Enron Fraud Detection Project

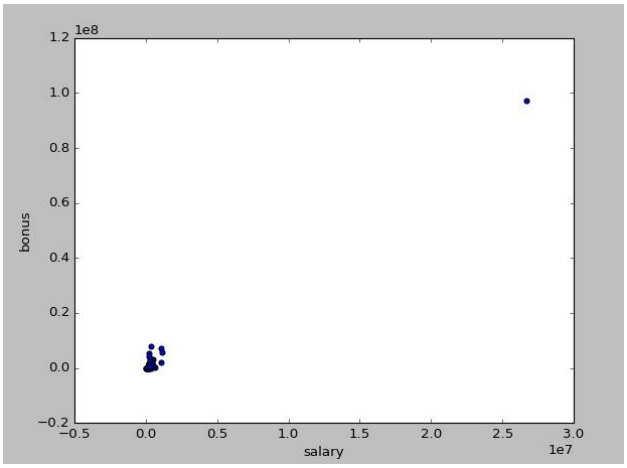
1. Project Goal and use of Machine Learning

Enron was one of the largest corporations in the US in 2000. By 2002, it was bankrupted due to a huge corporate fraud. After Federal investigation related to this fraud, a significant amount of data entered into the public record. Now we have access to emails and financial data for company executives. Goal of this project is to find a systematic way for finding person of interest(POI) in our dataset. POI means one of this kind of persons in this dataset: indicted, reached a settlement, plea deal with the government, testified in exchange for prosecution immunity.

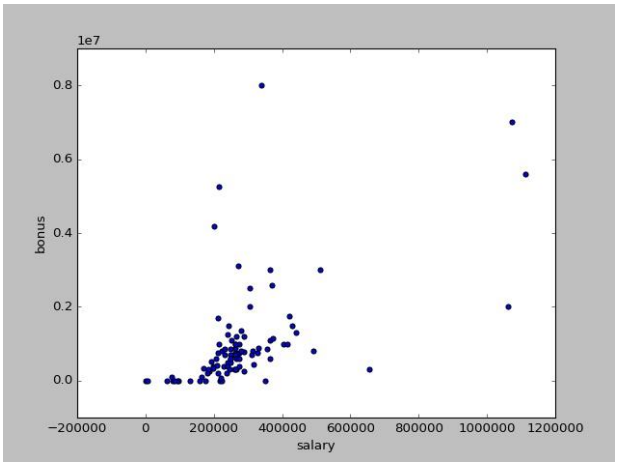
We manually put labels for all data points as: poi and non-poi.

So, we have some data point with some related features and we want to put each data point in one possible group. This is a perfect match for Machine learning supervised algorithms. We can train a classifier for distinguishing an input base on attributes(features) it has and returning one of our output labels for it. For this project, I am using 'scikit learn' library that contain many good methods for machine learning projects.

Our dataset contains 146 data points and 18 individuals are 'persons on interest' base on above definition. First step is visualizing data points by some features like 'salary' and 'bonus'.



Picture1: Visualizing data points with two features to find outliers



Picture2: Removing the outlier

Base on Picture1, there is one clear outlier. The related salary for this point is more than 25,000,000. I wrote some codes for finding name of this person. I found out it's not a person and it is the TOTAL row in our data set that mistakenly slipped between our data points. I delete it from data set and look at data again. (Picture2); In this plot, there are some more outliers. After implementing the classifier, I examine a potential outlier removal. I delete these big players (i.e. LAY KENNETH L, SKILLING JEFFREY K) because I think maybe it helps the classifier to work better for more ordinary POIs, but the result was on the contrary. So, I keep them for final data set.

It seems like there are more outliers, but as we know the story of Enron, we are looking for these outliers and we shouldn't remove them. They could be some of POIs we want to find.

Another issue was that there are some data points that has very few values for features and they h ave many NaNs. With a code snippet, I found out that “LOCKHART EUGENE E” has 20 NaNs out of 21 features, and “THE TRAVEL AGENCY IN THE PARK” is a company and not a perso n with 18 NaNs. Interestingly eliminating these two data points, decrease all performance metric at least 3%. So, I decide to keep them in the dataset. I guess these NaNs has a positive effect to c ategorize a point as non-POI. Because POIs by far has fewer NaNs.

2. Feature selection

First, I used all 19 numeric features as starting point for all classifiers I implemented. Moreover, because the range of these features quantities are very different I used a scaler to transform them between (0,1). After trying different features, finally I choose to use most of financial features and some more important email related features and I got better results for precision and recall. As we know this fraud is related to getting more money and I guess the most important features should be related to the extra money somebody can earn. Features like ‘bonus’ and ‘exercised stock options’. So, I created two new features that reflects square of these two features so they will show these two features stronger than others. First I added these two new features besides all other financial features and then I replace the squared new features with the original ‘bonus’ and ‘exercised stock options’ and I get better results. Replacing ‘bonus’ with ‘bonus_square’ improve accuracy 1%, Precision 3% and F1_Score 1%.

Similarly, I create a new feature that is the square of the salary, but this feature is not good as the two other new features. So, I didn’t use the square of salary as a feature in my final selection.

On the other hand, there many NaN in our data set. If a feature has many NaNs, it shouldn’t be very useful for our algorithm. Base on a snippet code I wrote for calculating the percentage of NaNs I found out there are 3 features that has more than 87% NaNs. I omit them one by one and look at performance metrics like F1.

Feature	Percentage of NaNs	Accuracy after removing it	F1 score after removing it
loan_advances	97%	85%	43%
director_fees	88%	85%	41%
restricted_stock_deffered	87%	86%	45%

So, the first and last feature should be eliminated together but not the second one.

