Project 5: Identify Fraud from Enron Email

By Wenny Wu

Introduction

In 2002, Enron, one of the largest companies in the United States, went into bankruptcy. The resulting federal investigation found widespread corporate fraud, leading thousands of confidential records, including emails and financial data for top executives, to be released to the public.

This project uses scikit-learn and machine learning algorithms to construct a point-of-interest (POI) identifier based on the emails and financial data to identify Enron employees who may have committed fraud.

Question 1

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

The goal of this project was to utilize the Enron dataset to build a predictive model that could identify persons of interest. The dataset contains a hand-generated list of POIs in the fraud case, which includes individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. Because the dataset already indicated whether a person was actually a POI or not, the value of this model on the existing Enron dataset is limited; however, one can imagine the significant value of this model when applied to fraud cases for other companies.

The dataset contained 146 records and 18 POIs:

- **14 financial features** (all units are in USD) - *['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']*
- **6 email features** (units are generally number of emails messages; notable exception is 'email_address', which is a text string) - *['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']*
- **1 labeled feature for POI** (boolean, represented as integer) – *['poi']*

An initial exploratory analysis of the Enron dataset identified three points to be removed:

- **THE TRAVEL AGENCY IN THE PARK**: This record did not represent an individual and only contained a value for 'total_payments.'

- **TOTAL**: This record did not represent an individual and was an extreme outlier for most numerical features.
- **LOCKHART EUGENE E**: This record contained ‘NaN’ for all features and hence, no useful data.

After outlier-removal, 143 records remained.

Question 2

What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not?

To determine which features to include in the POI identifier, I used the automated univariate feature selection algorithm sklearn.feature_selection SelectKBest to select the 10 most influential features. The dataset features were first scaled using the sklearn.preprocessing MinMaxScaler() module to ensure the different financial and email features, which had different units and spanned ranges of different orders of magnitude, would be weighted evenly. The top 10 most influential features with their associated scores are listed below:

Feature	**Score**
'exercised_stock_options'	*24.815*
'total_stock_value'	*24.813*
'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*

I engineered three new features: 'poi_email_ratio', 'assets', and 'tsvb_salary_ratio'.

- 'poi_email_ratio' is a ratio of ('shared_receipt_with_poi' + 'from_poi_to_this_person' + 'from_this_person_to_poi') and ('to_messages' + 'from_messages') and was intended to explore frequency of email interactions between POIs
- ‘assets’ is a sum of ‘exercised_stock_options', 'total_stock_value', 'bonus', and 'salary' and was intended to explore the combined wealth of POIs

- 'tsvb_salary_ratio' is a ratio of (‘total_stock_value' + 'bonus') and ‘salary’

Question 3

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

I tested many of the algorithms introduced in the Introduction to Machine Learning lectures by Udacity, including Gaussian Naïve Bayes, Decision Trees, Support Vector Machine (SVM), Adaboost, and Random Forest. Iterating over each of these classifiers, I found the accuracy, precision, and recall scores, listed below:

Classifier	Accuracy	Precision	Recall
Gaussian Naïve Bayes	*0.86*	*0.40*	*0.40*
Decision Tree	*0.84*	*0.33*	*0.40*
Support Vector Machine	*0.88*	*0.50*	*0.20*
Adaboost	*0.79*	*0.17*	*0.20*
Random Forest	*0.88*	*0.50*	*0.20*

SVM and Random Forest produced the highest accuracy scores but low recall scores. Gaussian Naïve Bayes and Decision Tree produced the highest recall score. I ultimately chose Gaussian Naïve Bayes as the classification algorithm to test due to its higher precision and recall score.

Question 4

What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm?

For this exercise, I chose to fit the data with a Gaussian Naïve Bayes classification algorithm, which does not have parameters to tune. However, many machine learning algorithms can be tuned to optimize parameters to better fit the data and make more accurate and precise predictions. If I had chosen the Decision Tree classification algorithm, for example, I would have tweaked the min_samples_split parameter to try and achieve a better precision and recall score.

In machine learning, one needs to be cautious of how algorithm parameters and features are tuned to reduce the risk of bias or variance influencing the results. High bias can lead to an oversimplified model that shows high error on the training set. High variance can lead to overfitting of the data and high error on the test set and less accurate predictions on new data.

