Enron Submission Free-Response Questions

A critical part of machine learning is making sense of your analysis process, and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your coach evaluates your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: Link to the rubric
Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that the rubric. If your response does not meet expectations, you will be asked to resubmit.

Once you've submitted your responses, your coach will take a look and ask a few more focused follow-up questions on one or more of your answers.

We can't wait to see what you've put together for this project!

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"]

This project investigates the popular Enron dataset which has been made publicly available after the famous Enron fraud scandal that eventually lead to the bankruptcy of the company in 2001. The dataset consists of thousands of emails as well as of a list of names involved in the case and corresponding information. The information falls into two categories: financial information (salary, bonus, etc) and information on the email data (number of emails from persons of interest etc). Also, persons that were identified having committed fraud are labeled as persons of interest ("poi").

The project asks to pick and tune a machine learning algorithm to identify if a person is a poi or not.

There are 146 data points of which 18 are poi. For each person the data holds 20 features (and the label "poi"). We do not include the email address in our analysis. Some features contain a lot of missing values, especially for the persons of interest. The distribution of missing values (thereof number of poi) is as follows:

1 salary: 50 (1)
2 deferral_payments: 106 (13)
3 total_payments: 21 (0)
4 loan_advances: 141 (17)
5 bonus: 63 (2)

6 restricted_stock_deferred: 127 (18)
7 deferred_income: 96 (7)
8 total_stock_value: 20 (0)
9 expenses: 51 (0)
10 exercised_stock_options: 44 (6)
11 other: 52 (0)
12 long_term_incentive: 79 (6)
13 restricted_stock: 35 (1)
14 director_fees: 128 (18)
15 to_messages: 59 (4)
16 from_poi_to_this_person: 59 (4)
17 from_messages: 59 (4)
18 from_this_person_to_poi: 59 (4)
19 shared_receipt_with_poi: 59 (4)

We see that 6 and 14 contain missing values for all pois. No values for pois are missing for total_payments, total_stock_value, expenses and 'other'.

We found at least one outlier, namely "TOTAL", which seems to be a spreadsheet error. As it is clearly not a person, we deleted it. Another value that was deleted was "THE TRAVEL AGENCY IN THE PARK" because it contains a lot of missing values and doesn't sound like an ENRON employee. Other outliers, in particular for the financial information, are often associated with executives of the companies and considering the huge amounts of money involved in the scandal seem plausible. Therefore, no further outliers were removed.

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"]

The list of features that were used for the final analysis were:

'salary',
 'bonus',
 'total_stock_value',
 'expenses',
 'other',
 'restricted_stock',
 'to_messages',
 'from_messages',
 'from_this_person_to_poi',
 'shared_receipt_with_poi',

'to_poi_fraction',
‘restr_to_total’.

Before doing algorithmic feature selection, I discarded some features as they contained too many missing values among the poi (in particular deferral_payments, loan_advances, restricted_stock_deferred and director_fees, see above).

The remaining features were selected using “stability selection”, which applies feature selection on different subsets of the data with different subset of features. It can be used for feature selection by looking at the how many times features were selected during the repeated application of the process. The sklearn randomized lasso algorithm outputs the following scores (how many times a feature was picked in the repeated selection process in %):

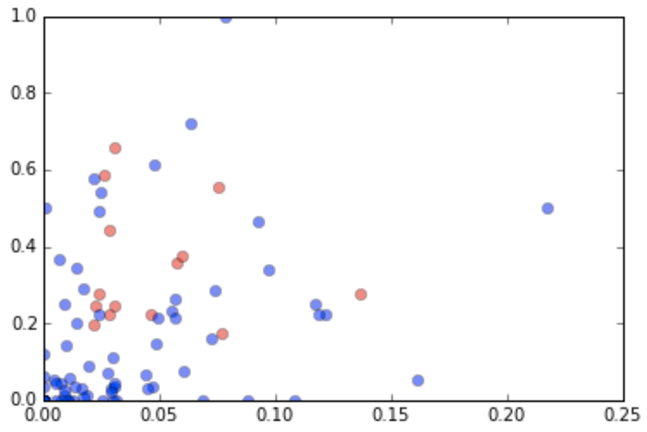
```
salary :  0.425
total_payments :  0.065
bonus :  0.785
total_stock_value :  0.965
expenses :  0.85
other :  0.555
restricted_stock :  0.245
to_messages :  0.29
from_poi_to_this_person :  0.205
from_messages :  0.51
from_this_person_to_poi :  0.61
shared_receipt_with_poi :  0.365
from_poi_fraction :  0.175
to_poi_fraction :  0.99
tot_to_salary :  0.085
tot_to_bonus :  0.245
restr_to_total :  0.82
```

In high-dimensional settings, it is often suggested to use 0.6 as a starting point, i.e. only use features with values above 0.6. In our case, with only a few features, it quickly showed that best performance can be achieved for lower values.

