Introduction to Machine Learning Project - Identify_Fraud_from_Enron_Email

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 build a model from existing Enron fraud training dataset such that we can predict if an individual is a "Person of Interest" (POI) in the fraud by his/her financial and email data. All those data was collected during federal investigations of the scandal. The whole dataset contained 146 person records. There were totally 21 features, 14 of which were financial features, 6 were email features, and 1 label feature named 'poi'. If 'poi' equals to True, it meant the corresponding person is a POI. The dataset had 18 POI and 128 non-POI.

A total of 2 outliers had been identified in the dataset. The first one was 'TOTAL', which was identified by plotting the scatter plot of salary and bonus features. Clearly It was produced by some preprocessing steps which regarded the row of total counts as a valid individual. Another outlier is 'THE TRAVEL AGENCY IN THE PARK', which was identified manually by looking at its name. The entry had most of its properties equal to either zero or 'NaN' so I regarded it as an outlier. These two entries had been removed from the initial training data.

Some features in the dataset had many missing values. The following table lists features having more than 100 'NaN' values:

Feature	**NaN count**
deferral_payments	106
restricted_stock_deferred	127
loan_advances	141
director_fees	128

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 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 importances 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. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

Firstly, I performed some analysis on the number of 'NaN' values for each feature and the corresponding POI ratio. I noticed that the features 'total_payments', 'total_stock_value', 'expenses' and 'other' had zero number of POI if the corresponding value is 'NaN'. In other words, for a particular individual, if any one of the four features is 'NaN', he must be a non-POI in the dataset. I decided to exclude these features from the feature selection process because they could introduce strong bias (as explained in the course Introduction to Machine Learning - Lesson 5).

Then I applied the univariate feature selection approach and used sklearn SelectKBest to look at the scores of the rest 15 features (excluded email_address and the above four biased features):

Feature	**Score**
exercised_stock_options	24.53272246
bonus	20.52464518
salary	18.00373999
deferred_income	11.32148678
long_term_incentive	9.772103538
restricted_stock	9.079076662
shared_receipt_with_poi	8.432635423
loan_advances	7.125382469
from_poi_to_this_person	5.142219195
from_this_person_to_poi	2.338836115
director_fees	2.14533425
to_messages	1.594256028
deferral_payments	0.232368058
from_messages	0.175383204
restricted_stock_deferred	0.066023245

From this list I see that the most important information were related to those financial features. I decided to create two new features to capture the most important financial and email information. The first new feature was ‘major_payment’, which sum up the top 3 financial features ‘exercised_stock_options’, ‘bonus’ and ‘salary’ resulted from SelectKBest. The reason for this new feature was to capture a more complete compensation package which should include salary, bonus and stock options. The second new feature was the POI message ratio, which equals to (from_poi_to_this_person + from_this_person_to_poi) / (from_messages + to_messages). I created this feature because I thought that an overall POI message ratio should be more meaningful than those email message numbers.

Finally I come up with the following 13 target features with corresponding scores:

Feature	**Score**
major_payment	31.09097964
exercised_stock_options	24.53272246
bonus	20.52464518
salary	18.00373999
deferred_income	11.32148678
long_term_incentive	9.772103538
restricted_stock	9.079076662
shared_receipt_with_poi	8.432635423
loan_advances	7.125382469
poi_message_ratio	5.280597911
director_fees	2.14533425
deferral_payments	0.232368058
restricted_stock_deferred	0.066023245

I had not applied any scaling to the features because I chose AdaBoost and DecisionTree as my training algorithms, which did not seem to be affected by feature scaling.

After some tuning, I ended up using only two features ‘major_payment’ and ‘exercised_stock_options’ in my POI identifier.

3) What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]

At first I tried the Decision Tree supervised learning algorithm. I found this algorithm easy to understand and it performed well without much parameter tuning (precision and recall both easily exceed 0.3 regardless the number of features I chose from the target feature list). Later I tried to use the AdaBoost algorithm for testing, which actually gave a slightly better prediction result than Decision Tree after parameter tuning.

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 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). [relevant rubric item: "tune the algorithm"]

Normally machine learning algorithms need some sort of parameter setting in order to make a better prediction. For example, in using a Decision Tree algorithm we may need to set the minimum number of samples required to split an internal node or set the maximum depth of the tree, in order to give a more accurate prediction, and at the same time, avoid data overfitting.

I had tuned the 'min_samples_split' parameter for the Decision Tree algorithm and 'n_components' parameter for the AdaBoost algorithm, with different number of features selected from the target feature list (with highest score features selected first). I manually set the number of features and used GridSearchCV to find the optimal parameter of the algorithms automatically. The following shows the result for both algorithms with the optimal setting highlighted..

