Enron Person of Interest Classifier

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?

The goal of this project is to build a model that can predict whether a particular employee could have been a person of interest (POI) in the Enron fraud case. The provided dataset has financial and email features, 21 of them, of 146 Enron employees, 18 of whom are designated as POIs. Machine learning provides a way to build a POI detector model by enabling investigation of a labelled dataset that provides details about the financial and email features of several Enron employees along with information on whether they are a POI. Machine learning algorithms also allow extraction of features of interest and construction of a model that ultimately helps in classifying a person as being of interest or not.

From the mini project on outliers and visualization of "salary" vs. "bonus," I knew that a row comprising the totals of each column, named "TOTAL," is present in the dataset. This was the primary outlier that I removed. Further, before starting the actual process of building the model, I looked up the financial information of employees from the enron61702insiderpay document. I observed that certain persons had been highlighted in yellow in the pdf, namely "BAXTER, JOHN C," "DERRICK JR., JAMES V," "FREVERT, MARK A," "PAI, LOU L," and "WHITE JR, THOMAS E." These persons are not POIs, but seemed to have disproportionate assets. Presence of such data might obscure variables that set apart POIs from non-POIs and prove detrimental to the construction of the classifier model. Therefore, I treated the corresponding data points as outliers and eliminated them.

Investigation also revealed that the columns "deferral payments," "loan advances," "director fees," and "restricted stock deferred" were not populated for the majority of POIs (13 or more NaN instances out of 18). The "email address" variable simply had contact information. Hence, I omitted these from the feature list, bringing down the number of variables from 22 to 17.

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. 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.

In order to set a baseline, I first used all the remaining variables and dataset with outliers removed to construct a model using the sample classifier GaussianNB() provided in poi_id.py with no scaling or feature selection.

```
features_list = ['poi', 'salary', 'to_messages', 'total_payments', 'exercised_stock_options', 'bonus',
                 'restricted_stock', 'shared_receipt_with_poi', 'total_stock_value',
                 'expenses', 'from_messages', 'other', 'from_this_person_to_poi',
                 'deferred_income', 'long_term_incentive', 'from_poi_to_this_person']
```

The following result was obtained by running the script tester.py for this model.

```
GaussianNB(priors=None)
        Accuracy: 0.83627       Precision: 0.35839      Recall: 0.28850 F1: 0.31967     F2: 0.30021
        Total predictions: 15000        True positives:  577    False positives: 1033   False negatives: 1423   True negatives: 11967
```

Next, I used the SelectKBest feature selection function to pick out the most significant features; I randomly set the number of features to 8, about half the number of variables. I used a Pipeline to bundle the two steps, feature selection and classification. The selected features and scores are shown below along with the model performance. Both the precision and recall values show an improvement.

```
Features used:
poi 16.07
to_messages 8.71
total_payments 25.85
exercised_stock_options 17.28
bonus 9.21
restricted_stock 9.15
shared_receipt_with_poi 24.83
from_this_person_to_poi 12.02
Pipeline(steps=[('selector', SelectKBest(k=8, score_func=<function f_classif at 0x107D9E30>)), ('classifier', GaussianNB(priors=None))])
        Accuracy: 0.84253       Precision: 0.38544      Recall: 0.30450 F1: 0.34022     F2: 0.31785
        Total predictions: 15000        True positives:  609    False positives:  971   False negatives: 1391   True negatives: 12029
```

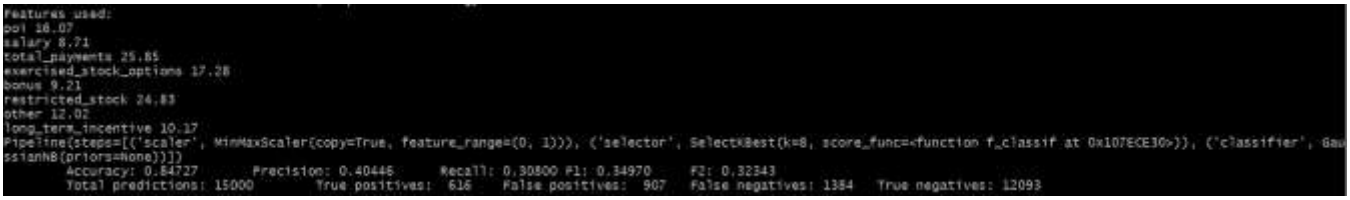
With respect to new features, I added the “fraction_from_poi” and “fraction_to_poi” parameters engineered as part of the coursework. The following results were obtained by adding the new features. The “fraction_from_poi” value was chosen by SelectKBest but the resulting recall value was smaller.

```
Features used:
poi 16.07
total_payments 25.85
exercised_stock_options 17.28
bonus 9.21
restricted_stock 9.15
shared_receipt_with_poi 24.83
from_this_person_to_poi 12.02
fraction_from_poi 17.6
Pipeline(steps=[('selector', SelectKBest(k=8, score_func=<function f_classif at 0x107EEE30>)), ('classifier', GaussianNB(priors=None))])
        Accuracy: 0.84347       Precision: 0.38583      Recall: 0.29400 F1: 0.33371     F2: 0.30869
        Total predictions: 15000        True positives:  588    False positives:  936   False negatives: 1412   True negatives: 12064
```