Base on trial and error, I remove a feature at a time and look at the effect this removal has on performance metrics. Because of limitation of volume of this report, I only mention some results of this process.

Feature	Accuracy after removing it	F1 score after removing it
shared_receipt_with_poi	86%	44%
from_poi_to_this_person	84%	41%
from_this_person_to_poi	84%	38%
'deferred_income'	85%	37%
'total_payments'	85%	43%
'total_stock_value'	85%	39%

Finally, the features list I used is:

features_list = ['shared_receipt_with_poi', 'from_poi_to_this_person', 'from_this_person_to_poi', 'salary', 'deferral_payments', 'total_payments', 'restricted_stock', 'other', 'total_stock_value', 'expenses', 'director_fees', 'deferred_income', 'long_term_incentive', 'bonus_square', 'exercised_stock_square'].

Because there are many features and a few data points, I am thinking about ways to select best features and focus on signals and eliminate noise. I used a PCA for creating principal components, and I feed the results of this part to a SelectKBest to choose a handful of components and use these final transformed features as an input for my classifier. I create a pipeline to try different approaches. The Pipeline I implemented has these parts:

Scaler ➔ PCA ➔ Selector ➔ Classifier

3. Choosing the Algorithm

I tried 3 different algorithms with different pipelines. The algorithms I used are:

Naïve Bayes, SVM and Decision Trees. For this specific project, I realized that SVM doesn't work properly and it's very hard to tune to get reasonable results. Decision Trees are good but they were very prone to overfitting with few amounts of data we have, and it returns poor results. After some trial and errors, I found out the Decision tree's best performance results are not very satisfying. The best results I got was:

Accuracy = 0.84, Precision = 0.33, Recall = 0.33, F1 = 0.33

The best results I got was with 'Gaussian Naïve Bayes' classifier. It works better than other algorithms I tried and I understand I should only focus on tuning the parameters for this classifier.

The best results I got for this classifier is:

Accuracy = 0.86, Precision = 0.47, Recall = 0.42, F1 = 0.45, F2 = 0.43

So, I ended up using the 'Naïve Bayes' as my classifier.

4. Parameters Tuning

Parameters tuning means finding the best parameters for an algorithm for a specific learning problem. Each algorithm has some parameters that has default values. For every problem, we should think and fine tune these parameters to get the best possible performance metrics. As a simple example consider 'min sample split' parameter in decision tree algorithms. The default value for this parameter is 2 and it means the algorithm split data points and create the decision boundary if sample has at least two leaves. This will lead to overfitting and poor performance results for some problems. By increasing this number, we are telling to algorithm, stop splitting when you get 15 leaves and it will prevent splitting when it's not useful. Like this there are other parameters that will effect performance results drastically.

After choosing the algorithm, the most important step is to tune the parameters related to the whole Pipeline. If we miss this step, we did half of the job and found the best algorithm and pipeline for the problem, but there is no reason that the default values that comes with each method in the 'scikit learn' package are best for our specific problem, and there is an essential part for doing trial and error and base on results we get, choose the best parameters that tailored to our own problem. For example, in Decision Tree algorithm there are many parameters to tune and I choose to tune these two: *criterion* and *min sample split*. By tuning these two parameters I got different results and in some point the result doesn't change drastically by tweaking parameters and I understand that's the final capacity of this algorithm and I should try another algorithm.

The next candidate was 'Naïve Bayes' classifier and it has only one attribute and that is the priors. In this project, we don't have any priors therefore I didn't put anything as priors and I focus on other parts of the Pipeline for tuning. For doing this step I used an automated approach and implemented the GridSearchCV. Then I tried different parameters for all parts of the Pipeline and at the end get the best estimator base on our goal. It means that I get the best parameters for getting best results for F1_Score. By using F1_Score as our criteria, we are getting best results for both Precision and Recall at the same time and not only one of them.

5. Validating the results

We have a specific amount of data and we need one batch of data for training the algorithm and another batch for testing the performance of our algorithm. Therefore, we need to split the dataset to two different parts as training part and testing part. On the other hand, we want both batches are as big as possible. The problem is the dataset is limited and whatever we put in our training set we lose it for testing set and vice versa. A practical approach for having maximum data in both batches is using cross validation technique. In this approach, we will use all the data we have in both sets. For example, we will put first 10% of data points in testing set and the other points in training set. Then we will use next 10% of data as testing set and others as training set, and so on. At the end, we can get average on performance metrics on each testing result. But there is a subtle point that we should keep in mind. It is possible that some data points are related to one label and they are settled in first part of data and if our classifier train on them and test on other data points the performance results will be awful. For avoiding this problem, we should shuffle data points first and then split them.

I used the StratifiedShuffleSplit() method from model selection package in 'scikit learn' for validating my model. I used random state = 42, to get approximately same results as the tester function.

6. Evaluation Metrics

For evaluating the results of implemented algorithms, I used accuracy, precision, and recall. Performance results for my 'Gaussian Naïve Bayes' classifier after scaling the features values and using PCA with 10 components and selecting 5 most effective of those components are:

Accuracy: 0.86 Precision: 0.47 Recall: 0.42

The Accuracy metric is 0.86. It means that on average the classifier is labeling 86% of the cases correctly as POI or non-POI.

The Precision metric is 0.47. It means when the classifier labels somebody as POI on average in 47% of the cases it's correct and that person is really a person of interest.

The Recall metric is 0.42. It means that on average the classifier can distinguish 42% of POIs between all data points.