Yunxiao Wang
Final Project Report

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 purpose of this project is to build a person of interest identifier based on the financial and email data of Enron. The project has already provided us with a hand-generated list of POIs in the Enron fraud case, so our identifier is going to perform supervised learning and make a model to predict if someone is also a POI.

The dataset is a dictionary that contains 146 keys and 21 features in total. Out of the 21 features, "POI" has no missing values as expected. 14 features fall into the range between 20 missing values and 64 missing values. The rest have at least 80 missing values. It's worth noting that although "deferred_income" has high number of missing values but I still used it because it improved the precision and recall when calling tester.py significantly. The first two entries that should be removed from the dataset are "TOTAL" and "THE TRAVEL AGENCY IN THE PARK" as they're not names of a person. I continued to look at each feature I've selected using "SelectKBest", and removed outliers with abnormal values:

- 'BANNANTINE JAMES M' and 'GRAY RODNEY' have salaries that are too low (477 and 6615 versus at least 60000 for everyone else).
- 'BELFER ROBERT' and 'GILLIS JOHN' both have very low exercised_stock_options (3285 and 9803 versus above 17000+ for the rest), 'BELFER ROBERT' is also the only one with negative total_stock_value

So I considered all these 4 entries as outliers and removed them 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"]

I use sklearn.feature_selection.SelectKBest to select the top features. I've found using the top 5 features gives me the best precision and recall when running tester.py. The features I choose and their scores are:

'exercised_stock_options': 24.815079733218194
'total_stock_value': 24.182898678566879
'bonus': 20.792252047181535
'salary': 18.289684043404513
'deferred_income': 11.458476579280369

After choosing these 5 features, I noticed they're all financial features, so I wanted to see if including some email features would improve the results. I first tried "from_this_person_to_POI" and "from_POI_to_this_person", the results were worse than without any email features. Then I replaced the "from_this_person_to_POI" with the ratio of "from_this_person_to_POI" and "from_messages", and did the same thing to the "from_poi_to_this_person" and "to_messages". I made this change because the absolute value of emails involving POI is not necessarily a good indicator of whether this person should be a POI. He/She may have very high number of emails, but almost no emails involve any POI, which probably indicates he/she is not a POI. On the other hand, if the majority of the emails he/she had involve a POI, this person is likely to be a POI as well. So the ratios are likely to be a better indicator whether someone is a POI. As a result, the precision improved by roughly 2% and recall dropped by roughly .2%. I still have the best result without no email features, so I decided not to include email features at the end. The precision and recall are:

	Precision	Recall
No email features	0.49830	0.36650
Absolute	0.41920	0.34500
Ratio	0.44003	0.34300

Because I'm planning to try Naive Bayes and Decision Tree classifiers, both of them treat each feature individually, feature scaling is not needed.

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 picked Gaussian Naïve Bayes as my classifier. I also tested Decision Tree Classifier and tuned the parameters for it. The performances when running tester.py are:

	Accuracy	Precision	Recall
GaussianNB	0.85679	0.49830	0.36650
DTC	0.83793	0.42804	0.40000

Naïve Bayes beat DTC in accuracy and precision, but DTC is slightly ahead in recall. However, I also calculated the metrics on the original dataset used to train the classifier:

	Accuracy	Precision	Recall
GaussianNB	0.86667	0.5	0.38889
DTC	0.97778	1.0	0.83333

It's clear that the scores are very similar between the original dataset and SSSCV for Naïve Bayes, but Decision Tree is showing clear sign of overfitting. So I choose Naïve Bayes as my algorithm.

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"]

Some algorithms have parameters that can be adjusted. The algorithms can perform quite differently based on the values set for these parameters. Good values can help the algorithm avoid high variance or high bias situations that the algorithm may otherwise be prone to. In a high variance situation, the algorithms can be too sensitive to small fluctuations in the training set and result in overfitting, having low recall from cross validation. In a high bias situation, the algorithm can make erroneous assumptions and miss the relations between features and target (underfitting), and give us a low precision score.

The algorithm I decided to use is Naïve Bayes which doesn't have parameters for tuning. But in trying out Decision Tree, I have adjusted "max_depth", "min_sample_split" and "max_features" to give me the best precision and recall from cross validation. Decision Tree is known for being prone to overfitting. "max_depth" defines the maximum depth of the tree, which can help prevent the tree from growing too deep and overfitting. "min_samples_split" does something similar, it is the minimum number of samples required to split an internal node, increasing it can stop the tree from splitting into very small leafs and overfitting the data. "max_features" is the number of features to consider when looking for the best split, which limits the algorithm's ability to make over complicated splits. Although the result still showed signs of overfitting as I mentioned in the previous answers, parameter tuning definitely improved the performances, the scores on the original dataset and from SSSCV are:

	Original dataset			SSSCV		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Before tuning	1.0	1.0	1.0	0.79064	0.25120	0.23500
After tuning	0.97778	1.0	0.83333	0.83793	0.42804	0.40000

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 test process after fitting the dataset with a classifier, it is used to find out if the model is well trained and how well it works on brand new data. In our case, the dataset is small and heavily skewed towards non-POIs, without using a validation technique to estimate the potential performances of the trained classifier on different data, the chance of overfitting and randomly splitting skewed and non-representative sets would be too high.

The way I validated my analysis was to call test_classifier function from the tester.py. This function uses Stratified Shuffle Split Cross Validation(SSSCV) to calculate several metrics, some of these metrics are:

Accuracy: 0.85679

Precision: 0.49830

Recall: 0.36650

The SSSCV randomly picks a portion of the dataset (defaults to 10%) as training set and uses the rest to train the model, the model will then be tested on the

training set. This process is repeated 1000 times (unless a different value is given to the “folds” parameter) to alleviate disadvantages of using a small dataset.

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 accuracy, precision and recall given by SSSCV are:

Accuracy: 0.85679

Precision: 0.49830

Recall: 0.36650

The meaning of these metrics are:

Accuracy: My model gives a correct prediction about whether someone is POI about 85.7% of the time.

Precision: Out of all the predictions my model considers someone as a POI, 49.8% of them are actually POI.

Recall: Out of all the POIs in the testing set, 36.7% of them are identified as POI by my model.