Whitney King

Machine Learning

6/28/2017

Enron Fraud Person of Interest Email Analysis

# Introduction

The fall of Enron occurred in December of 2001, after fraud and corruption gave way to bankruptcy. This massive failure, and the disaster it left in its wake, has gone down as one of the largest and most notorious corporate collapses in history. When the information from Enron’s corporate records was made available to the public during the investigation, so was one of the largest open sets of complete emails available. This analysis focuses on a preprocessed dataset derived from these emails (provided via Udacity resources), which is aimed at creating a classifier that can determine persons of interest (POI) connected to the fraud/corruption.

The dataset consists of 146 employee data points. When these were investigated for outliers, it was found that the **TOTAL** line was being treated as an employee instead of an aggregate, so this entry was popped, putting the total employee data points at 145. Out of these, 18 were pre-categorized as POI and 127 were not. To start off, each employee contains 18 features (either financial or email), and the POI label. Many features contained NaN or missing values (Appendix I), so these were handled in the Python script to ensure they didn’t impact the outcome.

# Features

As the data is further cleaned and processed, two additional features were added, bringing the total features up to 20. The feature **to\_poi\_message\_ratio** was added to measure the frequency someone sends emails to POIs, and **from\_poi\_message\_ratio** was added to measure the frequency someone receives emails from POIs. These were interesting features to create and investigate, however they didn’t end up being important enough for the final feature selection.

To determine the best features, cross validation was performed using **SelectKBest** with the **f\_classif** function. For , where k is the number of features, scores were determined for each of the chosen classifiers for each feature. RandomForest returned 4 best features, while AdaBoost returned 13 of them. For RandomForest, the top three feature scores were **salary** (18.58), **total\_stock\_value** (8.87), and **exercised\_stock\_options** (7.24). For AdaBoost, the top 3 feature scores were **restricted\_stock** (25.1), **exercised\_stock\_options** (24.47), **deferred\_income** (21.06). Feature scaling wasn’t done, since it isn’t needed for either classifier used.

# Algorithms

The algorithms chosen to fit to this dataset were **RandomForest** and **AdaBoost**. The classifiers were fit with the data, then used to evaluate best features, accuracy, precision, and recall. The scores were plotted (Figure 1 and Figure 2) to visualize the output. Finally, averages of these scores were returned for an idea of which classifier fit the dataset best overall (Appendix II).

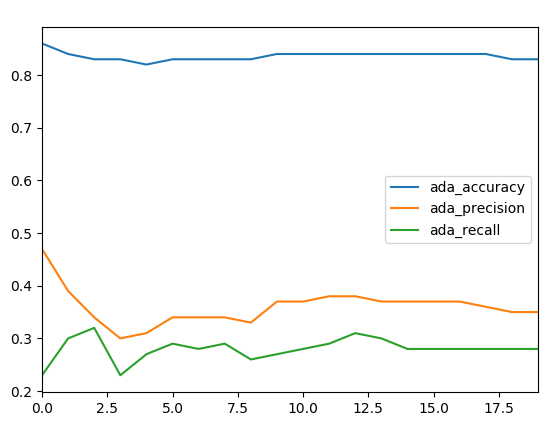
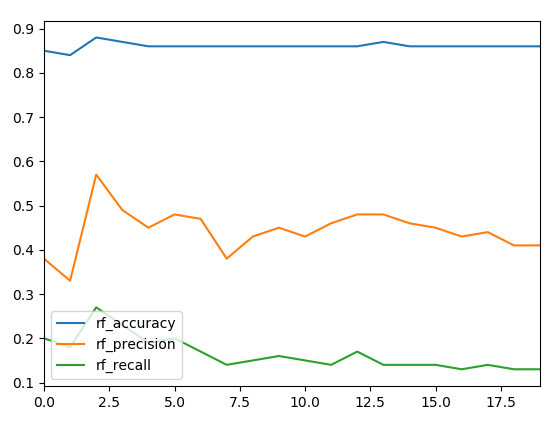
Those calculated averages were then used to figure out which algorithm might be the better fit for the selected best features. While AdaBoost had a lower average accuracy and precision, the recall was much closer to the benchmark of .3 that needs to be attained, so that one looks like it would be the better choice for tuning.

Figure 2 - RandomForest

Figure 2 - AdaBoost

# Tuning

Figure 3 - AdaBoost

Figure 4 – RandomForest

Tuning the parameters for each classifier is a means in which to ensure you’re going to get the best performance and reliability from the decisions it’s making on future sample data. Prior to being tuned, the average recall scores for AdaBoost and RandomForest (.29 and .18 respectively) were both lower than we needed (.3). **GridSearchCV** was used to permutate through various parameter values for each classifier to figure out what the best case for performance going forward is.

