Analysis of the Enron Fraud Scandal and Fraud Detection using Classification Algorithms

Written by Anwesa Basu,[1] Swapnendu Majumdar [2]
Khoury College of Computer Science, Northeastern University[1,2]
440 Huntington Ave 202
Boston, Massachusetts 02115
basu.anw@northeastern.edu, [1] majumdar.s@northeastern.edu [2]

Abstract

In the year 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

We have investigated the Enron corporate scandal by analyzing the Enron Email Dataset and used different classification algorithms to identify the persons of interest (POI), who have been involved in fraudulent activities. The Enron Email Dataset is freely available on Kaggle and contains approximately 500,000 emails generated by employees of the Enron Corporation. It was obtained by the Federal Energy Regulatory Commission during its investigation of Enron's collapse. We will attempt to use different classification algorithms on the email data to help us identify person of interest (POI) regarding fraudulent activity in the organization, which eventually led to its downfall. The sheer scale of perceived fraudulent activity in an organization as big as Enron makes this project fascinating.

Introduction

In the year 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. We have investigated the Enron corporate scandal by analyzing the Enron Email Dataset and used different classification algorithms to identify the persons of interest (POI), who have been involved in fraudulent activities.

To perform our analysis, we implemented untuned classification algorithms and then further tuned some of them to improve their performance.Finally, we documented the results of these algorithms, and performed a comparative analysis of the different algorithms used in the scenario

Background

The objective of this project was to design a classifier, using sci-kit-learn algorithms, and apply concepts of machine learning such as cross-validation, parameter tuning, feature engineering and metrics based evaluation to finalize an algorithm which has a higher success rate of predicting the POI candidates. We had to choose between applying supervised and unsupervised machine learning algorithms to this problem, and went ahead with supervised machine learning algorithms since the data-set was marked, and we had appropriate labels to compare with.

Related work

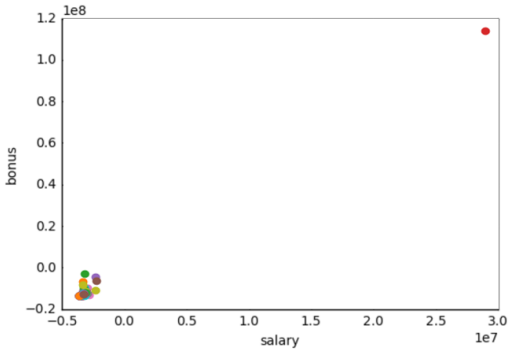
Alongside the algorithms that we decided to use, we also could have used other approaches such as Support Vector Machines (SVM), Long Short-Term Memory (LSTM) and logistic regression. SVMs are a type of supervised machine learning algorithm which can be used for classification problems by plotting in the n-dimensional space, where the value of specified coordinate is used to define the value of each feature and two classes can be differentiated by finding the hyperplane. However, since it takes longer time to train the larger data-set, alongside being difficult to comprehend the final model, variable weights and individual impact we chose not use SVMs. Similarly, while LSTM networks are well-suited to classifying, processing and making predictions, they are primarily useful for time-series data and were therefore ignored for the scope of this project. Finally, we chose to avoid implementing logistic regression in the program, since the type of raw data that we were dealing with in this data-set, is mostly discrete and not continuous.

Approach

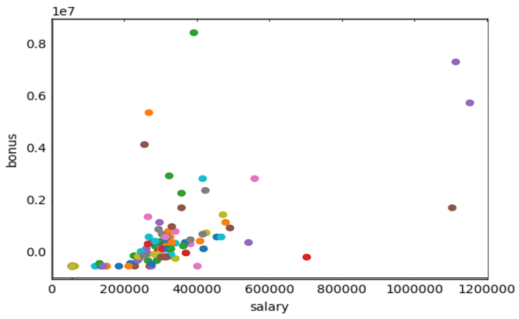
Data Cleaning

Having imported the data-set, we proceeded to clean the data by removing unnecessary and dirty data. We explored the data-set and found that the data-set initially had a total of 146 entries (143 after outlier removal) with details of the employees with their email and financial data. Out of the total 146 records, only 18 records (12%) were labeled as POI in the data-set. There was a total of 21 features (including POI) describing each of the records for respective employees. A lot of the features for employees were labeled as "NaN".

