Identify Fraud from Enron Email

D. Chris Young

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?

Introduction

The Enron Corporation is almost entirely famous today for all of the wrong reasons. Before its downfall in late 2001, Enron was a major player in the energy sector with apparent revenues of $111 billion across electricity, natural gas, communications, pulp and paper companies. However, it was all due to an elaborate corporate scheme involving accounting fraud and corruption by key personnel. The one positive that came out of the scandal is the after life of a large email archive collected as a part of the litigation. The corpus was later purchased and released to the public for research purposes. As a result, the corpus is a treasure trove of data with applications in machine learning projects and natural language processing.

In this project, supervised machine learning techniques were used to predict whether an individual in a dataset was a person of interest (POI). In this context, a POI is someone that was indicted, settled without admitting guilt or testified in exchange for immunity. These names were extracted from a USA Today article “A look at those involved in the Enron scandal”.

Data Exploration and Outlier Investigation

The dataset for this project was preprocessed for data exploration. Finance data for key individuals was integrated with aggregate email statistics from the corpus. Additional exploration steps were performed including converting the format to a dataframe with the name of the individual added as a new column for csv file export, checking for missing values for each data point, checking for valid values for each person and counting the number of labels. Below is a summary of key statistics:

- Total number of data points: **146**
- Number of features: **21**
- Features and count of missing values:
    1. bonus: **64**
    2. deferral_payments: **107**
    3. deferred_income: **97**
    4. director_fees: **129**
    5. email_address: **35**
    6. exercised_stock_options: **44**

7. expenses: **51**
8. from_messages: **60**
9. from_poi_to_this_person: **60**
10. from_this_person_to_poi: **60**
11. loan_advances: **142**
12. long_term_incentive: **80**
13. other: **53**
14. poi: **0**
15. restricted_stock: **36**
16. restricted_stock_deferred: **128**
17. salary: **51**
18. shared_receipt_with_poi: **60**
19. to_messages: **60**
20. total_payments: **21**
21. total_stock_value: **20**

- Allocation across classes:
    1. Non-POI: **128**
    2. POI: **18**

The first method to identify potential outliers was to check for any values in any of the features that were less than the 5% quantile or greater than the 95% quantile. The results reflect multiple records with values outside of the quantile boundaries. Based on the facts of the case, this made sense for some of the finance data points and key individuals within Enron. Therefore, the records for actual people were not removed. However, the TOTAL record is the grand total row from the finance data and not an actual person. This outlier is a quirk and was removed from the dataset. Below is an example of records from the quantile analysis for salary greater than the 95% quantile:

1. FREVERT MARK A
2. LAY KENNETH L
3. PICKERING MARK R
4. SKILLING JEFFREY K
5. TOTAL

As shown in the missing value data, total_payments and total_stock_value are the two features that have the least missing values. This makes sense since each person in the finance data should have at least some payment or stock value data. In addition to the POI label, that is 3 features that should have data for each data point. Below are the data points that have less than 4 valid values:

1. WHALEY DAVID A: 3
2. WROBEL BRUCE: 3
3. LOCKHART EUGENE E: 1
4. THE TRAVEL AGENCY IN THE PARK: 3

5. GRAMM WENDY L: 3

David Whaley, Bruce Wrobel and Wendy Gramm are valid since they had 1 finance value, total amount and POI. Eugene Lockhart only has the POI label and The Travel Agency in the Park is not a person so these were removed from the dataset. The footnotes indicate payments were made to the travel agency for business related travel and it was owned by the sister of Enron's former chairman.

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 importance 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.

Feature Selection and Engineering

Initial feature selection was performed using SelectKBest univariate statistical tests and identifying features with the highest k scores. New features that were engineered was cash_received and cash_received_pctg. The logic was to combine the finance data points (bonus, exercised_stock_options, loan_advances and salary) that represented actual cash received vs. long-term or deferred compensation and to express this amount as a percentage of total_payments and total_stock_value. For example, a POI may have gained more from actual total cash received vs. long-term income that was never realized. Another new feature based on the email data points was the number of POI messages to and from as a percentage of total messages. The logic here is a POI would interact with another POI more often than a non-POI.

