parameter settings recommended just by GridSearchCV were:

Parameters: {'min_samples_split': 10, 'max_leaf_nodes': 5, 'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 5}

However, even though the precision was approximately 0.4, the resulting recall was below 0.3. After playing around with the parameters starting from this, it was determined that it was sufficient to simply change the default criterion to 'entropy' from the default setting of 'gini' to see a marked improvement in the metrics.

We also need to make sure we cross validate our results to make sure they are reasonable. Cross validation is basically testing our model to make sure it would generalize well to additional data. We can do this by slicing our data in different training/testing splits to see if on average we get a similar level of performance. This eliminate any particular bias a subset of our dat might be having on our algorithm.

The results for our model were validated using the Stratified ShuffleSplit cross validation iterator from sklearn. Using just fold can be a basic mistake as the data is sliced in a linear way (for example the first half is cut for training and the second half is used for testing). The problem with this is that we could have a lot of our data points that are pointing to one prediction in the first half of our data which would bias our algorithm into predicting a larger number of those points for the second half which may not be the case. Fortunately, the cross validator Stratified ShuffleSplit which we used overcomes this because it is a mix of Kfold cross validation with stratified random folds so the data is not just being cut in one fashion but is being shuffled randomly to give different training/testing splits.

---

[2] While only the default settings for the models were compared against each other in the final code to select the model, an attempt was made at an intermediate stage of the process to tune Adaboost as well. However, the metrics did not improve enough to challenge the Decision Classifier performance.

Our final average metrics were:

Accuracy: 0.82893

Precision: 0.39633

Recall: 0.37750

Conclusion

Although the dataset was challenging (a small group of POIs in a large population of Non-POIs), we were able to get some decent performance out of our algorithm. We could potentially make further improvements by looking at the data in a more detailed fashion and trying to come up with more features.