Identifying Fraud at Enron using Emails and Financial Data

Udacity Data Analysis Nanodegree – Project 4

Introduction

Enron Corporation was one of the world's largest energy companies before it went bankrupt in December 2001. It was revealed that the bankruptcy was caused by 'institutionalized, systematic and creatively planned accounting fraud'. Subsequent legal trials found several members of Enron's board guilty of being involved in the financial fraud.

For this project I have attempted to identify the Persons of Interest (POI) involved in fraudulent activity which contributed to the collapse of Enron. To do this I have used several machine learning techniques to help characterise important features within the financial and email dataset and then used these to help predict which of the employees might be POIs.

Analysis

The dataset was made up of financial and email features as well as a POI indicator. The email data contained both the content of the email messages and also associated metadata (sender and recipient information), the financial data had a range of features including salary, stock options, bonuses, expenses etc. There were 146 entries in the dataset with each entry having 21 features. Of these 146 entries, 18 were classified as a POI and 128 as non-POIs. It is worth noting that this is only a subset of the total Enron employees

Within the dataset there were many features missing values, for example just over 14% had no value for 'total_payments'. IN this project I used the featureFormat function to convert the data dictionary into a numpy array, as it does this it converts any missing values ("NaN") into a 0 by default.

The original dataset contained an entry for the **TOTAL** of all the financial features of each of the Enron employees. This was obvious when plotting the data as it was several magnitudes larger than any other data point (Fig 1), I therefore removed this line from the dataset. Once this line was removed then the same plot (Fig 2) showed only a handful of employees as outliers, however this was expected in a large organisation.

Fig 1.

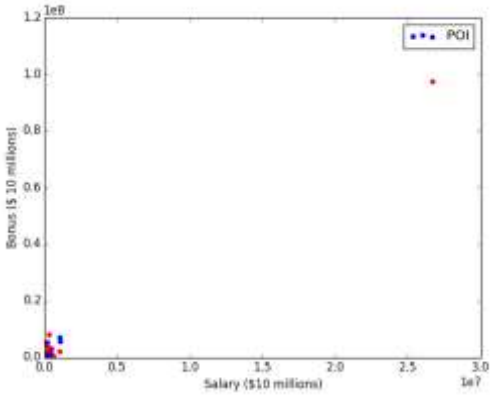
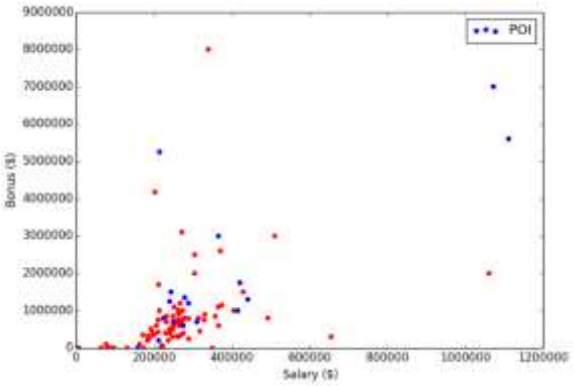


Fig 2.



I also removed the entry for **Eugene E. Lockhart** since this row had no values for any of the features. Finally after looking at the data in more detail I removed an entry for **The Travel Agency in the Park**

as I was only interested in people for my machine learning model. After removing these three entries I was left with 143 records in the dataset.

Feature Selection

In order to optimise and select the most relevant features I used the scikit-learn module **SelectKBest** to identify the most influential features. I chose to focus on the top 10 most influential features as I noticed the importance of the feature begin to reduce quickly after this point.

