Identify Person of Interest in Enron Dataset

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”]

Answer:
The goal of this project is to identify person of interest (POIs) in Enron dataset. The Enron dataset contains about 145 samples. There are 18 persons in the dataset are labeled as POI while the rest are labeled as non-POI. Each sample has 21 features, including target feature POI. Some features have lots of missing values, which is shown in the following table.

Features	**Number of missing values**
deferral_payments	107
loan_advances	142
restricted_stock_deferred	128
deferred_income	97
director_fees	129

The samples with key ‘TOTAL’ and ‘THE TRAVEL AGENCY IN THE PARK’ are outliers so that I removed these two samples from the dataset.

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”]

Answer:
I didn’t do scaling because I want to use the original data to analyze the performance of the model. For email features I created ‘poi_from_percent' and ‘poi_to_percent', which respectively are the percentage of emails received by a person from a poi and the percentage of emails sent by a person to a poi. I used those features because I think they can best represent the importance and the relationship of an employee with a poi, and I created the two email features instead of using the given ones, because a relative number in this case can be better than the absolute values.
When selecting features, I used SelectKBest and selected eight features with high score. The feature scores are listed as follows:

Features	**Score**
salary	15.86
total_payments	8.96
exercised_stock_options	9.68
bonus	30.73
shared_receipt_with_poi	10.72
total_stock_value	10.63
deferred_income	8.79
poi_from_percent	15.84

In regard to the use of POI to/from emails in engineering new features, there is a possible data leakage between training and test set. When the classifier is indeed used to predict a person's

POI-ness, we obviously do not know beforehand if this person is a POI or not, but in training and test we already know them as shown by existence of “from_this_person_to_poi". Given the limitations of the dataset (low number of data points, low proportion of POIs), I decided to accept this features from a practical standpoint.
Finally, I chose salary, total_payments, exercised_stock_options, bonus, shared_receipt_with_poi, total_stock_value, deferred_income, poi_from_percent.

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”]

Answer:
I ended up using DecisionTreeClassifier. I also tried GaussinNB. The performance of these two algorithms are listed below:

```
======== GaussianNB ========
Accuracy:  0.860465116279
F1; 0.4
Precision: 0.4
Recall: 0.4
======== Decision Tree ========
Accuracy:  0.837209302326
F1; 0.363636363636
Precision: 0.333333333333
Recall: 0.4
```

The difference of the result between these two algorithm is not large. I can do more in adjusting parameters of DecisionTreeClassifier than GaussinNB. So I choose DecisionTreeClassifier to do further optimization.

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?

Answer:
The performance of each algorithm varies greatly based on the selection of the parameters and dataset. As a result, I used StratifiedShuffleSplit to get stratified randomized folds and then tuned the parameters using GridSearchCV. In my final model, I used parameters: 'max_features': 8, 'max_depth': 5, 'min_samples_split': 6, 'criterion': ‘gini’.

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”]

Answer:
Validation is the way to check the performance of the algorithm. I split dataset to train data and test data, using train data to tune the parameters of the algorithm and using test data to measure performance. A classic mistake is to use the same set of train data to train and evaluate the model, which may lead to overoptimistic score to our model. Such model may underperform in the new test data, as we might have overlooked the issue of overfitting while using wrong validation technique. I used StratifiedShuffleSplit to get stratified randomized folds of dataset, separating given data in train and test sets.

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”]

Answer:
I used precision, recall and f1 score to evaluate the performance of my model. Precision is the fraction of people who are actually POI out of all the people predicted as POI. Recall is the fraction of people identified as POI out of all the people who are actually POI.

In poi_id.py, the scores are:
Accuracy: 0.84267
Precision: 0.39286
Recall: 0.33000
F1: 0.35870
F2: 0.34091
Total predictions: 1500
True positives: 66
False positives: 102
False negatives: 134
True negatives: 1198

In tester.py the scores are:
Accuracy: .34091
Precision: 0.37827
Recall: 0.31850
F1: 0.34582
F2: 0.32889
Total predictions: 15000
True positives: 637
False positives: 1047
False negatives: 1363
True negatives: 11953