Below are the k scores for the 19 features excluding poi and email_address as well as the 3 new engineered features:

1. bonus: **20.79**
2. cash_received: **19.59**
3. cash_received_pctg: **3.08**
4. deferral_payments: **.23**
5. deferred_income: **11.46**
6. director_fees: **2.13**
7. exercised_stock_options: **24.82**
8. expenses: **6.09**
9. from_messages: **0.17**
10. from_poi_to_this_person: **5.24**
11. from_this_person_to_poi: **2.38**

12. loan_advances: **7.18**
13. long_term_incentive: **9.92**
14. other: **4.19**
15. poi_messages_pctg: **5.40**
16. restricted_stock: **9.21**
17. restricted_stock_deferred: **0.07**
18. salary: **18.29**
19. shared_receipt_with_poi: **8.59**
20. to_messages: **1.65**
21. total_payments: **8.77**
22. total_stock_value: **24.18**

The features deferral_payments and restricted_stock_deferred were removed due to very low k scores of .23 and .07 respectively. The feature director_fees also had a lower k score of 2.13. Upon reviewing the footnotes, director_fees were cash payments and stock grants to non-employee directors. The missing value analysis reflect only 17 individuals received director_fees so this feature was also removed to eliminate potential bias to either class. The feature other includes miscellaneous type payments and with a k score of 4.19 was also removed.

It was observed the features used to derive the new cash_received feature have the highest k scores in addition to the new feature itself. The cash_received_pctg k score of 3.08 was much lower than the underlying features cash_received at 19.59 and total_value at 16.99. Therefore, the new feature cash_received_pctg was removed but cash_received was included in the initial algorithm selection process to assess the effect on performance.

The scaled poi_messages_pctg k score of 5.40 is higher than the underling features from_poi_to_this_person, from_this_person_to_poi, from_messages, to_messages at 5.24, 2.38, .17 and 1.65 respectively. In addition to shared_receipt_with_poi, the new scaled feature poi_messages_pctg were the only email features that was included in the initial algorithm selection process to assess the effect on performance.

All features were rescaled using the sklearn min/max scaler since the finance and email data points were measured by different units and on significantly different scales.

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

Algorithm Selection

As discussed above, several features were removed based on the k scores using SelectKBest univariate statistical tests. Six algorithms were run with the default parameters prior to adding the new engineered features. The tester.py results for accuracy, precision and recall are shown below.

Table 1

Algorithm	Accuracy	Precision	Recall
GaussianNB	.77093	.23545	.31950
SVC	N/A	N/A	N/A
DecisionTreeClassifier	.80740	.28763	.30100
KNeighborsClassifier	.87640	.63878	.16800
AdaBoostClassifier	.83713	.34190	.23950
RandomForestClassifier	.85647	.38799	.13250

The 3 new features were added and the performance results were rerun.

Table 2

Algorithm	Accuracy	Precision	Recall
GaussianNB	.77000	.23208	.31400
SVC	N/A	N/A	N/A
DecisionTreeClassifier	.81687	.30992	.30450
KNeighborsClassifier	.89400	.77333	.29000
AdaBoostClassifier	.84327	.36834	.24550
RandomForestClassifier	.86540	.48383	.14200

With the exception of precision for KNeighborsClassifier and RandomForestClassifier, there was no real change in the results. Since cash_received is simply an aggregate of bonus, exercised_stock_options, loan_advances and salary these were removed to balance the number of features in the model. The performance results were rerun.

Table 3

Algorithm	Accuracy	Precision	Recall
GaussianNB	.84627	.40116	.31050
SVC	N/A	N/A	N/A
DecisionTreeClassifier	.82287	.33517	.33400
KNeighborsClassifier	.89007	.84479	.21500
AdaBoostClassifier	.84333	.36328	.23250
RandomForestClassifier	.85787	.38298	.10800

GaussianNB had pretty good improvement in accuracy and precision and actually establishes a good baseline for the other algorithms that have parameters that can be tuned. DecisionTreeClassifier and AdaBoostClassifier change was pretty indiscriminate. KNeighborsClassifier had good precision improvement but recall took a hit. RandomForestClassifier took a hit with precision and recall.