In this example, I did try adjusting the test_size and random_state parameters in the sklkearn.cross_validation train_test_split module, but the adjustments had no effect on

accuracy, precision, and recall scores. I also tested the effect of including/excluding my engineered features ‘poi_email_ratio’, ‘tsvb_ratio’, and ‘assets’ to observe their effect on the precision and recall scores. I started by running Udacity’s tester.py with just the Gaussian Naïve Bayes classifier with the 10 most influential features as scored by the SelectKBest algorithm to generate accuracy, precision, recall, and F1-scores and the confusion matrix:

Gaussian Naïve Bayes	SelectKBest 10 features		
Accuracy	*0.82*	*True positives*	*623*
Precision	*0.33*	*False positives*	*1291*
Recall	*0.31*	*False negatives*	*1377*
F1	*0.32*	*True negatives*	*11709*

I then added in each engineered feature into the features_list and re-ran tester.py to observe any changes in the evaluation metrics. It turns out that adding the ‘poi_email_ratio’ feature or the ‘tsvb_ratio’ had no effect on the accuracy, precision, and recall scores, so I did not add these features to the final features_list.

Gaussian Naïve Bayes	SelectKBest 10 features + ‘assets’		
Accuracy	*0.83*	*True positives*	*604*
Precision	*0.34*	*False positives*	*1194*
Recall	*0.30*	*False negatives*	*1396*
F1	*0.32*	*True negatives*	*11806*

Adding the ‘assets’ feature slightly increased precision at the cost of recall. The number of true positives decreased, offset by an increase in the number of true negatives. In this case, I chose to not add this feature to the final features_list to maintain the higher recall score.

Question 5

What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation provides a sense that the machine learning algorithm can generalize onto an independent dataset or new data points. It also serves as a check for the classic mistake of overfitting, where the model fits well to the training dataset but performs significantly worse on the test datasets. In this project, I validated my analysis with the K-Folds (k=4) cross validation iterator:

Gaussian Naïve Bayes	*SelectKBest 10 features | KFold (k = 4)*
Accuracy	*0.89*
Precision	*0.5*
Recall	*0.5*

Question 6

Give at least 2 evaluation metrics and your average performance for each of them.

The primary evaluation metrics were precision and recall. Precision is the ratio of true positives to true and false positives ($\frac{TP}{TP+FP}$), providing insight into how well the model distinguishes between true POIs and false alarms. Recall is the ratio of true positives to true positives and false negatives ($\frac{TP}{TP+FN}$), describing how sensitive the model is in detecting the POIs. Given the small number of POIs in the entire dataset (12.3%), the accuracy score does not provide as much information as the precision and recall scores. To summarize, the Naïve Bayes model produced the following scores for the evaluation metrics of interest:

Gaussian Naïve Bayes	*train_test_split (test_size=0.3, random_state=42)*	*KFold (k = 4)*
Accuracy	*0.82*	*0.89*
Precision	*0.33*	*0.50*
Recall	*0.31*	*0.50*

Conclusion

This exercise contained a few surprises for me. For one, I did not expect the email data to have so little effect on the results, and the new features I created did not appear to impact the recall and precision scores for the algorithm. I was also surprised at how the simpler algorithms (i.e. Naïve Bayes, Decision Tree) tended to perform better than the more complex algorithms and can be more robust and resistant to overfitting since they have fewer parameters to tune. For future studies, I would like to explore other classification algorithms available in scikit-learn to compare their relative performance.

For evaluation metrics, I placed heavier emphasis on maintaining a higher recall score and chose not to include my ‘assets’ feature in the final model even though it could produce a slightly better precision score. In this particular case of corporate fraud, I would argue that recall is the most important evaluation metric. To identify as many culpable suspects to further

investigate is more important than trying to avoid false alarms and possibly letting a guilty individual escape scrutiny.

Resources and References

- Udacity Introduction to Machine Learning
- Udacity Discussion Forum
- scikit-learn Documentation
- https://github.com/udacity/ud120-projects

Files

~ud120-projects\final_project

- pickle files
- poi_id.py
- tester.py
- explore_enron_data.py
- enron_outliers.py