Project 5: Identify Fraud from Enron Email

1. Data Exploration

The goal of this project is to build a model to identify "Person of Interest" (POI) among Enron employees based on their financial and email information. A POI is basically who committed fraud actions in the Enron scandal, and we are very interested in these POIs. Our dataset contains 146 records. There are 14 financial features (e.g. salary, bonus), 6 email features (e.g. number of message sent/received), and 1 target label which shows if a person is a POI or not. Since we expect POIs might have received more money from the company or communicated with each other, we think we may be able use these features to predict the target label of POI.

Before we move to feature selection, I looked at the data to identify outliers that we should get rid of. Two records are removed since they do not stand for real people (the keys are "TOTAL" and "THE TRAVEL AGENCY IN THE PARK"). In addition, I also noticed a person ("GLISAN JR BEN F") with *shared_receipt_with_poi* larger than *to_messages* of him by 1. I considered this as a small data issue and decided to make *shared_receipt_with_poi* equal to *to_messages*.

2. Feature Selection

Based on the features in the dataset, I go ahead and create several new features as listed below:

- ratio of emails sent from poi to this person compared to total emails sent to this person (*ratio_from_poi_to_this_person*): *from_poi_to_this_person* / *to_messages*
- ratio of emails sent to poi from this person compared to total emails sent from this person (*ratio_from_this_person_to_poi*): *from_this_person_to_poi* / *from_messages*
- ratio of emails that this person shared with poi as recipients compared to total emails sent to this person (*ratio_shared_receipt_with_poi*): *shared_receipt_with_poi* / *to_messages*
- ratio of total stock value to total payments (*ratio_stock_to_payments*): *total_stock_value* / *total_payments*

Firstly, I created the first three ratios because they are more meaningful than the absolute number of emails sent to or received from POIs. Secondly, I created the fourth ratio because POIs may have higher stock-to-payment ratio than non-POIs since this ratio is comparable between employees with higher payments and those with lower payments.

With these features in hand, I used random forests to get feature importance. Top 10 features with highest importance are kept for simplicity. These 10 features and their importance values are listed below:

1) *exercised_stock_options*: 0.095450
2) *ratio_from_this_person_to_poi*: 0.095439
3) *bonus*: 0.084748
4) *ratio_shared_receipt_with_poi*: 0.080936

5) *total_stock_value*: 0.072917
6) *ratio_stock_to_payments*: 0.054205
7) *other*: 0.054014
8) *deferred_income*: 0.052035
9) *expenses*: 0.051659
10) *restricted_stock*: 0.049306

These features are scaled by the MinMaxScaler of sklearn. The scaling is necessary since otherwise features of big scale would play more important roles than those of small scale in some algorithms.

3. Algorithm Selection

I have tried several algorithms including Naïve Bayes, Logistic Regression, SVM, and Decision Tree. It turns out that Naïve Bayes could give the satisfying precision and recall score; Logistic Regression performs well in accuracy but poor in recall score; SVM and Decision Tree perform worse than the prior algorithms. Therefore, I chose Naïve Bayes as the algorithm for further tuning.

4. Algorithm Tuning

Tuning helps us to optimize the performance of an algorithm by testing different parameter settings. Since there are no parameters in Naïve Bayes, there is little we can do. The only tuning I did is to try different numbers of PCA components used in Naïve Bayes. Regarding the tuning of algorithm parameters, when I did the algorithm selection, I tried it for some algorithm. For example, I used the GridSearch of sklearn to find the best parameters for SVM kernel and SVM penalty.

5. Model Validation

Model validation is a necessary step to avoid overfitting by testing the model on many train/test sets. If you failed to do it correctly, the model might perform very badly on the test dataset that we didn't use to train the model. To validate the final model, I used the StratifiedShuffleSplit of sklearn. It creates many splits that split data in train/test sets. The model is tested on each split and we could see its average performance.

6. Model Evaluation

To evaluate the model performance, I used two metrics called precision and recall. The value of precision tells us the ratio of correctly predicted POI compared to all predicted POI. The value of recall tells us the ratio of correctly predicted POI compared to all actual POI. The average precision of our model is 0.36 and the average recall of our model is 0.33.

Reference:

1. https://github.com/udacity/ud120-projects
2. Sebastian Raschka. (2015). Python Machine Learning. Birmingham, UK: Packt Publishing Ltd.
3. http://scikit-learn.org/stable/modules/classes.html#

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.