David Broadwater
Data Analyst Nanodegree, Dec. '14 Cohort
Project 4: Identifying Fraud From Enron Email

Enron Submission Free-Response Questions

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? [relevant rubric items: "data exploration", "outlier investigation"]

The goal of this project is to try to identify known persons of interest (POIs) in the Enron corporate fraud case by using financial and email information make public after the investigation and machine learning algorithms. POIs are individuals who were indicted, reached a settlement, or testified in exchange for prosecution immunity. Machine learning algorithms could be useful in identifying POIs by determining underlying patterns or relationships in the financial and email information and then applying them to predict POIs. The financial information was taken from `enron61702insiderpay.pdf`, and includes financial compensation information across many different categories for 145 former Enron employees. The financial data could contain useful information for identifying POIs, since it is possible the Enron employees most involved in the fraud had a financial incentive for doing so. As a result, their compensation could have been different than non-POIs. The email information was collected during the course of the fraud investigation and included things like the number of emails sent by each person and to whom (e.g., POIs or non-POIs). The email data could contain information to help identify POIs as well, especially if POIs tended to send more emails to other POIs, or just in general.

Prior to outlier detection and removal, there were 146 entries included in the dataset dictionary, with 18 POIs and 128 non-POIs. There were a total of 2190 data points, but only 1228 (56%) contained values; the rest were empty (`NaN`). During the course of investigating the email and financial information, a few outliers were identified in the financial information. The biggest outlier was associated with the `TOTAL` entry in the dataset, which represented the sums of all of the financial features. This `TOTAL` entry was discarded because it was mainly a spreadsheet quirk from the financial data source and was not tied to an individual. While perhaps some of this information could have been useful for analysis (i.e., scaling individual financial features relative to the total values), it was not used in this effort. There were other outliers associated with the financial data for some of the POIs (i.e., Kenneth Lay, Jeffrey Skilling, etc.), but they were included because they were valid data points for training the machine learning algorithms. These outliers were investigated by plotting features against each other and by comparing the maximum values for each financial feature to the financial information source document.

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? As part of the assignment, you should attempt to engineer your own feature that doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

In order to determine which features to use, the fraction of valid data points (e.g., everything but `NaN` values) was calculated for each feature across three classes (or groups): everyone in the dataset, non-POIs in the dataset, and POIs in the dataset. A list of data coverages (the fraction of people in the dataset which had values for those features) is provided below for the entire dataset, non-POIs, and POIs, respectively.

Financial Feature Coverages

```
total_payments             [0.86, 0.84, 1.00]
total_stock_value          [0.86, 0.84, 1.00]
restricted_stock           [0.75, 0.73, 0.94]
exercised_stock_options    [0.70, 0.70, 0.67]
salary                     [0.65, 0.61, 0.94]
bonus                      [0.56, 0.52, 0.89]
long_term_incentive        [0.45, 0.42, 0.67]
deferred_income            [0.34, 0.30, 0.61]
deferral_payments          [0.27, 0.27, 0.28]
loan_advances              [0.03, 0.02, 0.06]

Too much POI coverage:
expenses                   [0.65, 0.60, 1.00]
other                      [0.64, 0.59, 1.00]

Insufficient POI Coverage:
restricted_stock_deferred  [0.12, 0.14, 0.00]
director_fees              [0.12, 0.13, 0.00]
```

Email Feature Coverages

```
poi                        [1.00, 1.00, 1.00]
to_messages                [0.59, 0.56, 0.78]
from_poi_to_this_person    [0.59, 0.56, 0.78]
from_messages              [0.59, 0.56, 0.78]
from_this_person_to_poi    [0.59, 0.56, 0.78]
shared_receipt_with_poi    [0.59, 0.56, 0.78]

Too much POI coverage:
email_address              [0.76, 0.73, 1.00]
```

For the financial features, the `total_payments`, `total_stock_value`, `expenses`, and `other` financial features all had adequate overall data coverage, but had 100% coverage for POIs and less than 100% coverage for non-POIs. This scenario was problematic because it wouldn't provide the classifier algorithm any training points for cases where a POI had missing data for those features, which was the case for some non-POIs. This could lead to the classifier simply using the existence (or non-existence) of those features on a testing point to predict if it was a POI, instead of using the values of those features. As a result, those features were removed from use on their own, but some were used to create other new features. In addition, `restricted_stock_deferred`, and `director_fees` all had no POI data coverage, so they were discarded for the same reasons listed above. All of the email features had sufficient coverage for analysis use; the `email_address` feature had 100% POI coverage, so it was also discarded.

