Identify Fraud from Enron Email

Note: From steps below it might seem that each step was performed once and after the previous one. However, there were many repetitions and many steps were revisited after changes were made to algorithm or validation approach.

Answer to 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?**

   This goal of this project is to identify the people involved in the Enron fraud from their financial and communication information. The financial and communication information about Enron employees are provided along with indication whether they are a person of interest (poi). The objective is to utilize this data and develop a classification tool to differentiate between poi people and non-poi people.
   The dataset includes financial information and email communications for 146 samples. One of the samples named 'TOTAL is in fact the sum of all samples and not an actual sample. The sample named 'TOTAL' was removed from dataset as it is likely unintentionally included there.
   In the data set, each sample has 21 reported features. One feature is the label of the sample defining whether the person is POI or not. Also excluding the feature representing the email address of the person, there are 19 features useful for classification. 5 of these features are related to email communications and the other 14 features are financial information.
   It is worth mentioning that there are only 18 poi samples in the data set compared to 127 non-poi samples. This difference in population of different classes requires careful valuation of classification algorithm to prevent bias to one class.
   Additionally, not every person had all the features available. Table below shows the percentage of samples missing specific feature:

Table 1 - percentage of samples missing each feature

	feature	Percent samples missing the feature	Percent poi samples missing the feature
Financial	bonus	44.14	11.11
	deferral_payments	73.79	72.22
	deferred_income	66.90	38.89
	director_fees	88.97	100.00
	email_address	23.45	0.00
	exercised_stock_options	30.34	33.33
	expenses	35.17	0.00
	loan_advances	97.93	94.44
	long_term_incentive	55.17	33.33
	other	36.55	0.00

	restricted_stock	24.83	5.56
	restricted_stock_deferred	88.28	100.00
	salary	35.17	5.56
	total_payments	14.48	0.00
	total_stock_value	13.79	0.00
email	from_messages	40.69	22.22
	from_poi_to_this_person	40.69	22.22
	from_this_person_to_poi	40.69	22.22
	shared_receipt_with_poi	40.69	22.22
	to_messages	40.69	22.22

From the data, it is not directly clear if the unavailable features are missing or not applicable to the person. I decided to move forward with replacing the unavailable features with zero. Assuming that some features are unavailable because the do not apply to that person, it is reasonable to consider zero for them. Some features, such as ‘loan_advances’, are missing for majority of the samples and are not likely to be useful for classification.

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.**

   Since the number of features were fairly limited, I decided to choose them manually. First, I identified the features that were unavailable for most of the ‘poi’s and excluded them from feature list (see table 1 above) . These features are ‘director_fees’, ‘restricted_stock_deferred’, ‘deferral_payments’, and ‘loan_advances’. Excluding these features, we would have 15 features left. However, we have only 18 ‘poi’ samples and ideally the features should be less than that to avoid overfitting. I used the ‘feature_importances_’ values reported by DecisionTreeClassifier (DTC) to remove features with low importance. I used DTC-based and linear classifiers; therefore, there was no need for scaling and normalizing the features.
   Table 2 below show the initial feature_importances observed. At first step, I chose all the features with importance value of higher than or equal to 0.05 (highlited in the table 2 bleow). Removing the features with low importance resulted in relatively better classification performance.

Table 2 - feature importances based on the decision tree classifier

	feature	**feature_importances**
Financial	salary	0.0375
	total_payments	**0.05**
	bonus	**0.0625**

	deferred_income	**0.0875**
	total_stock_value	0.025
	expenses	**0.175**
	exercised_stock_options	0.0375
	other	**0.175**
	long_term_incentive	0
	restricted_stock	**0.125**
email	to_messages	0.0375
	from_poi_to_this_person	0.025
	from_messages	0.025
	from_this_person_to_poi	**0.1375**
	shared_receipt_with_poi	0

After identifying the most important features from DTC's feature_importances values, based on trial and error, I further removed and added features one by one. The goal was to find the minimum number of features that would achieve similar performance as the larger feature set. Given the small number of poi samples, having less features would help with the generalization of the classification.