Feature	Score
exercised_stock_options	*24.815*
total_stock_value	*24.183*
Bonus	*20.792*
Salary	*18.290*
deferred_income	*11.458*
long_term_incentive	*9.922*
restricted_stock	*9.213*
total_payments	*8.773*
shared_receipt_with_POI	*8.589*
loan_advances	7.184

Running a simple DecisionTree Classifier on these top 10 most influential features produced the following scores:

Accuracy: 0.80060, Precision: 0.25531, Recall: 0.25850

When I first ran the **SelectKBest** algorithm the only email feature in the top ten was shared_receipt_with_POI. I was surprised that neither from_this_person_to_POI nor from_POI_to_this_person were included in the most influential features as I would have thought the level of interaction with a POI would have been a strong indicator of also being a POI. Plotting the data (Fig 3) it didn't really show a strong correlation, hence the low score. However when I plotted the fraction of the total emails that went to a POI or came from a POI (Fig 4) I found a much clearer link. This made sense as email volumes can vary quite dramatically and so a percentage of the total is a far better indicator. Once I re-ran the **SelectKBest** the fraction_emails_to_POI became the 5th strongest feature.

Fig 3.

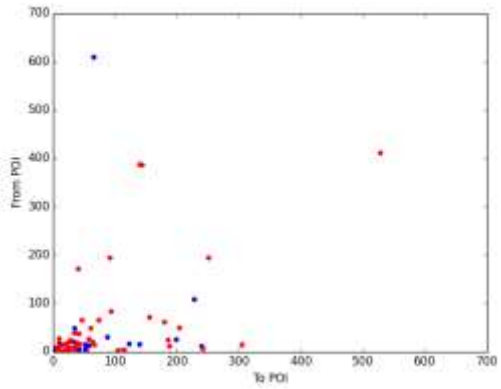
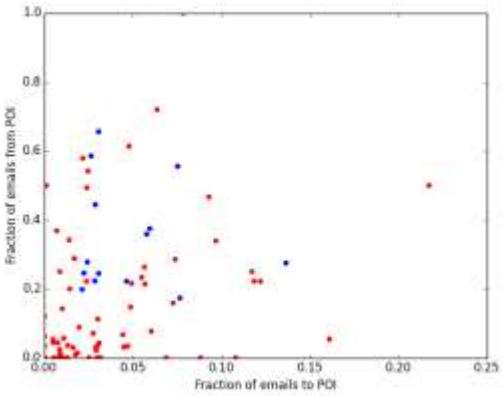


Fig 4.



The new 10 most important features were:

Feature	Score
exercised_stock_options	*24.815*
total_stock_value	*24.183*
Bonus	*20.792*
Salary	*18.290*
fraction_emails_to_POI	*16.410*
deferred_income	*11.458*
long_term_incentive	*9.922*
restricted_stock	*9.213*
total_payments	*8.773*
shared_receipt_with_POI	*8.589*

Running these through the same simple DecisionTree classifier produced improved results with accuracy, precision and recall all increasing.

Accuracy: 0.82340 Precision: 0.33469 Recall: 0.32850

I then tested running the same algorithm using the top 7,8,9,10 and 11 features to test the impact on the results. It was clear that using the top 10 most important features in this dataset maximised accuracy and precision and had the second highest recall score so was a suitable choice:

No. of features	Accuracy	Precision	Recall
7	0.82027	0.32721	0.32950
8	0.82027	0.32764	0.32600
9	0.82140	0.33017	**0.33000**
10	**0.82340**	**0.33469**	0.32850
11	0.81907	0.31951	0.31600

Algorithm

I looked at four different machine learning classifiers; Naïve Bayes, Decision Tree, K Means Clustering and Support Vector Machines (SVM).

Prior to training the K Means Clustering, Decision Tree and SVM classifiers I scaled all the features using the Min-Max feature scaler. This was incredibly important as the features had different units of measurement which varied by several orders of magnitude. By scaling the features it meant that I could use these classifiers and know that the features would be weighted evenly.

Algorithm	Accuracy	Precision	Recall
Naïve Bayes	0.33547	0.14750	0.83350
Decision Tree	0.82047	0.32632	0.32550
K Means Clustering	0.83760	0.23086	0.09350
SVM	0.84119	0.26882	0.15700

After testing these different algorithms I decided to investigate the Decision Tree classifier and the Support Vector Machines as they initially produced the best overall results. Each machine learning classifier has a selection of parameters (e.g. c, kernel, gamma) that can take a range of values, **Parameter Tuning** is used to find the optimal parameter combinations. I tuned the

DecisionTreeClassifier using **GridSearchCV** which runs through all possible combinations and then returns the optimal choice. I first tuned the Decision Tree classifier on the following parameters:

- **criterion** - the function used to measure the quality of the split. This can take the value "gini" for the Gini impurity, and "entropy" for the information gain.