Eight new features were created for this effort, all based on determining the fraction of some "sub-feature" (i.e., `salary`) compared to it's associated "parent feature" (`total_payments` in this example, which included `salary`) for each individual. These were calculated to gain better perspective on the relationship between these "sub-features" and their "parent features", and for better comparison between individuals. The new financial features included:
`fractional_bonus` (`bonus` / `total_payments`),
`fractional_salary` (`salary` / `total_payments`),
`fractional_long_term_incentive` (`long_term_incentive` / `total_payments`),
`fractional_exercised_stock_options` (`exercised_stock_options` / `total_stock_value`), and
`fractional_restricted_stock` (`restricted_stock` / `total_stock_value`).

While `total_payments` and `total_stock_value` could not be used on their own, this strategy allowed for some of the information contained within them to be used in the analysis, since the other features used to create them did not have full POI coverage. Lastly, the new email features included:
`fraction_from_poi` (`from_poi_to_this_person` / `to_messages`),
`fraction_to_poi` (`from_this_person_to_poi` / `from_messages`), and
`fraction_shared_receipt_with_poi` (`shared_receipt_with_poi` / `to_messages`).

Min-Max scaling was used to also account for the large ranges of values within some of the features (particularly in the financial data). While this wasn't necessary for all of the machine learning algorithms used (i.e., Decision Tree), applying the scaling allowed for greater flexibility and performance in using other algorithms where feature scaling would have an impact (i.e., SVM).

Finally, the features used in the final POI identification algorithm were selected using the feature importances from the DecisionTreeClassifier, shown below.

DecisionTree Feature Importances:

```
exercised_stock_options             0.200157604413
shared_receipt_with_poi             0.177984330906
fraction_to_poi                     0.13649901634
fractional_bonus                    0.119228517488
fractional_restricted_stock         0.0978094649933
fraction_from_poi                   0.0966545848436
salary                              0.0422863808691
long_term_incentive                 0.0422863808691
from_poi_to_this_person             0.0422863808691
restricted_stock                    0.0281948316394
fractional_salary                   0.01661250677
bonus                               0.0
deferred_income                     0.0
deferral_payments                   0.0
loan_advances                       0.0
fractional_long_term_incentive      0.0
fractional_exercised_stock_options  0.0
to_messages                         0.0
from_messages                       0.0
from_this_person_to_poi             0.0
fraction_shared_receipt_with_poi    0.0
```

All of the features with a feature importance of 0.0 were discarded for use in the final identifier, with the remaining features included in the analysis. A total of 11 features were used in the final identifier (not counting the `poi` "feature", which was simply a label for each entry).

3. What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: "pick an algorithm"]

Four different classification algorithms were tested for this effort: Gaussian Naive Bayes, Decision Tree, AdaBoost, and Support Vector Machine. For each, two scenarios were tested: one where Principal Component Analysis (PCA) was applied, and one without. The Decision Tree Classifier performed the best of all of the algorithms tested based on the evaluation metrics used in this project (and described later).

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? (Some algorithms

don't 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 if you used, say, a decision tree classifier). [relevant rubric item: "tune the algorithm"]

Tuning the parameters of an algorithm refers to changing the algorithm's input parameters (usually across a range of values) and measuring the performance of each combination of them in order to determine the optimal set of input parameters. If parameter tuning is not done well, the performance of the algorithm could potentially be improved, because the "best" parameters for the problem were not used, or at least a better combination existed. For this effort, `GridSearchCV` was used to tune each algorithm using stratified shuffle split cross-validation, as used by the `tester.py` script provided by Udacity. For each case, a number of operations were applied to the data at once using the `pipeline` function, with the best performing combination of values (based on the metrics used in this project) selected by `GridSearchCV` and passed on to the tester function for evaluation.

For Gaussian Naive Bayes, AdaBoost, and SVM, `SelectKBest` was tuned to determine the optimal features to use for analysis. These features included those discarded earlier for use in the Decision Tree Classifier, since the `SelectKBest` algorithm uses different selection criteria to determine the best features to use. The parameters tuned for each algorithm or pipeline operator are shown below:

- Select K Best (`SelectKBest`): `k`
- Principal Component Analysis (`PCA`): `whiten` , `n_components`
- Gaussian Naive Bayes (`GaussianNB`): None, no input parameters to tune
- Decision Tree (`DecisionTreeClassifier`): `criterion` , `min_samples_split`
- AdaBoost Ensemble Classifier (`AdaBoostClassifier`): `n_estimators`
- Support Vector Machine (`SVC`): `C` , `gamma` , `kernel`

The best performing parameter and algorithm combination identified using `GridSearchCV` was `DecisionTreeClassifier` with `criterion` = `entropy` and `min_sample_split` = `30` , with the rest of the input parameters set to default values.