The parameters used in this tuning process were **n\_estimators** and **learning\_rate** for AdaBoost, and **n\_estimators, min\_samples\_split**, and **max\_features** for RandomForest. For AdaBoost, the best parameter for **n\_estimators** was 90 and **learning\_rate** was 0.6. For RandomForest, the best parameter for **n\_estimators** was 125, **max\_features** was 3, and **min\_samples\_split** was 2. In general, higher values for **n\_estimators** and **max\_features** will lead to a more complicated classifier. Conversely, the higher the value for **min\_sample\_splits** is, the less complicated the classifier is.

# Validation

**StratifiedShuffleSplit** was used as a form of cross validation to understand more statistics about our best features with tuned classifiers. This method was chosen due to the relatively small nature of the dataset, as it’s well suited for this type of scenario. Performing this evaluation gives us an idea of how we can expect the algorithm to perform when fed future data, and is overall a better-quality measurement than just using sample data alone. If validation isn’t performed, it’s easy to fall into scenarios where you won’t be getting the best results you could be.

# Metrics

After the tuning and validations was completed, the output was investigated (Appendix III), and the most important statistics are compared below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifier** | **Accuracy** | **Precision** | **Recall** |
| AdaBoost | *0.85347* | *0.4358* | *0.336* |
| RandomForest | *0.83969* | *0.46447* | *0.2745* |

To calculate precision , we can use the following equation employing true positive and false positive counts:

To calculate the recall , we can use the following equation employing true positive and false negative counts:

Once tuned, we can see that AdaBoost has a higher accuracy and recall, but a slightly slower precision than RandomForest. The recall from RandomForest didn’t get above .3 with the tuned parameters, and had a lower accuracy. Due to these factors, AdaBoost was chosen as the final classifier since it will do a better job overall, and meets the required benchmarks.

# Conclusion

This was an interesting project that I didn’t fully grasp the concept of until almost all the way through coding the required Python script. Having all of the POIs preidentified didn’t make sense until I really dissected what the classifier tuning was doing with the different parameters. Once I related this back to the data structure, it made sense why we’d want to know who was a POI and who wasn’t going into training the algorithm, since we’d be concerned with classifying future or different sample data than what we’d use in this set down the line. I did struggle a lot with getting the recall for the classifiers over .3, but once I changed the seed, I was able to get it where it needed to be for AdaBoost.

Overall, I feel I learned a lot, and I’m looking forward to finding way to employ the features of scikit-learn to other datasets in the future, since the statistical part of this whole course has been very engaging for me.

# Appendix I: NaN Values

**# NaNs in Each Feature:**

poi: 0

salary: 51

deferral\_payments: 107

total\_payments: 21

loan\_advances: 142

bonus: 64

restricted\_stock\_deferred: 128

deferred\_income: 97

total\_stock\_value: 20

expenses: 51

exercised\_stock\_options: 44

long\_term\_incentive: 80

restricted\_stock: 36

director\_fees: 129

to\_messages: 60

from\_poi\_to\_this\_person: 60

from\_messages: 60

from\_this\_person\_to\_poi: 60

shared\_receipt\_with\_poi: 60

# Appendix II: Average Classifier Scores

**Average Scores For Each Classifier:**

Average AdaBoost Accuracy: 0.84

Average AdaBoost Precision: 0.36

Average AdaBoost Recall: 0.28

Average RandomForest Accuracy: 0.86

Average RandomForest Precision: 0.44

Average RandomForest Recall: 0.17

# Appendix III: Tuned Scores

Beginning tuning AdaBoost...

Please wait...

Best Parameters:

{'learning\_rate': 0.6, 'n\_estimators': 90}

Time Spent Tuning AdaBoost: 1482.87s

Tuned AdaBoost Metrics:

AdaBoostClassifier(algorithm='SAMME.R', base\_estimator=None,

learning\_rate=0.6, n\_estimators=90, random\_state=None)

Accuracy: 0.85347 Precision: 0.43580 Recall: 0.33600

F1: 0.37945 F2: 0.35213

Total predictions: 15000

True positives: 672

False positives: 870

False negatives: 1328

True negatives: 12130

Beginning tuning RandomForest...

Please wait...

Best Parameters:

{'max\_features': 3, 'min\_samples\_split': 2, 'n\_estimators': 125}

Time Spent Tuning RF: 50757.37s

Tuned RandomForest Metrics:

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features=3, max\_leaf\_nodes=None,

min\_impurity\_split=1e-07, min\_samples\_leaf=1,

min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

n\_estimators=125, n\_jobs=1, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

Accuracy: 0.83969 Precision: 0.46447 Recall: 0.27450

F1: 0.34507 F2: 0.29895

Total predictions: 13000

True positives: 549

False positives: 633

False negatives: 1451

True negatives: 10367