In the end, after evaluating the performance for various cut-off points, we selected all features appearing more than 24% of the time. Three examples of performance metrics for different cut-off values is provided below.

The final features include three features that we created ourselves, i.e. ‘to_poi_fraction’, ‘tot_to_bonus’ and ‘restr_to_total’.

The feature ‘to_poi_fraction' is the fraction of the number of emails from this person to a person of interest (feature ‘from_this_person_to_poi’) over the total number of messages from this person (feature ‘from_messages’). The rationale behind this idea is that the total number of emails to poi alone don’t give enough discriminating power. The fraction might give more information as it relates the number to the total number of emails which can serve as a baseline that allows for better comparison of the feature for different people. Therefore, the new feature serves as some sort of scaling, which is useful as some people may general write more emails than others. (The feature ‘from_poi_fraction’ follows the same reasoning with number of received emails, a scatterplot of both features is shown in the figure).



The second engineered feature is ‘restr_to_total’ which gives the amount of ‘restricted_stock’ divided by the ‘total_stock_value’. Again, this can be useful as scaling can give better comparison. This was also done for ‘total_payments’ and ‘salary’ or ‘bonus’.

The feature importances (ordered as in the list above) using the final decision tree are
0.,
0.26309826,
0.08130909,
0.1129056 ,
0.086381 ,
0.,
0.,
0.,
0.,
0.17668113,
0.27962493,
0.

Literature: This blog article helped my with my feature selection strategy: http://blog.datadive.net/selecting-good-features-part-iv-stability-selection-rfe-and-everything-side-by-side/

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

I ended up using a Decision Tree. I tried a Random Forest first, which gave me best results for n_estimators = 1, which lead me to the idea to use a Decision Tree in the first place. A summary of different parameter settings and performance metrics is given in the following table. The decision tree algorithm outperforms the random forest for the given (and other) parameter combinations.

Stability selection: cut-off	Number of features	Random Forest	Decision Tree	
0.6	6	min_samples_split : 2 n_estimators: 1 max_features: 5	min_samples_split : 2 max_features: 6	**Grid search parameter results**
		Accuracy: 81.2% Precision: 32.9% Recall: 30.2%	Accuracy: 81.6% Precision: 34.4% Recall: 32.0%	**Evaluation metrics**
0.3	10	min_samples_split : 2 n_estimators: 1 max_features: 10	min_samples_split : 2 max_features: 5	**Grid search parameter results**
		Accuracy: 80.9% Precision: 31.7% Recall: 29.2%	Accuracy: 83.7% Precision: 42.1% Recall: 37.4%	**Evaluation metrics**
0.24	11	min_samples_split : 2 n_estimators: 1 max_features: 9	min_samples_split : 10 max_features: 5	**Grid search parameter results**
		Accuracy: 81.0% Precision: 28.8% Recall: 28.8%	Accuracy: 85.4% Precision: 45.3% Recall: 47.6%	**Evaluation metrics**

As decision trees and random forests are invariant towards scaling, no scaling was applied to the features.

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 an algorithm means the process of finding a set a parameters that gives good (or possibly best) performance. Badly tuned algorithms can result in under- or overfitting, i.e. not to achieve satisfying fit of the data or a fit that doesn't generalize to other datasets.

I tuned the algorithm using GridSearchCV, which performs a search over a given set of parameters and picks the best combination (according to a scoring function). As our dataset is small, I performed the grid search multiple times for different train and test sets (using

StratifiedShuffleSplit) and looked for the best parameter combination (on average). For every run, I weighted the parameter combinations with their precision and recall score.

The parameters I found to work best (with the features mentioned above) are

min_samples_split = 10 and
max_features=5.

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"]

As described under 4, I used precision and recall scores the evaluate my results. (Precision is the number of true positives divided by the number of all positives, which means the number of correctly identified poi over the total (possibly incorrect) number of identified poi. Recall gives the percentage of correctly identified poi. (https://uberpython.wordpress.com/2012/01/01/precision-recall-sensitivity-and-specificity/)

Validation is particularly important to assess how well the tuned algorithm performs on other datasets than the training data. This can for example help to identify overfitting. This makes it important to split the data in to train and testing sets. If the evaluation is performed on the same data as the algorithm tuning, the result is prone to overfit and perform worse even on similar datasets.

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"]

The final algorithm (with the specified features) achieves the following performance:
- Accuracy: 0.85347
- Precision: 0.45290
- Recall: 0.47600

Precision states that of all persons that were identified by the algorithm as poi, 45.3% were correct. Recall states that on average 47.6% of all 18 poi are labeled correctly.

Accuracy gives the percentage of correctly classified persons, that is 85.3% of the person in the dataset were correctly classified as either poi or non-poi.