My objective is to maximize accuracy, precision and recall by tuning the algorithm parameters and to identify the optimal number of features that balances bias and variance. The features used for the initial results shown in Table 3 are the baseline for the next step in the process. SVC has not had results yet since precision and recall could not be calculated with zero true and false positives so this will be explored further. DecisionTreeClassifier, AdaBoostClassifier and RandomForestClassifier are all pretty comparable but DecisionTreeClassifier will be tuned further since precision and recall already exceed .3. KNeighborsClassifier is the leading choice with really strong accuracy and precision but recall needs improvement.

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).

Algorithm Tuning

Tuning is an important step to identify the variable parameters (if applicable) of an algorithm that improves performance measured by an evaluation metric. If the tuning process is not done well then typically you will not achieve the optimal performance results for the particular dataset, features and algorithm.

SVC was listed as N/A since it returned zero true positives and false positives resulting in the divide by zero message in the tester.py script. Various kernel options were manually attempted and a max_iter value was added to terminate the solver early to view the results. Recursive feature elimination with cross validation was run to determine the optimal number of features. There was only 1 optimal feature (cash_received) so all other features were removed. The final parameters used was a linear kernel and 1000 max_iter. Final performance results are shown below.

Table 4

Algorithm	Accuracy	Precision	Recall
SVC	.45646	.14274	.50600

For the DecisionTreeClassifier and KNeighborsClassifier, StratifiedKFold cross-validation was used for the GridSearchCV. One risk of overfitting the classifier is evaluating different algorithm parameters on the same data until it performs optimally. StratifiedKFold splits the testing set and runs k separate learning experiments to produce the best estimator. 10 k folds were used.

The DecisionTreeClassifier performance results were already above the minimum criteria using the standard algorithm parameters. The feature_importances_ output was reviewed to identify features that could be removed to simplify the model and improve performance results. Below is the feature_importances_ output for the DecisionTreeClassifier.

1. cash_received: **.42**
2. deferred_income: **.15**
3. expenses: **.03**
4. long_term_incentive: **0**
5. poi_messages_pctg: **.22**
6. restricted_stock: **.10**
7. shared_receipt_with_poi: **0**
8. total_payments: **.04**
9. total_stock_value: **.04**

The features long_term_incentive and shared_receipt_with_poi had zero feature importance so these features were removed. The updated performance results are below.

Table 5

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.82253	.31853	.29050

Below is the updated feature_importances_ output for the DecisionTreeClassifier.

1. cash_received: **.42**
2. deferred_income: **.11**
3. expenses: **.04**
4. poi_messages_pctg: **.16**
5. restricted_stock: **.16**
6. total_payments: **.04**
7. total_stock_value: **.09**

The features deferred_income, expenses, total_payments and total_stock_value were removed. The updated performance results are below.

Table 6

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.82971	.40184	.39300

The performance results for precision and recall with the DecisionTreeClassifier have improved simply by removing features with lower importance in the model.

GridSearchCV was run for these features for the DecisionTreeClassifier to identify the best_estimator_ parameters for criterion, max_features, max_depth, min_samples_split and min_samples_leaf. The output is shown below.

