Huy Do

Enron Submission Free-Response Questions

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 person of interest (POI) identifier using machine learning and the Enron dataset. A POI is a person who either committed fraud and was indicted, settled without admitting guilt, or testified in exchange for immunity. The Enron dataset contains both financial data and email data made public as a result of the investigation.

There are 146 datapoints in the Enron dataset. Each datapoint supposedly represents an employee. Each datapoint has 21 features. The POI feature indicates whether a person is a POI or not. In the dataset, there are 18 POIs and 128 non-POIs. Almost all features have unfilled data, which is denoted as NaN.

When checking for outliers, I’ve removed the following datapoints

- TOTAL: the total line at the end of the table. Somehow this is added as a datapoint
- THE TRAVEL AGENCY IN THE PARK: this is the travel agency that books travel arrangements for Enron employees. There is no significant data except for the **other** feature.
- LOCKHART EUGENE E: this is probably a real Enron employee. However, since all data for this person is missing, there is no use for it

There are other datapoints (e.g. SKILLING JEFFREY K, LAY KENNETH L, etc.…) that are significantly higher financial values such as salary, bonus, etc.… than others. However, since they are legitimate POI and famously known offenders in the case, I’ve decided to keep them in 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 importance 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”]

3 new features are introduced to the dataset:

- emailwithPOI is the percentage of email a person sends to or receives from a POI over the total emails that person sends or receives. The idea is that a person with more communication with a POI is more likely to collude in the scandal
- short_term_interest is the new metric that combine all payments that an Enron employee can receive within a year. These payments tend to incentivize an employee to focus on short-term success of the company to maximize payout
- long_term_interest is the total long-term financial reward that an Enron employee can receive. It forces the employee to think about long-term success of the company as these incentives cannot be received right away

As the number of features for this dataset is small, I initially opted to manually experimenting and choosing the features myself. Using decision tree, an algorithm with built-in feature importance metric, I first tested it on all available features and gradually limited the list of features used to the ones with highest score. The final list of features was:

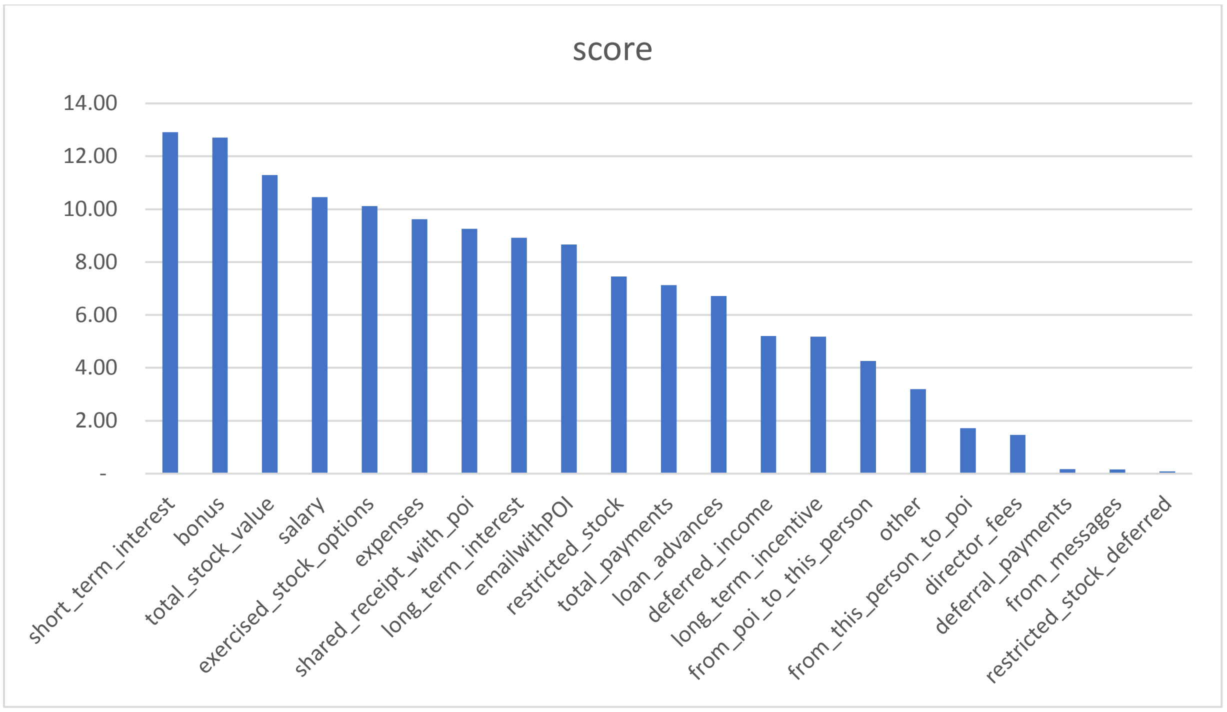
```
    features_list = ['poi','long_term_interest', 'bonus', 'deferred_income', 'expenses',
'emailwithPOI' ]
    Feature importance score:
        0.341525     deferred_income
        0.309113               bonus
        0.278598        emailwithPOI
        0.043134  long_term_interest
        0.027630            expenses
    accuracy is: 0.878048780488
    Precision score is: 0.5
    Recall score is: 0.4
```

