Kevin Burnham
Udacity Data Analyst Nanodegree November 2014 cohort
Project 4 Write Up

1. The purpose of this project was to gain experience with the machine learning process from start to finish. Specifically, we use data from the Enron email dataset and related financial information to identify 'persons of interest' - individuals whose pattern of remuneration and email activity suggest that they may have been involved in illegal activities and therefore warrant closer investigation. Machine learning can be useful in accomplishing this by helping us to spot patterns in the data provided to us that are indicative of potentially illegal behavior. Because there are numerous data points for each individual in the database, it would be difficult for a person to see these patterns without the aid of machine learning algorithms.

The data for this project are derived from the 'Enron Corpus', a database of emails from the Enron corporation that became part of the public record during the trials that followed its bankruptcy. Our dataset also includes data on how the Enron employees were paid. The dataset contains 146 observations each one representing an individual associated with Enron prior to its bankruptcy. Of these individuals 18 are identified as a 'person of interest' (POI). This initial determination was made by a prior investigator based on knowledge of the Enron case and intuition about which individuals might be involved in the scandal. The POI label (True of False) is the independent variable in our investigation. The dependent variables in our dataset come from a large corpus of Enron emails and from financial data about how individuals at Enron were paid. The email data include counts of total to and from messages, counts of messages to and from POIs and counts of emails in which the individual and a POI were recipients. The financial data indicates how much money each individual was paid under certain categories - salary, bonus, expenses, loan advances, director fees, deferral payments, and other. It also includes information about payments in the form of company stock. Because the data come from two different sources, many of the variables are not filled for all individuals in the dataset. For example there are 60 individuals with no email data and 21 without financial data. I furthermore found it difficult at times to appropriately conceptualize the dataset since the POI label is itself vague and also because the variables that belonged to each individual could only be created based on prior assumptions about who was a POI. So if for example we changed our minds about whether or not a given individual was a POI we would also have to change several of the email variables for every other person in the dataset. Therefore small changes in our initial assumptions would have very large effects on how we model the data.

Several of the features used in the dataset had data points that were far removed from the bulk of the other points. In two cases, the outliers that we found were removed. We found that as an artifact of the data import process one of our rows of observations was in fact the 'TOTAL' of all observations for the financial variables. This point obviously does not belong since it does not represent a single observation, and so was removed. We also remove the row with the name 'THE TRAVEL AGENCY IN THE PARK' since it too clearly does not belong.

We found data entry errors for two individuals - Sanjay Bhatnagar and Robert Belfer - as both of these individuals had positive entries for the variable restricted_stock_deferred which should always be negative. The document enron6102insiderpay.pdf appears to have the correct values, so we make those corrections. In the case of Belfer we only need to change the sign; however for Bhavnagar several of the entries were incorrect. I believe this was caused by the

absence of a ‘-‘ in the original pdf file so that all of the entries to the right of that were shifted one space. See POI_id.py for details[1].

Although there were numerous other outliers in other variables, I saw no justification for removing them. For example, the name Kenneth Lay shows up as an outlier in a number of financial categories (e.g. ‘salary’, ‘bonus’, ‘other’) because he was the chairman and CEO of the company and as such was well compensated. It is also worth noting that there is no entry for Sherron Watkins, the famed ‘Enron whistle-blower’. It may be that other important data have been removed as well.

2. My final model used the following features:

'salary',
'total_payments',
'exercised_stock_options',
'restricted_stock',
restricted_stock_deferred',
'total_stock_value',
'expenses',
'from_messages',
'other',
‘from_this_person_to_poi',
'deferred_income',
‘from_poi_to_this_person'

These features were selected after initially running the model with all possible features and then removing those with a feature importance of 0. Numerous other combinations of the initial features were also tried, but none gave better results than the combination above.

I also engineered several other features and tested the model with them, but these did not improve the fit and were not included for the final model. The other features that I engineered included ratio_from_POI and ratio_to_POI which indicated what percentage of a person’s to_messages were to a POI and what percentage of a person’s from_messages were from a POI. These features were created with the idea that individuals that are in frequent contact with POIs relative to total contacts are likely themselves to be POIs.

I also engineered features based on financial data. For each of the payment categories ('salary', 'bonus', 'long_term_incentive', 'deferred_income', 'deferral_payments', 'loan_advances', 'other', ‘expenses', ‘director_fees’) a variable was created indicating what percentage of total_payments comes from that category. I felt that individuals that were engaged in fraud might have common patterns of payment. Although these variables did not improve the model, we do see below that the most important features were those relating to pay.

I scaled the data with a MinMaxScaler only for one of the algorithms that I tested - the SVM. My best algorithm, the AdaBoost classifier is based upon decision trees and does not require feature scaling.

---

[1]I just went to post about this and saw that these errors have already been noted

Here are the feature importances of the model:

0.18 expenses
0.16 other
0.1 total_payments
0.1 exercised_stock_options
0.1 deferred_income
0.08 salary
0.08 restricted_stock
0.06 from_this_person_to_POI
0.06 from_POI_to_this_person
0.04 total_stock_value
0.02 restricted_stock_deferred
0.02 from_messages

It appears that variables related to how a given individual was compensated are the most important for the model.