- **splitter** - the strategy used to choose the split at each node. This can take the value "best" to choose the best split, and "random" to choose the best random split.

The optimal combination of parameters for my model was:
DecisionTreeClassifier() - {'splitter': 'random', 'criterion': 'gini'}

Using parameter tuning I was able to produce the following results for the Decision Tree classifier:

Classifier	Accuracy	Precision	Recall
Decision Tree	0.82213	0.33283	0.33250

I then looked at tuning the **SupportVectorClassifier** on the following parameters:

- **kernel –** specifies the type of kernel function used in the classifier. A kernel is a similarity function that takes two inputs and determines how similar they are. I looked at the 'linear' 'rbf' and 'poly' kernels.

- **C** – this value trades off the misclassification of training examples against the simplicity of the decision surface, a low C makes the decision surface smooth, a high C aims at classifying all the training examples correctly. I looked at C values between 1 and 1000.

- **gamma** – defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. I tested gamma values between 0 and 1000

The optimal combination of parameters for my model was:
SupportVectorClassifier() - {'kernel': 'linear', 'C': 1, 'gamma': 0.0}

Using parameter tuning I was able to produce the following results for the Support Vector classifier:

Classifier	Accuracy	Precision	Recall
Support Vector	0.87042	0.04656	0.01738

The parameter tuning for the Support Vector Classifier had the effect of improving the accuracy score but had a detrimental effect on both precision and recall. The tuning of the Decision Tree Classifier made marginal improvements on accuracy, precision and recall to levels above the desired minimum result. I therefore decided to use the Decision Tree Classifier as my final algorithm.

Validation

Validation is performed to ensure that a machine learning algorithm generalises well. A classic mistake is overfitting, this is where a model is trained and performs very well on the training dataset but a lot worse on the test datasets. The Enron dataset is small and skewed towards non-POIs, I needed to ensure that I used a technique that accounted for that. To avoid this I used **cross_ validation** to split the Enron data into a test and a training set. I took the average precision and recall

over 1000 randomised tests splitting the data into 30% test and 70% training. The chance of randomly splitting skewed and non-representative sets was high, therefore stratification was used in the cross validation function to ensure a percentage of samples for each class were present in each part of the split, ensuring accurate results.

Evaluation

Precision is the rate at which the algorithm correctly predicts the POI. It is the ratio of true positives to the records that are actually POIs. It is calculated by: *True Positives/(True Positives + False Negatives).* My DecisionTreeClassifier had a precision of **0.32632**

Recall measures the sensitivity and refers to the proportion of the POI the model can detect of all the POI. Recall is calculated as *True Positives/(True Positives + False Positives)*. My model achieved a recall score of **0.32550**.

In this investigation accuracy was not a good metric given the high number of non-POIs in the dataset. For example if non-POI had been predicted for all records then an accuracy of over 87% could have been achieved.

Conclusion

The Enron dataset provided a very interesting challenge of how to apply machine learning methods to a small but complex dataset. The data needed to be cleaned and then a combination of machine learning classifiers, parameter tuning and cross-validation techniques led to the creation of a reliable predictive model for POIs which tried to maximise the recall and precision while maintaining a high accuracy score.

I enjoyed the challenge of studying machine learning techniques and the workflows required for data analysis in Python.

References

Udacity module page - https://www.udacity.com/course/intro-to-machine-learning--ud120

Enron Wikipedia - https://en.wikipedia.org/wiki/Enron

Sci-kit learn page - http://scikit-learn.org/stable/

SelectKBest page - http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

Mix Max Scaler - http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

Naïve Bayes - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

Decision Trees - http://scikit-learn.org/stable/modules/tree.html

K Means Clustering - http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Support Vector Machines (SVM) - http://scikit-learn.org/stable/modules/svm.html

Validation - http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html