In order to observe the outliers, we decided to plot a few graphs between features such as payments, stocks, salary and bonus to help us visualize better. As seen below, outliers were easily spotted in a few plots -



From the plot above, we can see that outliers are present in the data, and therefore, we will need to clean the data-set before we can proceed. In this figure, a clear outlier is visible, which was later, through code, found out to be a spreadsheet entry of the 'TOTAL' of all employees. This was later removed, after which the same graph looked much better, as can be seen below -



Having removed the outliers, we could clearly visualize the distribution of the points in the plot. Three outliers data points were removed from the data-set. These are -

i) TOTAL: Was an extreme outlier for financial data, and seemed to be a spreadsheet error.

ii) THE TRAVEL AGENCY IN THE PARK: Record did not seem to represent an Enron employee.

iii) LOCKHART EUGENE E: Record contained no useful data, as all values were set to "NaN"

Feature Selection In the dataset, we could initially find the below features -

```
'poi'

'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'

'salary'
```

Other than these, the three new features engineered and used are – 'net_worth', 'proportion_from_poi', and 'proportion_to_poi'.

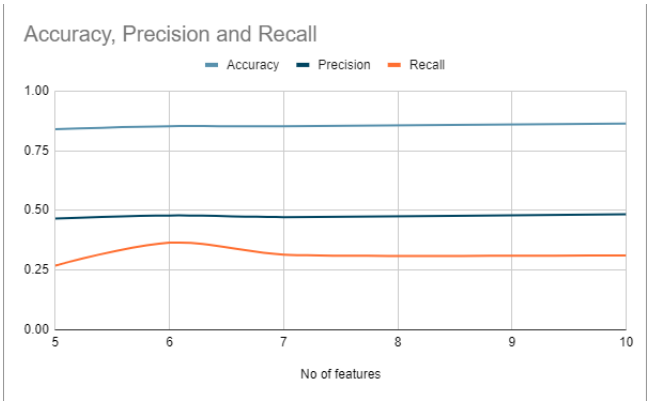
'net_worth' is a feature which is used to encompass the total wealth of a person by combining all financial data available. It was an important feature, as it was a part of our final features list after applying SelectKBest.

'proportion_from_poi' and 'proportion_to_poi' were features engineered to calculate the proportion of messages sent to an average employee by a POI, and the proportion of messages sent by an average employee to a POI.

Out of these, 6 features were chosen as POI Identifiers, by running selectKBest. The selection was based on ANOVA F statistics, and comparing performance of Gaussian Naive Bayes over other number of features.

No of features	Accuracy	Precision	Recall
10	0.86393	0.48407	0.3115
7	0.85179	0.47187	0.3145
6	0.8525	0.47866	0.3645
5	0.84008	0.4658	0.269

The best combination of accuracy, precision and result were obtained when the number of features were at 6, the complete results for which can be seen later in the report.



As we can see, it is better to proceed with 6 features, in keeping with the Bias-Variance trade-off too. According to the Bias – Variance trade-off, too many features increase the variance, and therefore complexity, while too less features leads a model to be biased. –

```
features_list--- ['poi', 'salary', 'bonus', 'defer
red_income', 'total_stock_value', 'exercised_stock
_options', 'net_worth']
feature scores
('exercised_stock_options', 24.815079733218194)
('total_stock_value', 24.18289867856688)
('bonus', 20.792252047181535)
('salary', 18.289684043404513)
('net_worth', 17.80879117424012)
('deferred_income', 11.458476579280369)
('long_term_incentive', 9.922186013189823)
('restricted_stock', 9.2128106219771)
('total_payments', 8.772777730091676)
('shared_receipt_with_poi', 8.589420731682381)
('loan_advances', 7.184055658288725)
('expenses', 6.094173310638945)
('from_poi_to_this_person', 5.243449713374958)
('proportion_from_poi', 5.123946152756893)
('other', 4.187477506995375)
('proportion_to_poi', 4.094653309576953)
('from_this_person_to_poi', 2.382612108227674)
('director_fees', 2.1263278020077054)
('to_messages', 1.6463411294420076)
('deferral_payments', 0.2246112747360099)
('from_messages', 0.16970094762175533)
('restricted_stock_deferred', 0.06549965290994214)
```

The six chosen features were - 'salary', 'bonus', 'deferred_income', 'total_stock_value', 'exercised_stock_options', 'net_worth'

Comparing classification algorithms

We will train different classification algorithms on this data set using the features chosen above, and try to perform a comparative analysis based on the following metrics -

i) Accuracy - Accuracy is the ratio of correctly predicted values to the total predicted values.

$$\text{Accuracy} = \frac{TruePositive+TrueNegative}{TruePositive+FalsePositive+TrueNegative+FalseNegative}$$

.

ii) Precision - Precision describes the ratio of how often our model correctly identifies a positive label,(POI = 1), to the total times it guesses a positive label (all POI = 1 predictions). It can be calculated as -

$$\text{Precision} = \frac{(TruePositives)}{(TruePositives+FalsePositives)}.$$

A high precision value usually signifies that the solution using the algorithm (POIs identified in this case) tend to be correct.

iii) Recall - Recall on the other hand describes the ratio of correct positive labels identified to all the records flagged as POIs. It can be calculated as

$$\text{Recall} = \frac{TruePositives}{TruePositives+FalseNegatives}.$$

A high recall value means, in this case, if there are POIs in the dataset, an algorithm has good chance to identify them.

