Identifying Fraud from Enron Emails (and Financial Data)

In 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.

In this project, I used python's scikit-learn library that contained numerous machine learning algorithms. I built a person of interest identifier based on financial and email data made public as a result of the Enron scandal. This data has been combined with a hand-generated list of persons of interest (POI) in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Run `poi_id.py` to view the selected and tuned feature selections, and the machine learning algorithms' performances. Run `tester.py` to view the performance of the chosen algorithm, K-nearest neighbors classifier. The following is a snippet of the scores returned from `tester.py`:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=3, p=2, weights='uniform')
	Accuracy: 0.86869	Precision: 0.63679	Recall: 0.34100	F1: 0.44415	F2: 0.37592
	Total predictions: 13000	True positives:  682	False positives:  389
False negatives: 1318	True negatives: 10611
```

Free-response Questions

> 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 is to choose a combination of features, financial and email, of former Enron employees and choose an appropriate machine learning algorithm to predict whether that person is considered a person of interest (POI) or not. It should be noted that this dataset contained labeled data where POIs were flagged and as such the purpose of this project is to build a supervised classification model that could be used on other unlabeled datasets.

The Enron dataset contained 146 records with 14 financial features, 6 email features, and 1 labeled feature for POI. Out of the latter dataset, 18 people were labeled as persons of interest (POI). Interestingly, I had to omit `email_address` from the email features list due to it being a string while all the other features were integer values; it was hard to justify keeping unique email addresses as a POI identifier for fraud. Furthermore, I visualize the dataset's feature to conduct some basic exploratory data analysis. I erred on the side of not removing values with large residual errors due to this model being a classifier as opposed to a quantitative regression one. But the following three records were removed:

- *TOTAL* - A statistical artifact for reporting purposes that was of no value for the model.
- *LOCKHART EUGENE E* - This record contained no useful data with NaN across the board.
- *THE TRAVEL AGENCY IN THE PARK*: This record is clearly not a person.

> 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? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values.

I ended up using `salary`, `exercised_stock_options`, `bonus` along with the `poi` label. Surprisingly, and very frustratingly, I happened on these features randomly. To optimize my performance on the algorithms being tested, I used a `min-max scaler`, `SelectKBest` and `PCA` (with pipeline). The scaler was very important to normalize the data as features had different units and varied widely by several orders of magnitude in some cases. The K-best approach was used to select the 10 most important features that drive POI labels - there was a marked drop in scoreswith more features. The following table shows the eleven k-best scores and the percentage of invalid (NaN) entries they contained:

Features	Scores	Percent invalid
exercised_stock_options	24.815	28.7
total_stock_value	24.183	11.9
bonus	20.792	42.7
total_compensation	19.225	56.8
salary	18.290	33.6
deferred_income	11.458	65.7
long_term_incentive	9.922	53.8
restricted_stock	9.212	23.1
total_payments	8.772	12.6
shared_receipt_with_poi	8.589	39.9
loan_advances	7.184	97.9

It was alarming to see the high percentages of invalid records, such as `loan_advances`. Seeing how the scores started to dip by the eleventh, `loan_advances`, and the fack that there were a lot of invalid entries there, I decided to cut off my feature set at the top ten. It was also odd seeing only one email feature, `shared_receipt_with_poi` in the best features list. I decided not to pursue the email features as there were better suited financial features and it seemed very likely that for a very high risk activity as fraud, financial gains would be a key motivator. Therefore, I engineered a `total_compensation` feature that included `salary`, `bonus`, and `total_stock_value`. Unfortunately, this feature was not included in the final algorithm as it brought down the precision and recall metrics from 0.63679 and 0.34100 to 0.6021 and 0.3111, respectively.

After numerous iterations of the features list used in ML analysis, I found that the constituent features of the engineered `total_compensation` feature gave a very favorable score. Swapping `exercised_stock_options` for `total_stock_value` gave a 5% better precision (0.63679) and recall (0.34100) scores ; cash and liquidity is king when you need to run from the law on short notice.

> What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? 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? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier).

The chosen algorithm was the K-nearest neighbors classifier. I selected it purely based on its performace in accuracy, precision and recall, as well as it's simplicity to use. I tried using the following models as well: "Linear SVM", "RBF SVM", "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes", "LDA", "QDA", "Logistic Regression".

I tried tuning the algorithm initially with `PCA` to some disastrous effects. I promptly abandoned `PCA` in favour of `GridSearchCV` with some improvements in the following parameters and using a stratified shuffle split on a thousand fold:

- KNeighborsClassifier: `metric`, `weights`, and `n_neighbors`
- KMeans: `n_clusters`, and `tol`

The `KNeighborsClassifier` with n_neighbors of 3 gave the best results - more so than KMeans - and was thus selected. I saw a marked increase in precision and recall from 0.34744 and 0.30474 to 0.63678 and 0.44415, respectively. So I commented out the PCA pipeline optimizer and cleaned up the GridSearchCV parameters to have the ideal performance. Here are the results:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=3, p=2, weights='uniform')
    Accuracy: 0.86869   Precision: 0.63679  Recall: 0.34100 F1: 0.44415 F2: 0.37592
    Total predictions: 13000    True positives:  682    False positives:  389
False negatives: 1318   True negatives: 10611
```