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]

Validation refers to the process of training and testing a machine learning algorithm in order to assess its performance, and to prevent overfitting. Overfitting can occur when the algorithm is fit too closely to the training data, such that it performs really well on the training data, but poorly on any other new unseen data. This is why it is important to always set aside data for testing, since testing on the same data used for training can lead to a misleading assessment of an algorithm's performance. Because machine learning is often used to try to form predictions about new data, a proper validation strategy is critical.

Aside from forgetting to actually perform validation by splitting data into testing and training sets, another classic mistake is to not shuffle and split the testing and training sets adequately. If there are patterns in the way classes are represented or inserted into the main dataset (i.e., sequentially by class), this could lead to an uneven mix of data across classes in the training sets. As a result, the algorithm could wind up being training mainly on data from one class, but tested mainly on data from another, leading to poor performance.

For this effort, all of the testing and training datasets were created using `StratifiedShuffleSplit` in Scikit-Learn, which shuffled the data and split it into 1000 different sets, called folds. This was already implemented in the testing function provided by Udacity, but was also used in the `GridSearchCV` function in order to best identify the optimal combination of parameters, and to be consistent with the Udacity testing function.

6. 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. [relevant rubric item: "usage of evaluation metrics"]

While there are many different ways to measure the performance of a machine learning algorithm, the three metrics used in this effort were precision, recall, and the F1-score. These metrics are based on comparing the predicted values (POI or non-POI, in this case) to the actual ones. Precision is calculated by dividing the number of times the algorithm positively identifies a data point (known as a true positive) divided by the total number of positive identifications (regardless of whether they were correct). For this case, a high precision value means POIs identified by the algorithm tended to be correct, while a low value means there were more false alarms, where non-POIs were flagged as POIs.

Recall is calculated by dividing the number of true positives by the sum of the true positives and the number of times the algorithm incorrectly negatively identified a data point (known as a false negative). In this case, a false negative would be represented by incorrectly labeling a POI as a non-POI. Here, a high recall value means that if there were POIs in the test set, the algorithm would do a good job of identifying them, while a low value means that sometimes POIs slipped through the cracks and were not identified.

Finally, the F1 score is a weighted average of precision and recall. It's calculated by multiplying the product of the recall and precision by two, and then dividing by the sum of the precision and recall. For this effort, both precision and recall values of 0.3 had to be achieved using one of the algorithms. In order to use `GridSearchCV` to find the optimal combination of input parameters given those performance requirements, the F1 score was used as the scoring function to select the best combination. The performance of all of the algorithms was tested using the `test_classifier` function included in the project. The precision, recall, and F1 score for the `GridSearchCV` selected input parameters for each algorithm are presented below.

```
Decision Tree Classifier
Precision: 0.48093      Recall: 0.56100     F1: 0.51789

Decision Tree Classifier with PCA:
Precision: 0.23742      Recall: 0.07550     F1: 0.11457

Gaussian Naive Bayes with Select K Best:
Precision: 0.42269      Recall: 0.33900     F1: 0.37625

Gaussian Naive Bayes with Select K Best and PCA:
Precision: 0.37241      Recall: 0.34150     F1: 0.35629

AdaBoost Classifier with Select K Best:
Precision: 0.36452      Recall: 0.26300     F1: 0.30555

AdaBoost Classifier with Select K Best and PCA:
Precision: 0.34997      Recall: 0.27000     F1: 0.30483

Support Vector Machine (SVM) with Select K Best:
Precision: 0.41406      Recall: 0.24450     F1: 0.30745

Support Vector Machine (SVM) with Select K Best and PCA:
Precision: 0.25385      Recall: 0.15650     F1: 0.19363
```

The Decision Tree classifier performed the best overall, followed by Gaussian Naive Bayes, AdaBoost, and SVM. The use of PCA led to worse performance in each of the algorithms tested here, perhaps because the data wasn't well suited for it and didn't have a very high degree of dimensionality. The Decision Tree and Gaussian Naive Bayes algorithms were the only two which met the performance threshold for this effort.