Enron POI Project Submission Answers

William McKee **August 2017**

(1) Understanding the Data Set and Question

(a) 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.

The goal of this project is to analyze the publicly available data set for now former Enron employees and identify persons of interest (POIs) in the company's corporate fraud scandal. The data set includes emails sent among Enron employees plus financial data for each employee. There are potentially many features in the email and financial data a program must explore to correctly identify POIs. Combinations of features may need to be explored, making a straight-forward algorithm infeasible. This problem requires the use of machine learning, where a model can be built from sample inputs, known as training data. The model can be used to make predictions about POIs using testing data.

The data set contains a list of former Enron employees (and vendors), and includes the features as shown in the table below. There are 143 employees and 21 features in the data set. Among the 143 employees in the data set, there are only 18 persons of interest (POIs)!

Feature Name	Known value count	Known value count (POIs)	Known value count (non-POIs)
salary	94	17	77
to_messages	86	14	72
deferral_payments	38	5	33
total_payments	123	18	105
loan_advances	3	1	2
bonus	81	16	65
email_address	111	18	93
restricted_stock_deferred	17	0	17
deferred_income	48	11	37
total_stock_value	125	18	107
expenses	94	18	76
from_poi_to_this_person	86	14	72
exercised_stock_options	101	12	89
from_messages	86	14	72
other	91	18	73

from_this_person_to_poi	86	14	72
long_term_incentive	65	12	53
shared_receipt_with_poi	86	14	72
restricted_stock	109	17	92
director_fees	16	0	16

Financial amounts (such as 'salary') are in U.S. dollars annually. Email fields (such as 'to_messages') are total message counts. The 'email_address' itself is a string.

In the data set, there are two features which are the sum of other features. They are `total_payments` and `total_stock_value`.

```
total_payments = salary + bonus + long_term_incentive +
deferred_income + deferral_payments + loan_advances + other +
expenses + director_fees
```

```
total_stock_value = exercised_stock_options + restricted_stock
+ restricted_stock_deferred
```

(b) Were there any outliers in the data when you got it, and how did you handle those?

The 'TOTAL' values for all quantities is included in the data set. The total values include both POIs and non-POIs. These totals should not be analyzed with the rest of the data, so 'TOTAL' is removed from the data set.

There is also an entry for 'THE TRAVEL AGENCY IN THE PARK' which received about 360K in 'other' payments. There was no other information about this entry. This entry will not help us identify POIs so I removed it from the data set.

I checked the pdf file containing the list of Enron employees along with financial data. I removed Eugene E Lockhart since there was no financial data listed for him.

I kept all other data points, including those with extreme values for features such as salary, bonus, or total stock value, since these points will help build the POI identifier. The result is that there are 143 employees in the data set.

All 'NaNs' were replaced with zeroes in the data set. The `total_payments` and `total_stock_value` assume zero for 'NaN' in their totals.

(2) <u>Optimize Feature Selection/Engineering</u>

(a) What features did you end up using in your POI identifier, and what selection process did you use to pick them?

I selected the features by creating a Decision Tree classifier using the default parameters. I invoked sklearn's `model_selection` function `test_train_split` to create training and testing points. I fitted the training data onto the classifier (via the `fit` function), then printed `features_importances_` to see how the model weighed the features. Finally, I ran the tester code to see the model's performance.

I initially tried all nine components of `total_payments` and saw the model give precedence to only five of the features - `salary`, `bonus`, `deferred_income`, `other`, and `expenses`. The results were as follows:

```
features_list = ['salary', 'bonus', 'long_term_incentive',
'deferred_income', 'deferral_payments', 'loan_advances',
'other', 'expenses', 'director_fees']

clf.feature_importances_ = [0.0931  0.2547  0.  0.1079  0.  0.
0.1643  0.3800 0. ]

Precision = 0.30814

Recall = 0.29350
```

I updated the features list to include those five features only. In this case, four of the five were consistently given non-zero weight – `salary`, `bonus`, `other`, and `expenses`! This set of results was as follows:

```
features_list = ['salary', 'bonus', 'deferred_income',
'other', 'expenses']

clf.feature_importances_ = [0.0987  0.3320  0.   0.2304
0.3388]

Precision = 0.32404

Recall = 0.33700
```

I made another update to the features list – this time including only `salary`, `bonus`, `other`, and `expenses`. These four features always had non-zero weights with tester results above 0.32 for both precision and recall!

```
features_list = ['salary', 'bonus', 'other', 'expenses']

clf.feature_importances_ = [0.2044  0.3537   0.1721  0.2698]

Precision = 0.33247
```

```
Recall = 0.32050
```