However, when this classifier is used on the Udacity provided tester, the recall score falls short of the 0.3 minimum requirement. Tuning the classifier and changing the list of features produce no significant improvement. I also tried random forest but did not achieve better result.

```
        Accuracy: 0.82393      Precision: 0.34406      Recall: 0.25650          F1: 0.29390     F2:
0.27026
        Total predictions: 14000        True positives:  513    False positives:  978   False
negatives: 1487          True negatives: 11022
```

Thus, I switched to unsupervised machine learning and used SelectKBest to choose the best features. I experimented with other machine learning algorithms such as SVC and KNeighbors but they took too long to run with Udacity tester. I finally settled on naive_bayes.

I first run the algorithms with all testing features to examine their scores. Seeing that the feature scores gradually declines, I started at 10 features and experimented with the number of features to achieve the best result. Using a trial and error approach, I found that the best precision and recall scores are achieved with the top 5 features: 'poi', 'bonus', 'exercised_stock_options', 'salary', 'total_stock_value', 'short_term_interest'



Precision score is: 0.6
Recall score is: 0.6

Tester's result

```
GaussianNB()
Accuracy: 0.84764
Precision: 0.45366
Recall: 0.32550
F1: 0.37904     F2: 0.34499

Total predictions: 14000
True positives:  651
False positives:  784
False negatives: 1349
True negatives: 11216
```

Out of the 3 features I created, short-term-interest has the highest score and the biggest effect on the performance of the model. This is reasonable as people with high financial interest on the short-term performance of the company are more likely to take action to inflate current earnings. As in the case of Enron, these people can make the most immediate gains from the falsely reported high earnings via salary, bonus, exercised stock option, etc…

The other 2 features I created (emailwith POI and long_term_interest) did not have a big impact on the model. It is possible that the higher percentage of email exchange with POI (emailwithPOI) only implies normal business relationship and not intention to collude. Lastly, long_term_interst may be a good signal for a person's long-term commitment in the company but is a poor predictor of said person's fraudulent intent.

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"]

I tried DecisionTree, and Random Forest before finally settled on Naive_bayes GaussianNB. With Decision Tree and Random Forest, I can get prevision and recall score to be 0.4 or above. However, they are below 0.3 when being measured by Udacity tester. The difference is probably due to the difference between the dataset used to build my model (final_project_dataset.pkl) and the dataset used in Udacity tester (my_dataset.pkl). With Naive_bayes GaussianNB, I can pass the 0.3 benchmark on both datasets.

The most notable difference between the algorithms I used is speed. While Decision Tree, Random Forest, and Naive_bayes GaussianNB ran relatively fast (training time and testing time are often less than 10 seconds), SVC and Kneighbors take much longer. Kneighbors would take more than 10 minutes and SVC would take more than 30 minutes.

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"]

Tuning the parameters of an algorithm allows users to customize the algorithm to the dataset and features. Incorrect parameters can lead to overfitting or underfitting, neither of which is desirable. To tune the Decision Tree algorithm, I adjusted the min_samples_split. The best min_samples_split I found was 15. Anything lower or higher resulted in lower accuracy.

For the Naive_bayes GaussianNB, there is no parameter to tune.

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 the use of training and testing dataset to test and measure performance of a machine learning model. After the model is trained on the training dataset, it is run on the testing dataset and examined for accuracy and effectiveness. This allow users to minimize over/underfitting and troubleshooting problems with the model.

I used Kfold cross validation to validate the model. Since the dataset is small with few features, using Kfold enables training and testing to be done multiple times on the same dataset; thus improving the accuracy of the model.

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"]

Accuracy, precision, and recall are used to evaluate the model. The final result on testing set was:

```
accuracy is: 0.913
Precision score is: 0.6
Recall score is: 0.6
Confusion Matrix: labels=[1,0]
        [ 3,  2],
        [ 2, 39]
```

Precision measures the likelihood that a predicted POI is actually a POI. In this case, the model predicts a total of 5 POI, 3 of whom are actually POI. Therefore, the prevision score is 0.6. Recalls measures the likelihood that a POI is identified by the model. In this case, there are also 5 POI in the testing dataset. The model correctly identifies 3. Hence, the recall score is also 0.6

When the model is tested with Udacity tester, the result was:

```
GaussianNB()
Accuracy: 0.84764
Precision: 0.45366
Recall: 0.32550
F1: 0.37904    F2: 0.34499

Total predictions: 14000
True positives:  651
False positives:  784
False negatives: 1349
True negatives: 11216
```

The result is slightly different as a different dataset is used. The precision score drops to 0.45. Out of 1435 POIs identified, 651 of whom are true POI. The recall score drops to 0.325. Out of 2000 POIs in the dataset, 651 of whom are correctly identified.

http://scikit-learn.org/stable/tutorial/machine_learning_map/