Untuned Classifiers: Out of the box, we tried a host of classifiers without tuning them. Here are the results which we could see:

Algorithm	Accuracy	Precision	Recall
Gaussian NB	0.8525	0.47866	0.3645
Decision Tree	0.78771	0.25255	0.248
AdaBoost	0.83186	0.37708	0.2715
K Nearest Neighbours	0.85143	0.30198	0.0305
Nearest Centroid	0.82543	0.35714	0.2775
Random Forest	0.85771	0.50499	0.2025

From the above table, it is evident that Random Forest had greater success than all the other algorithms that we tried, owing to better Accuracy, Precision and Recall values.

Tuned Classifiers: We will now try to tune the parameters for two of the above models, in an attempt to improve their performance -

i) Gaussian Naive Bayes
ii) AdaBoost

Parameter tuning is the final step of applied machine learning, which involves finding the appropriate values of the various parameters available for an algorithm to achieve best possible performance, while maintaining a reasonable computational time. Tuning also saves our model from overfitting. Basically, it is just changing some of the variable values to optimize a model. If a model is not tuned, the default values are used, which might reduce performance and accuracy of the model, while increasing the time taken. Tuning the model wrong on the other hand, will possibly break the algorithm, or have it use up a lot more time, leading to wastage of valuable time and resources.

Finally, individual tuning parameters for each dataset are required to obtain the best fit and predictions. So using the default value is just an amateur solution while tuning is necessary, for better performance.
i) I have combined PCA with **GaussianNB** in order to improve the performance of the algorithm. GaussianNB and PCA were combined using Pipeline, while GridSearchCV was also used. GridSearchCV took in a list of [4,5,6] as PCA n_components parameters, and the best results were obtained at n_components = 5. The results can be seen below –

```
GausianNB with PCA fitting time: 1.616s
Pipeline(steps=[('pca', PCA(n_components=5)), ('classifier', GaussianNB())])
        Accuracy: 0.85214       Precision: 0.47796      Recall: 0.37950 F1: 0.42308     F2: 0.39581
        Total predictions: 14000        True positives:  759    False positives:  829   False negatives: 1241   True negativ
es: 11171

GausianNB  evaluation time: 1.47s
```

ii) Similarly, GridSearchCV was used to find the best-suited parameters for **AdaBoost**. The parameter if inputted in GridSearchCV were the base_estimator, which was a decision tree classifier of depth varying from 1 to 6. The results that were obtained are –

```
AdaBoost fitting time: 4.179s
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   n_estimators=45, random_state=40)
        Accuracy: 0.85643       Precision: 0.49650      Recall: 0.35500 F1: 0.41399     F2: 0.37646
        Total predictions: 1400 True positives:   71    False positives:   72   False negatives:  129   True negatives: 1128

AdaBoost evaluation time: 4.563s
```

While precision and recall were above 0.3 for both classifiers, the fitting time and the evaluation time taken by the tuned GaussianNB classifier were vastly superior to those required by the tuned AdaBoost classifier. No feature scaling was required, and hence wasn't deployed.

Validation - Validation is the process of testing our model against new, unknown data, and then evaluating various metrics to gauge its performance. If the same data that has been used for training is used for validation, it will lead to a condition known as Overfitting. As a result of this, the model will perform well and have a perfect score on already seen data, but will have low ability to adapt to previously unseen data and therefore perform poorly on it.

The model in this project is validated using the provided testing script tester.py, where the function test_classifier() makes use of a cross validation method called StratifiedShuffleSplit to split the data into testing and training datasets and returns the indices for testing and training data points separately. The training data in this case was 70% of the whole data.

Conclusion

From this analysis we can conclude that by and large most of the classification algorithms perform equally well on this dataset, with some minor differences in performance metrics. We could also see how tuning the classifiers improved the performance of the algorithms. Among the untuned classifiers, we could see that the Random Forest Classifier had the best metrics, while tuning the AdaBoost Classifier in particular made it perform really well too. The algorithms used in this project can similarly be used to perform other classifications too.

References

i) TowardsDataScience.com
ii) TheCleverProgrammer
iii) Udacity
iv) Tex.stackexchange
v) Mage.ai
vi) Scikit-learn
vii) Kaggle