Intro to Machine Learning -- Final Project

1. The goal this project is to use publicly available financial and email data from Enron to build a machine learning algorithm that can identify Enron employees that may have been committed fraud. Machine learning is useful for such a task as it allows us to apply a procedure to multi-dimensional datasets in order to 'see' patterns in the data that can used to classify the data into different groups. In this project, we are interested in two groups: those who were involved in fraud (POI) and those who were not (non-POI).

   The dataset used for this project consists of 146 samples with 20 features and 1 target variable for a total of 2920 data points (excluding the target variable). In the target variable, there are 18 POI's. Overall, there are 1323 missing values in the dataset, ranging from about 13%, for the feature with the fewest missing values, up to about 97%, for the feature with the most missing values.

   There is what I would call one definitive outlier in the dataset that was noted in class and refers to the totals for each feature. This I have removed. The remaining outliers (for example, if outliers are defined as being above or below 1.5 times the interquartile range) I have left in as they seem to be valid data points and may potentially contain information that will help identify who was involved in fraud.

2. The final feature set was:

   features_list = ['poi', 'salary', 'to_messages', 'total_payments', 'exercised_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi', 'total_stock_value', 'expenses', 'from_messages', 'other', 'from_this_person_to_poi', 'deferred_income', 'long_term_incentive', 'from_poi_to_this_person', 'from_ratio', 'to_ratio', 'stock_salary', 'bonus_salary', 'bonus_stock']

   The selection process was a mixture of SelectKBest and manual analysis. I first split the data into training and test sets, ran SelectKBest for all values of K, and used the result (K='all') in the final classifier. When run through tester.py, however, both the precision and recall scores dropped below the 0.3 threshold. I then manually adjusted which features to include to get the best scores. For the final model I used scaling, since K Nearest Neighbors uses a distance metric, which means the different features should be around the same magnitude. The original feature scores from SelectKBest were:

   9.45493049, 0.37046178, 7.83452077, 29.30376041,
   11.50541321, 6.8950336 , 5.76523731, 21.60408746,
   0.48146566, 0.58945562, 1.9329649 , 1.36907114,

5.64377539, 6.00007939, 3.05457093, 13.44007592,
1.16835001, 15.17113754, 14.02984879, 14.5368681

I tried mixing different features together and in the end kept the following engineered features:

'from_ratio' = 'from_this_person_to_poi' / 'from_messages'
'stock_salary' = 'exercised_stock_options' * 'salary'
'bonus_salary' = 'bonus' * 'salary'
'bonus_stock' = 'bonus' * 'exercised_stock_options'

The first feature was based on the idea that the absolute number of messages from a person to a poi may not be as strong an indicator as the proportion of messages sent to a poi. The other three features were based on high correlations scores with the 'poi' label found during data exploration (see Enron_data_exploration.ipynb). The idea here was to combine these features to see if I could amplify this correlation.

3. I ended up using the K Nearest Neighbor algorithm, as implemented in sklearn. I also tried the Random Forest, Support Vector Machine and Gaussian Naive Bayes algorithms (all as found in sklearn). Algorithm happened in two phases. In the first phase I was interested in the latter three algorithms. I trained the algorithms on 80% of the data and tested it on the remaining 20%. The results were as follows:

Algorithm	Scaling	Fill NaN	Accuracy	Precision	Recall
RandomForest	No	0	0.79	0.33	0.2
		median	0.83	0.5	0.2
		mean	0.79	0.33	0.2
	Yes	0	0.86	1.0	0.2
		median	0.79	0.3	0.2
		mean	0.83	0.5	0.2
SVM	No	0	0.83	0	0
		median	0.83	0	0
		mean	0.83	0	0
	Yes	0	0.83	0	0
		median	0.83	0	0
		mean	0.83	0	0

Naive Bayes	No	0	0.76	0.33	0.4
		median	0.76	0.33	0.4
		mean	0.79	0.33	0.2
	Yes	0	0.72	0.29	0.4
		median	0.72	0.2	0.2
		mean	0.79	0.4	0.4

Based on these results I decided to use the Random Forest algorithm. After feature selection and parameter tweaking, my tests showed results of:

Accuracy = 0.83
Precision = 0.5
Recall = 0.4

When implemented and tested through tester.py, the results dropped below the 0.3 threshold for recall. Then, based on this resource (http://bit.ly/1Y7cNBh) I tried K Nearest Neighbors, which produced better results.

4. Most algorithms have a set of parameters that can take on several different values. This gives the algorithm flexibility and allows one to tune the algorithm for the particular problem being solved. For example, K Nearest Neighbors uses a distance metric, so for a given problem Manhattan distance may be more appropriate than the usual Euclidean distance (tuned through parameter 'p'). Not tuning the parameters could lead to problems of overfitting or underfitting as the default parameter values may over- or under-emphasize certain features. For my algorithm I used GridSearchCV to tune the parameters. The parameters and ranges tested are:

   n_neighbors = 1, 2, 3, 4, 5, 6, 7, 8
   weights = 'uniform', 'distance'
   leaf_size = 20, 30, 40
   p = 1, 2

5. Validation is the process by which one validates (or not) the model that has been built in order to get a more reliable estimate on how accurate the model will be when it runs on data it has not seen before. The classic mistake made is to use all of the data to train and test the model. This will lead to overfitting and an inflated accuracy. I validated my analysis by splitting the data into a training and test set. In this way, the model was built with 80% of the data and then validated using the remaining 20% that the model had not seen before. In addition to this, the model was run through tester.py which applies cross-validation. The results from both approaches were similar.

6. I used accuracy, precision, and recall as evaluation metrics. Average performance for these was:

    accuracy = 0.89
    precision = 0.64
    recall = 0.33

    Accuracy is the number of correct answers the model produced. In our context this means that 89% of the time the algorithm predicts POI or non-POI when the person was POI or non-POI. Precision here is: of all the people the algorithm classified as POI, what percent were actually POI's. So, for my model, of the people predicted to be POI's the model gets it correct 64% of the time. Recall here is: of all the POI's in the dataset, how many of them can the algorithm identify. So, for my model, of all the POI's in the dataset, the model can identify 33% of them.

Resources

http://scikit-learn.org/stable/
http://stackoverflow.com/
https://www.quora.com/Can-restricted-stock-units-go-negative
http://avc.com/2010/11/employee-equity-restricted-stock-and-rsus/
http://www.eisneramper.com/uploadedFiles/Resource_Center/Articles/Articles/Chapter_7_-_Stock_Options_Restricted_Stock_and_Deferred_Compensation.pdf
http://finance.zacks.com/can-stock-value-negative-9119.html
http://stats.stackexchange.com/questions/57010/is-it-essential-to-do-normalization-for-svm-and-random-forest
http://peekaboo-vision.blogspot.de/2013/01/machine-learning-cheat-sheet-for-scikit.html