3. My final model was built using the AdaBoost algorithm. This algorithm uses a group of weak decision trees. Each of the trees splits the data based only on a single variable (assuming max_depth=1). At classification, each of these trees 'votes' for its preferred classification and the decision is based on a weighted total of the votes. AdaBoost also weights the importance of classifying a given point correctly during iterations of the algorithm so that the points that early iterations do a poor job classifying are considered more important in later iterations. More details about my final model are found in parts 4, 5 and 6 below.

I was able to get similar accuracy from some other algorithms, but these had either very low recall or very low precision.

I first used GaussianNB() with several different combinations of features, but found that this algorithm tended to have very low accuracy (.23). These models also tended to have low precision due to very many false positives and high recall due to very few false negatives. This model was much too eager to identify someone as a POI.

I then tried an SVM classifier. For this algorithm data were scaled with a MinMaxScaler so that variables with values falling in larger ranges would not be treated as more important. Initially the algorithm simply classified all examples as 0 (i.e. not a person of interest). This gives an accuracy of about .87, but since true positives are zero, precision and recall are also 0(or undefined). Increasing the value of C, which increases the cost of misclassifications, leads to better precision. For example, with C=100 precision is about .5. However, the model continues to have many false negatives so recall remains low (about .1). Compared to my final model the SVM classifier had similar accuracy and precision, but much lower accuracy.

Using a DecisionTreeClassifier() I was able to get accuracy of about .82. There was improvement on recall compared to the SVM to about .26. However, this model, even with several different parameter tunes has low precision, around .2.

4. Tuning an algorithm means experimenting with different values of the algorithm's parameters to find the model that best fits the data. The parameters that can be tuned depend on the algorithm that is being used and affect the specific way that the algorithm tries to fit the data. An improperly tuned algorithm may either underfit or overfit the data. An underfit model fails to learn from the data. It will typically give poor results on both training and testing data. An overfit model in a sense "learns too much" from the training data and fails to ignore noise. As a consequence it does not generalize well to data from outside the training set.

I first experimented with the learning_rate parameter of the AdaBoostClassifier(). This parameter controls the weight assigned to the individual classifiers (but not the weights of the individual observations). By reducing the learning_rate from the default of 1.0 the weight of each successive estimator in the classifier is reduced. I found that a learning_rate of .8 gave the best results on the test set of data.

I also varied the n_estimators parameter which controls the number of decision trees that the AdaBoostClassifier() creates. I found the best results with n_estimators = 30.

It is also possible to vary the parameters of the base estimator used by the AdaBoostClassifier(). The base algorithm used in my AdaBoostClassifier was a decision tree and I experimented with manipulating some of these parameters. Increasing the max_depth parameter beyond the default of 1 always led to worse results on the test data. However we were able to get modest improvement in recall (to .36 from .33) by increasing min_samples_leaf to 2. This means that the decision tree would need to have at least two observations on each leaf of the tree.

5. Validation is when we split the dataset we are holding into a training set and a testing set. We use the training observations to train our algorithm, but when we evaluate the algorithm's performance we see how it does on the test set, that is, data that it has not previously been exposed to. In this way we test the ability of our algorithm to generalize its conclusions to new data. If we do not validate our algorithm we are very likely to overfit our model. That is, we are likely to create an algorithm that is too specific to the data it was trained with and consequently does a poor job classifying observations that it has not previously seen.

Validation can go wrong when the process by which the data are split into training and testing sets is biased. If there is something about a given data point (for example being more recent) that would cause it to end up in the test set then important information could be lost. It is therefore important to randomly select the observations that belong in the test set. Validation can also be a problem because the data in the test set, which may provide valuable information, is not included in training the algorithm. A way to mitigate this problem is to use k-folds cross validation which splits the data into sets and uses each set as the testing set once and uses the rest of the data to train a classifier. We then take the average of the test results generated by the classifiers. This way all of the data is used in training, and all of the data is used in testing as well.

I validated my model using the test_classifier function that was included with the tester.py file. This function takes in a classifier object, a dataset, a list of features to use from the dataset and a k parameter to indicate the number of folds of the dataset to use. This parameter was set at 1000. The function then uses the StratifiedShufleSplit function from sklearn to generate a much larger set of data that it samples from the original dataset and splits into training and testing

halves. It then trains the classifier using the training half and tests the classifier using the testing half.

6. I primarily relied on precision and recall when evaluating the performance of my algorithm. Because the dataset is unbalanced (there are many fewer POIs than non-POIs) accuracy does not give a good measure of the algorithms performance. Simply guessing “not a POI” for all examples gives an accuracy of around .87. This was roughly equivalent to the average accuracy for my model which was .86011.

With an unbalanced dataset precision and recall can be more useful metrics for evaluating a model. The average precision of my final model was .467. Precision is the ratio of true positives (observations correctly identified as POIs) and total positives. This means that just over half (53%) of the individuals identified by the model as POIs were in fact not POIs.

The average recall of my model was .344. Recall is a measure of the ratio between true positives and the total number of trues (true positives + false negatives). This result indicates that only 34% of the individuals that were POIs were identified as such.

Therefore, using the original data with 144 observations and 18 POIs  our model would identify approximately 12 individuals as POIs of which only 6 would actually be POIs. Approximately 12 other individuals that should have been identified as POIs would not be.