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
        max_features='auto', max_leaf_nodes=None, min_samples_leaf=1,
        min_samples_split=5, min_weight_fraction_leaf=0.0,
```

random_state=42, splitter='best')

Below are the new performance results after updating the parameters.

Table 7

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.82036	.20904	.09250

The parameter changes from the best_estimator_ actually had a significant negative impact on performance. At this point, the algorithm was manually tuned to identify the optimal performance. The min_samples_split parameter was changed to 2 and the updated performance results are below.

Table 8

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.82250	.21097	.08850

The min_samples_leaf parameter was changed to 2 and the updated performance results are below.

Table 9

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.8336	.23944	.07650

The max_depth parameter was changed to 5 and the updated performance results are below.

Table 10

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.81757	.22410	.1150

The max_features parameter was changed to none and the updated performance results are below.

Table 11

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.84021	.42542	.33800

The max_features parameter change is what negatively impacted performance. Auto uses the max_features=sqrt(n_features) for the best split so using all features in the classifier yields the best performance.

The KNeighborsClassifier was tuned next with the DecisionTreeClassifier as the baseline. GridSearchCV was run for KNeighborsClassifier to identify the best estimator parameters for n_neighbors, weights, algorithm, metric and leaf_size. The output is shown below.

KNeighborsClassifier(algorithm='auto', leaf_size=5, metric='minkowski',
        metric_params=None, n_neighbors=10, p=2, weights='distance')

Below are the new performance results after changing the parameters.

Table 12

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	.86907	.90909	.02000

The performance results for accuracy and precision got slightly better and still very strong but recall is pretty much awful. KNeighborsClassifier does not have a feature_importances_ output so different features with lower k scores in the initial feature selection process were removed to observe the results. The features deferred_income, expenses, long_term_incentive, poi_messages_pctg and restricted_stock were removed. The best estimator from the GridSearchCV also now recommended 2 n_neighbors and uniform weights. The output is shown below.

KNeighborsClassifier(algorithm='auto', leaf_size=5, metric='minkowski',
        metric_params=None, n_neighbors=2, p=2, weights='uniform')

Below are the new performance results after changing the parameters.

Table 13

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	.88593	.68549	.26700

Precision was negatively impacted but recall improved. Since the default parameters of 30 leaf_size and 5 n_neighbors had strong initial results these original parameters were re-examined but with features removed. Below are the new performance results after changing the parameters but with the same features from the prior run.

Table 14

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	.89647	.80081	.29750

Clearly on the right path to improve recall performance. The email feature poi_messages_pctg was then added back. Below are the new performance results.

Table 15

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	.89647	.80081	.29750

No real change so now the feature shared_receipt_with_poi was removed. Final performance results are shown below.

Table 16

Algorithm	Accuracy	Precision	Recall
KNeighborsClassifier	.89813	.80649	.31050

The best results were including the cash_received, poi_messages_pctg, total_payments and total_stock_value features and default parameters. Below are the best DecisionTreeClassifier and KNeighborsClassifier performance results.

Table 17

Algorithm	Accuracy	Precision	Recall
DecisionTreeClassifier	.84021	.42542	.33800
KNeighborsClassifier	.89813	.80649	.31050

The KNeighborsClassifier was selected as the algorithm for final analysis.

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

Validation

Validation is the process of testing your algorithm performance by splitting the dataset into separate training and test data. This gives an estimate of performance on an independent dataset. The classic mistake is to create an overfit classifier on the entire dataset that just repeats the labels of the samples it has just seen but would not be able to adapt to a different dataset in the wild.

The tester.py script was utilized heavily to measure results for each iteration. The cross-validation process used in this script is StratifiedShuffleSplit that returns stratified randomized folds that preserves the representation of each class in the sample. The dataset is small and skewed towards non-POI so this minimizes the risk of random chance splitting the training and test data with potential unequal distribution of the classes.

The tester.py script also utilizes 1000 iterations of the StratifiedShuffleSplit. The accuracy, precision and recall performance results are an average across each iteration. A single iteration StratifiedShuffleSplit was added to the poi_id.py script for quick validation. The variation in performance results with the single iteration StratifiedShuffleSplit compared to the tester.py script was very apparent throughout the tuning and validation process. This clearly demonstrates the importance of the multi-iteration StratifiedShuffleSplit approach to get a true indication of performance. For example, the accuracy of the classifier with a single iteration was .767442 compared to the average performance of .89813.

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.

Evaluation

Accuracy is the number of accurate predictions divided by all data points. The KNeighborsClassifier had the best accuracy with 621 true positives and 12851 true negatives divided by 15000 or **89.813%**.

Precision is true positives divided by true positives plus false positives. The KNeighborsClassifier also had the best precision with only 621 true positives and 149 false positives or **80.649%**.

Recall is true positives divided by true positives plus false negatives. The SVC had the best recall with 1012 true positives and 988 false negatives or 50.6%. However, this is a good example of not using just one evaluation metric since the SVC is overall very poor at making accurate predictions but better than the other two with smaller false negatives.

The KNeighborsClassifier does not have the best recall but it has great precision. This means if a POI is in the dataset there is a lot of confidence that it is a real POI and not a false alarm. The algorithm does not really guess when it is on the edge for a POI classification. This is very apparent when false positives of 1010 for the DecisionTreeClassifier is compared to only 149 for the KNeighborsClassifier. Recall for the KNeighborsClassifier was **31.05%**.