DecisionTreeClassifier				
# Features	**min_samples_split**	**Precision**	**Recall**	**F1-score**
1	7	0.33712	0.31150	0.32380
2	**2**	**0.43846**	**0.51650**	**0.47429**
3	2	0.37544	0.43100	0.40130
4	4	0.34192	0.34500	0.34345
5	7	0.29907	0.27200	0.28489
6	7	0.28564	0.27250	0.27892
7	3	0.31378	0.30750	0.31061

8	7	0.33188	0.26700	0.29593
9	7	0.31528	0.25900	0.28438
10	8	0.32050	0.25800	0.28587
11	2	0.33796	0.31650	0.32688
12	2	0.32751	0.30950	0.31825
13	2	0.33124	0.31650	0.32370

AdaBoostClassifier				
# Features	**n_components**	**Precision**	**Recall**	**F1-score**
1	10	0.60683	0.24000	0.34396
2	**10**	**0.63898**	**0.48850**	**0.55370**
3	10	0.56964	0.36400	0.44417
4	20	0.49398	0.36950	0.42277
5	10	0.46255	0.31800	0.37689
6	10	0.44741	0.31050	0.36659
7	10	0.41661	0.29600	0.34610
8	10	0.42923	0.27750	0.33708
9	10	0.42824	0.27750	0.33677
10	50	0.42138	0.30150	0.35150
11	50	0.42281	0.30400	0.35369
12	30	0.42172	0.31650	0.36161
13	30	0.43077	0.33600	0.37753

The optimal results of this two algorithm are pretty close. DecisionTreeClassifier gave the best performance when the number of features is 2 (‘major_payment’ and ‘exercised_stock_options’) and min_samples_split parameter equals 2. AdaBoostClassifier gave a slightly better result (in terms of f1-score) with the same features and n_components parameter equals to 10.

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 is a necessary step to verify that the learning algorithm performs well on unseen data. A classic mistake for a classification algorithm is **overfitting**, which normally happens when an algorithm takes the same dataset for both training and testing. One common approach to handle overfitting issue is to split the original dataset into different training and test sets, so that we could train and test the algorithm using different data subsets.

I splitted the original dataset into 70% of training and 30% of testing data respectively, and then feed the training dataset to the GridSearchCV() function to find the optimal parameter value. By default, GridSearchCV used a 3-fold cross-validation to evaluate the parameter combinations. The final validation was done by tester.test_classifier method, which in turn used sklearn StratifiedShuffleSplit() function to create 1000 folds of the entire dataset to cross validate the final classifier.

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"]

Precision and recall are important metrics to evaluate machine learning algorithms. The Decision Tree algorithm gave a precision of 43.8% and a recall of 51.7%. AdaBoost algorithm gave a precision of 63.9% and a recall of 48.9%. While Decision Tree had a slightly better recall, AdaBoost gave a far better precision. Therefore finally I considered AdaBoost was a better algorithm when considering the F1-score.

In this project, as the number of POI is much smaller than the total number of persons, accuracy (i.e. (true positive + true negative) / total predictions) is not a very useful metrics because a bad algorithm simply returning non-POI prediction for whatever input will have an accuracy of 87%.

Precision is a metrics measuring 'how many positive predictions are correct over all positive predictions'. In this case, it meant the ratio of correct POI predictions over all POI flagged by the algorithm. Over the total 13000 predictions the AdaBoost algorithm made during the 1000-fold cross validations, it made 1529 POI predictions and 977 of which was correct. Therefore the precision of AdaBoost algorithm is 63.9% (977 / 1529). A high precision implied that it is unlikely for the algorithm to flag a non-POI as POI incorrectly.

Recall is a metrics measuring 'how many correct positive predictions are made out of all actual positives', as known as the sensitivity of the algorithm. In this case, it meant the ratio of correct POI predictions made over all POI records. Over the total 13000 predictions made by AdaBoost, it predicted 977 POI correctly out of 2000 POI records. Therefore the recall of the algorithm is 48.9% (911 / 2000). A high recall implied that the algorithm could identify most of the true POIs.

I found it hard to determine which of the two algorithms performed better as one had a better precision and the other had better recall. Finally I used F1-score, as a balanced score of precision and recall, to decide that AdaBoost was the better algorithm.

References

- http://dshincd.github.io/blog/feautre-scaling/
- http://scikit-learn.org/stable/modules/tree.html
- http://scikit-learn.org/stable/modules/cross_validation.html
- https://en.wikipedia.org/wiki/Precision_and_recall
- (Book) Python Machine Learning - by Sebastian Raschka (Sep 2015)

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.