I performed the same process for the components of '`total_stock_value`'. The decision tree classifier gives weights to two factors - `exercised_stock_options` and `restricted_stock`. The precision and recall are poorer for the Decision Tree classifier compared to the factors of salary, bonus, other, and expenses.

```
features_list = ['exercised_stock_options',
'restricted_stock', 'restricted_stock_deferred']

clf.feature_importances_ = [0.4913  0.5087  0. ]

Precision = 0.25040

Recall = 0.23450
```

When using the features `exercised_stock_options` and `restricted_stock`, the feature importance, precision, and recall metrics are listed below. The results are about the same as the previous case.

```
features_list = ['exercised_stock_options', restricted_stock']

clf.feature_importances_ = [0.4756  0.5244]

Precision = 0.25310

Recall = 0.23500
```

I performed same process for the "email" fields `to_messages`, `from_messages`, `from_poi_to_this_person`, `from_this_person_to_poi`, and `shared_receipt_with_poi`. The first feature is barely a factor while the last four features have near equal importance. The precision, however, was very poor (under 20%).

```
features_list = ['to_messages', 'from_messages',
'from_poi_to_this_person', 'from_this_person_to_poi',
'shared_receipt_with_poi']

clf.feature_importances_ = [0.0174  0.2451  0.2180  0.2609
0.2587]

Precision = 0.19233

Recall = 0.28100
```

I achieved the best results using the features salary, bonus, other, and expenses. I decided to try using just the features salary, bonus, and expenses, avoiding the catch-all category of other and compared model performance.

```
features_list = ['salary', 'bonus', 'expenses']

clf.feature_importances_ = [0.2226  0.1630  0.6144]

Precision = 0.36359

Recall = 0.36850
```

I achieved the best results using these three features, with expenses getting over 60% of the weight. Precision and recall scores are better than 35%, above the project's required 30% threshold for both.

(b) Did you have to do any scaling? Why or why not?

I did not use feature scaling since this has no impact on the Decision Tree classifier.

(c) What feature did you try to make that is not part of the data set? Why?

I created and tested two new features listed below. I thought that higher monetary sums would correlate positively to POIs. I also wanted to see if POIs sent more emails among themselves compared to non-POIs.

- `total_money = total_payments + total_stock_value`
- `total_poi_emails = from_poi_to_this_person + from_this_person_to_poi + shared_receipt_with_poi`

The feature importance, precision, and recall metrics are listed below for these additional features. The precision and recall are very poor! I chose not to use these features in the final model.

```
features_list = ['total_money', 'total_poi_emails']

clf.feature_importances_ = [0.4910  0.5090]

Precision = 0.10381

Recall = 0.08850
```

(3) <u>Algorithm Selection</u>

(a) What algorithm did you end up using?

I chose to use a Decision Tree Classifier on the features `salary`, `bonus`, and `expenses`. Since the POI for the data points are known, a supervised learning algorithm is the best choice.

(b) What other one(s) did you try?

I also tried the Gaussian Naïve-Bayes classifier while performing the feature selection detailed in the answer for question (2a). The Decision Tree consistently performed better than the Naïve-Bayes precision, recall, F1 score, and F2 score. One result is shown in the answer for question (3c).

(c) How did model performance differ between algorithms?

I did the comparison using the default parameters for both algorithms using the features `salary`, `bonus`, and `expenses`. The performance is shown below.

Algorithm	Accuracy	Precision	Recall	F1 Score	F2 Score
Naïve-Bayes	0.80382	0.40771	0.17450	0.24440	0.19704
Decision Tree	0.76727	0.36097	0.36350	0.36223	0.36299

(4) <u>Algorithm Tuning</u>

(a) What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well?

Parameter tuning can be used to improve the accuracy, precision, and recall of an algorithm. For a Decision Tree Classifier, the default value for `min_samples_split` is two, meaning that two data points can still be split into two leaves of one data point each. This behavior can lead to strange boundaries for outliers. Increasing `min_samples_split` may result in a more sensible border which fits testing data better than outliers.

The default value for `max_leaf_nodes` is "None", meaning there could be an unlimited number of leaf nodes in the decision tree. `max_leaf_nodes` can be changed to limit the number of nodes in the decision tree.

Poor parameter tuning can lead to overfitting, when the model works extremely well for the training data but very poorly for any testing data. A model fitted to include outliers in the training data is more likely to result in overfitting.