Seeing how many email features were chosen, “shared_receipt_with_poi,” “from_this_person_to_poi,” and “fraction_from_poi,” I tried combining all of these into a single “comm_with_poi” parameter. This parameter was not found significant enough for inclusion; however, the model showed improved performance based only on financial features.

```
Features used:
poi 16.07
salary 8.71
total_payments 25.85
exercised_stock_options 17.28
bonus 9.21
restricted_stock 24.83
other 12.02
long_term_incentive 10.17
Pipeline(steps=[('selector', SelectKBest(k=8, score_func=<function f_classif at 0x107C4E30>)), ('classifier', GaussianNB(priors=None))])
        Accuracy: 0.84640       Precision: 0.39960      Recall: 0.30250 F1: 0.34434     F2: 0.31795
        Total predictions: 15000        True positives:  605    False positives:  909   False negatives: 1395   True negatives: 12091
```

Further, I felt that scaling could help distinguish small deviations in features by exaggerating their relative positions. This could be helpful for use in the Enron dataset, which is sparse and fundamentally unbalanced between the number of POIs and non-POIs. The MinMaxScaler seemed to be most appropriate for the task. A small improvement in precision and recall values were observed when scaling was included in the Pipeline.



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

In total, I tried three models: the Gaussian Naive Bayes classifier, Decision Tree classifier, and AdaBoost. I built each of these models iteratively as described in the previous answer and fine-tuned the model using GridSearchCV as well as manual adjustments.

Decision Tree classifier

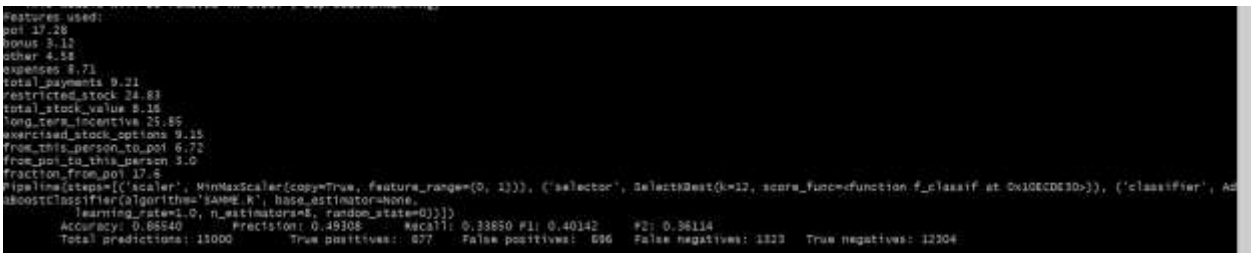
```
features_list = ['poi', 'salary', 'to_messages', 'total_payments', 'exercised_stock_options', 'bonus',
                 'restricted_stock', 'shared_receipt_with_poi', 'total_stock_value',
                 'expenses', 'from_messages', 'other', 'from_this_person_to_poi',
                 'deferred_income', 'long_term_incentive', 'from_poi_to_this_person']
```

The computed "fraction_from_poi" and "fraction_to_poi" were added and the "from_poi_to_this_person" and "from_this_person_to_poi" variables were deleted. The Decision Tree parameters were set as criterion="entropy", splitter = "random", and min_samples_split = 4.

AdaBoost

```
features_list = ['poi', 'bonus', 'other', 'expenses', 'total_payments', 'restricted_stock', 'total_stock_value',
                 'long_term_incentive', 'exercised_stock_options', 'shared_receipt_with_poi',
                 'from_this_person_to_poi', 'from_poi_to_this_person']
```

The computed variables "fraction_from_poi" and "fraction_to_poi" were added to the feature space. This algorithm performed optimally when the value of k was set to 12 in SelectKBest.