> What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

Validation is performed to ensure that a machine learning algorithm generalizes well. The dataset is small and skewed towards non-POI; if non-POI had been predicted for all records, an accuracy of 87.4% would have been achieved. I needed a technique that accounts for that else the risk of poor performance is high. The chance of randomly splitting skewed and non representative sets could be high as well, therefore the need to use stratification (preservation of the percentage of samples for each class) to achieve robustness in a dataset with the aforementioned limitations.

A classic mistake is over-fitting, where the model is trained to perform very well on the training dataset, but markedly worse on the cross-validation and test datasets. I built a helper function called `clf_evaluator` that iterates over a thousand times with the data subsetted with a 3:1 training to test ratio. It generated the following average accuracy, precision and recall metrics.

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	0.85628	0.43429	0.31000
KMeans	0.85524	0.43220	0.29915
SVC (linear)	0.85421	0.00000	0.00000
SVC (gamma=2)	0.85877	0.12917	0.03309
DecisionTreeClassifier	0.80544	0.33356	0.28450
RandomForestClassifier	0.84544	0.44635	0.21223
AdaBoostClassifier	0.80289	0.32749	0.25727
GaussianNB()	0.84667	0.47131	0.28897
LDA	0.85465	0.46062	0.19268
QDA	0.84386	0.46832	0.24596
LogisticRegression	0.85693	0.46794	0.19322

Accuracy refers to the ratio of correct predictions out of its total. Precision captures the ratio of true positives to the records that are actually POIs. This highlights the false positives that happens in regards to POI. Recall captures the ratio of true positives to the records labeled as POIs, which describes sensitivity. Since POIs in this dataset were essentially needles in a haystack, accuracy is not a useful metric as can be seen with SVC (linear) where the precision and recall values were null but a high accuracy was returned.

The above table provided a quick and dirty snapshot of the different algorithm's performances, and from which I selected KNeighborsClassifier and KMeans. They were tuned using GridSearchCV and stratified shuffle split as highlighted above. I observed varied results from the tuning process.

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	0.86127	0.62121	0.34375
KMeans	0.35271	0.33228	0.11111

Oddly, KMeans performed poorly after the tuning stage while KNeighborsClassifier saw some marked improvements. The latter had a higher recall value that gives confidence that true POIs would be flagged for further investigation. Thus the KNeighborsClassifier was chosen as the final machine learning algorithm for this Enron dataset.

Note

`data/` contains the input enron dataset and output pickle files used. `tools/` contains helper scripts used, most notably, `helper.py` that contains many custom functions used in this application.

References

- scikit-learn Documentation
- How to Improve Machine Learning Results
- Introduction to Machine Learning
- 8 Best Machine Learning Cheat Sheets