Most email correspondences features had low importance value and were not included in important features. It seemed to me that a potential good feature would be the ratio of emails communicated with POI relative to total emails. I created three new variables:

1. `to_messages_poi_ratio = from_poi_to_this_person / to_messages`
2. `from_messages_poi_ratio = from_this_person_to_poi / from_messages`
3. `shared_receipt_with_poi_ratio = shared_receipt_with_poi / to_messages`

I included these new features; however, they did not have noticeable positive impact on the classification performance. That being said, they slightly improved the balance between **precision** and **recall** rate. After adding and removing features by trial and error I chose 5 features as listed in table 3 below.

Table 3 - final list of features and their importance values

feature	**feature_importances**
deferred_income	0.1
expenses	0.1
exercised_stock_options	0.2
Other	0.5
from_messages_poi_ratio	0.1

Table below shows the performance of the final classifier with and without the newly implemented feature 'from_messages_poi_ratio'. This feature had the highest feature importance compared to other implemented features. Note that during the validation the classifier parameters were varied to be optimal for each set of features.

The classifier performance with and without the new achieved classification performance (5-fold cross validation) with the final 4 features and the case with the extra 3 new features are shown below.

Original 4 features		Original 4 features + newly implemented feature	
Precision	**Recall**	**Precision**	**Recall**
0.76	0.45	0.67	0.57

3. **What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?**

   I started with DecisionTreeClassifier as it is relatively fast and provide insight into features. However, the performance of the algorithm was not satisfactory. Then, I tried SupportVectorMachine (SVM). The linear SVM would train quickly, but did not result in good performance etiher.
   Finally, I tried the AdaBoost and achieved much better performance compared the above classifiers. Given the performance obtained, I decided to use AdaBoost as the final algorithm of choice.

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).**

   Each algorithm usually has some parameters that need to specified before the training starts. These parameters are fixed during the training and are affected by training process. Therefore, they can impact the performance of algorithm if not chosen properly. I employed the GridSearchCV function in Sklearn to exhaustively search for various combination of parameters and choose the parameter set that result in best performance.
   For the AdaBoost algorithm, I considered the following choices for tunable parameters: n_estimators : [5, 10, 20, 40, 80] and learning_rate : [.25, .5, 1, 2]. Resulting in 20 different combination. The algorithm will be then trained with each combination and the one with highest performance is chosen. The tuned parameters were n_estimators = 10, and learning_rate = 1.
   I also tried to tune the parameters of the underlying DecisionTreeClassifier, but I was not successful in finding any parameters for DecisionTreeClassifier that would result in better performance than AdaBoost with default base_estimator.

5. **What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?**

   Validation is the process of breaking the data samples into training and test, and utilizing the training data for training the machine learning algorithm and using test data to evaluate its

performance. The goal is to ensure the machine learning algorithm can generalize to new and unseen samples and does to blindly memorize the training samples.
Since the number of POI samples were limited, I opted for a kfold cross validation. I employed 5 folds, resulting with 80% training and 20% test samples in each training and testing iteration. To ensure each fold has reasonable number of POI and non-POI samples, I employed the StratifiedKFold function in Sklearn. The average performance of the 5 iteration was then considered as the overall performance.

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.**

   To tune the parameters, I started with 'accuracy' metric. But, given the disparity between number of POI and non-POI samples, accuracy would not paint a good picture of the algorithm performance for POIs. I tried both 'precision' and 'recall' then, but focusing on either would sacrifice the performance of the other one. The recall score represents the probability of catching a POI, while precision score represents the probability of a person being a POI if that individual is flagged by the algorithm as POI. Eventually, I decided to use 'f1' score, to balance between 'precision' and 'recall' while also focusing on performance of POI samples.
   The final performance numbers using the tester.py function is presented below. These performance metrics are for the final algorithm after deciding the features and tuning the parameters.

```
Accuracy: 0.88029    Precision: 0.61360    Recall: 0.43750 F1: 0.51080    F2: 0.46414
```

References

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.

I relied on class material as well as Sklearn and Python documentation to carry on the project. I have also relied on my previous knowledge on machine learning classification gained from my university education.