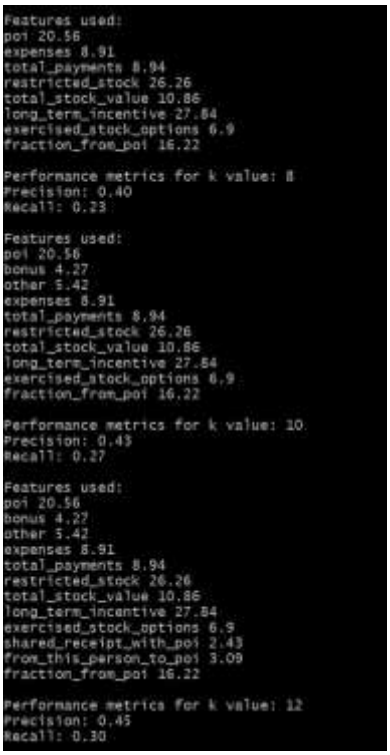
Model Performance Summary

	Gaussian NB	Decision Tree	AdaBoost
Accuracy	0.84680	0.82680	**0.86540**
Precision	0.40053	0.33444	**0.49308**
Recall	0.3	0.30200	**0.33850**

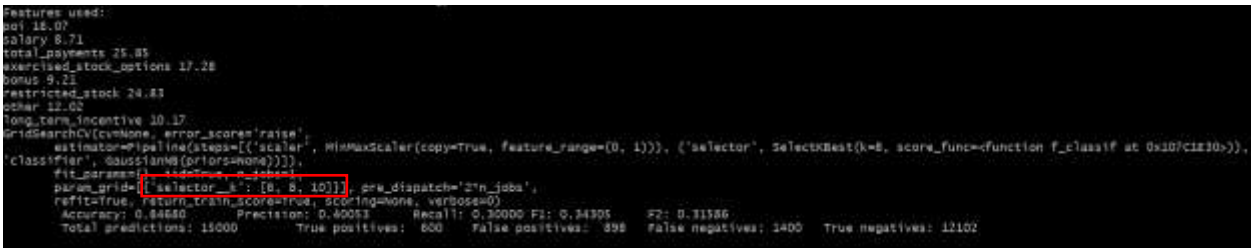
AdaBoost provided the best performance and hence I chose it as my final submission model.

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? What parameters did you tune?

Parameter tuning involves optimization of the hyperparameter values that are passed to the constructor of a model, i.e., the parameters that are not trained during the ML process. These hyperparameters could possibly have a significant impact on the performance of the model and the designer should ensure that the most optimal values are selected for the model to provide the best results, which is evaluated in terms of some score. In the absence of tuning, even if the correct algorithm is used, the model might exhibit subpar performance. For example, the value of k in SelectKBest instructs the model to restrict itself to the topmost k parameters. This could have a significant impact on the precision and recall results of the classifier. Tuning usually involves iterating over a range of values to see which provides the best performance in terms of some metrics. I evaluated the AdaBoost model over three values of k (8, 10, and 12) before setting k to the most optimal value of 12 based on the precision and recall scores. The exact values obtained are shown below.



There are also some automated functions available; for example, for the GaussianNB model, I used GridSearchCV to search for optimal values of k in SelectKBest. This function can also operate over a Pipeline in which case the metric will be optimized over all the steps; however, it is also associated with higher computational time and cost.



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

When a model is very closely fit to a given dataset, it leads to overfitting. In this case, while the model works extremely well on data from the training set, it does not generalize to other cases and might provide unacceptable performance on test sets. This can be avoided by validation, in which the provided dataset is segregated into training and test sets. The model is fit to data from the training set and evaluated using data in the test set. Because of its mention in the sample script, I was motivated to find out what StratifiedShuffleSplit is and how it works. According to the scikit-learn page, "the folds are made by preserving the percentage of samples for each class." This would be really helpful in the case of the unbalanced Enron dataset where there are only 18 POI samples in total. Based on the implemented in the tester script, I used a StratifiedShuffleSplit in my code as well. I used the metrics of precision and recall to check the performance of the model. The implementation of this was coded earlier as part of the evaluation mini project and I reused the same here.

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.

The two main evaluation metrics used in this study are precision and recall. The average values for these metrics got from the models are as follows.

	Gaussian NB	Decision Tree	AdaBoost
Accuracy	0.84680	0.82680	**0.86540**
Precision	0.40053	0.33444	**0.49308**
Recall	0.3	0.30200	**0.33850**

Interpreting these values for the Enron dataset, precision can be described as a parameter that indicates the probability of a person who has been identified by the model as being a POI/non-POI really belonging to that category. For the case of AdaBoost, when a person is identified as a POI, the chances of him being one are approximately half. Recall provides the probability that a person is identified as a POI/non-POI when he really is a POI/non-POI. For the case of AdaBoost, the possibility of a POI being identified as one is 0.33.

References

1. Udacity course materials and code from mini projects

2. Scikit-learn website

3. StackOverflow