(b) How did you tune the parameters of your particular algorithm? What parameters did you tune?

For parameter `min_samples_split`, we see precision, recall, F1 score, and F2 score decline for value = 10. The best F1 and F2 scores occur when this parameter is set to three or four.

Value	Accuracy	Precision	Recall	F1 Score	F2 Score
2	0.76727	0.36097	0.36350	0.36223	0.36299
3	0.78227	0.39295	0.36250	0.37711	0.36821
4	0.78464	0.39721	0.35650	0.37576	0.36396
5	0.78336	0.39248	0.34950	0.36974	0.35733
10	0.78945	0.37380	0.23400	0.28782	0.25292

For parameter `max_leaf_nodes`, the precision improves considerably when this parameter is set to ten. The F1 and F2 scores are highest for this line of data. Accuracy also increases.

Value	Accuracy	Precision	Recall	F1 Score	F2 Score
None	0.76727	0.36097	0.36350	0.36223	0.36299
500	0.76755	0.36124	0.36250	0.36187	0.36225
100	0.76982	0.36700	0.36700	0.36700	0.36700
50	0.77018	0.36667	0.36300	0.36482	0.36373
10	0.81373	0.48307	0.34950	0.40557	0.36996
5	0.80891	0.45446	0.25450	0.32628	0.27096

When `max_leaf_nodes` is set to ten, which `min_samples_split` yields the best results? I found that setting `min_samples_split` to four yields the best results. Setting `min_samples_split` to three or five also yields better F1 scores than the previous tests.

Leaf Value	Samples Value	Accuracy	Precision	Recall	F1 Score	F2 Score
10	3	0.81909	0.50367	0.34350	0.40844	0.36883
10	4	0.82118	0.51193	0.35400	0.41856	0.37728
10	5	0.81900	0.50329	0.34450	0.40902	0.36770

The feature selection weights are changed when these parameters are applied to the Decision Tree Classifier. Salary and bonus are given over 20% weight each while expenses weight falls to about 55%.

```
features_list = ['salary', 'bonus', 'expenses']
clf.feature_importances_ = [0.2104  0.2223  0.5672]
```

(5) Validation

(a) What is validation, and what's a classic mistake you can make if you do it wrong?

Validation is the process by which data scientists assess how a model will perform in real world situations. Data scientists must split a data set into training and testing data. The model is fitted on the training data. Predictions are made against testing data to see how the model will perform in practice.

It is possible to make a major mistake in the splitting of training and testing data. The model could be trained on data points with result A. Predictions can be made on data points with result B. The metrics will be very poor. The data points must be sufficiently randomized so that different results are reflected in both training and testing data sets.

(b) How did you validate your analysis?

I performed the validation using the `train_test_split` function from the `sklearn.model_selection` library. This function splits the data set into random training and testing sets. For the Enron data set, this function shuffles the points so that the training and testing sets each contain some POIs.

(6) Evaluation Metrics

(a) Give at least two evaluation metrics and your average performance for each of them.

I performed five runs using the Decision Classifier Model with parameter `min_samples_split = 4` and `max_leaf_nodes = 10`. The results of each run, and the average, is shown in the table below.

Run	Accuracy	Precision	Recall	F1 Score	F2 Score
1	0.82155	0.51349	0.35200	0.41768	0.37563
2	0.82000	0.50717	0.35350	0.41662	0.37630
3	0.82100	0.51119	0.35400	0.41832	0.37720
4	0.81891	0.50288	0.34900	0.41204	0.37175
5	0.82055	0.50939	0.35250	0.41667	0.37564
Average	0.82040	0.50882	0.35220	0.41627	0.37530

(b) Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance.

The metrics cited above have the following meanings:

- Accuracy: Represents the proportion of correct predictions. In this model, 82% of data points are correctly identified as a person of interest (POI) or non-POI. This data is highly skewed so accuracy is not best indicator of model performance.
- Precision: Represents the proportion of positive predictions that are correct. In this model, just over half of all data points predicted as POIs are POIs. Nearly half of the identified POIs are non-POIs (false positive).
- Recall: Represents the proportion of positive data points correctly identified as positive. In this model, only 35% of all POIs are correctly identified as POIs. 65% of POIs are incorrectly identified as non-POI (false negative).
- F1 score: The harmonic mean of precision and recall – an average of the ratios – about 41.6% for this model.
- F2 score: A variant of the F1 score where the recall is given a higher weight than precision - about 37.5% for this model.

Final note: I did not use any web sites, books, forums, blog posts, nor GitHub repositories in the submission of the code.