Enron Submission Free-Response Questions – Doug Waxler, 5/2/21

1. In the Enron data set, there are a number of unknown but possible persons of interest, along with a specific list of known persons of interest form the Enron scandal in the early 2000's. This project took us through a way of automating the identification of other possible persons involved by looking at traits of those who were known POIs, and comparing their attributes and patterns to the rest of the employees in the data set. This allowed an automated way to pick out the signs that a person may have been involved, and identify the likelihood of this in a quantifiable way. Assuming this were ever used in real-world investigations, it would be a great way to prioritize those people you would want to investigate further.

The dataset had a number of Enron employees in it. There were 18 positively identified POIs, and 146 total in the data set. Each of these entries had various pieces of information available. Some had all financial data available, along with counts of emails sent and received, including to and from the POIs identified. There were others in the dataset that had primarily "NaN" values, if not entirely, which makes them difficult subject on which to make these inferences and predictions. There were outliers with extremely high bonuses and salaries, as well as negative stock values and holdings. In my exploration I found bogus data in at least two obvious places – the "Total" was included for all persons, and another entry for a non-person labeled as 'THE TRAVEL AGENCY IN THE PARK'. I removed those entries. I also took a look at the minimums and maximums of each features' values both before and after creating my new features.

2. In the end, I leveraged "features_list_n", which included many of the financial items and email counts. I hoped to leave it to the SelectKBest feature selection to whittle this down, but it seemed that when I did so my performance decreased below my target thresholds. Accuracy dropped nearly in half and I lost about 10% in precision. Feature scaling had similar effects, and I removed it from my final run(s) at the dataset for the SVM algorithm. (Optimize features, part 2 udpate)

I created a feature to look at the number of emails sent to and received from known persons of interest, and did so by creating a percentage. I did this under the assumption that folks with a higher percentage of emails to or from known POIs might be more likely to also be a POI themselves. I added this feature to my "features_list_n" feature set.

*Update 5/3 for re-submission:

Tonight I explored various ways that SelectKBest can be leveraged. Previously I had some difficulty with the function, but I was able to modify some sample code I found and reduce the number of features. Previously I used features_list_n, which included all email & financial features. Today I ran SelectKBest and came up with the following, after attempting a few different values for "K":

```
[('exercised_stock_options', 24.815079733218194),
 ('total_stock_value', 24.18289867856688),
 ('bonus', 20.792252047181535),
 ('salary', 18.289684043404513),
 ('percent_to_poi', 16.40971254803579),
 ('deferred_income', 11.458476579280369),
```

('long_term_incentive', 9.922186013189823),
('restricted_stock', 9.2128106219771),
('total_payments', 8.772777730091676),
('shared_receipt_with_poi', 8.589420731682381),
('loan_advances', 7.184055658288725),
('expenses', 6.094173310638945),
('from_poi_to_this_person', 5.243449713374958),
('other', 4.187477506995375),
('percent_from_poi', 3.128091748156719),
('from_this_person_to_poi', 2.382612108227674),
('director_fees', 2.1263278020077054),
('to_messages', 1.6463411294420076),
('deferral_payments', 0.2246112747360099),
('from_messages', 0.16970094762175533),
('restricted_stock_deferred', 0.06549965290994214)]

I landed on these features, which improved my accuracy and precision by about a percent each:
['poi',
'exercised_stock_options',
'total_stock_value',
'bonus',
'salary',
'percent_to_poi',
'deferred_income',
'long_term_incentive',
'restricted_stock',
'total_payments',
'shared_receipt_with_poi']

3. I created functions that could be modified and called easily to test data, an idea modified from one of my references mentioned. Each function was an algorithm, which allowed me to comment out and test each algorithm efficiently in a single-line function call. I also could re-run my code which created the function definitions easily which allowed me to re-tune the parameters in the algorithm's function. I selected Decision Tree, even though I had super high hopes for adaboost. In researching and learning about these functions I thought that I would get more predictive power from Adaboost due to classifiers learning from and improving upon prior classifiers. In testing the various algorithms, it seemed my SVM took entirely too long to run. Tuning the decision tree yielded much better precision & recall than SVM, Naïve Bayes, or Adaboost. The other algorithms yielded accuracy closer to or below 80%, and precision and recall were either below or barely above the .3 threshold I was given to target.

4. Tuning algorithms is crucial to the performance of a model. A limitation can be set on particular behaviors of the algorithm, such as the maximum depth of a tree, or the maximum number of estimators where boosters in Adaboost are terminated. You can also define the kernel type to use in SVM, which allows you to select a particular separation of groupings that could be more or less linear. Sometimes data groupings aren't easily separable with linear algorithms.

To tune my decision tree algorithm, I leveraged SKLearn documentation and read about the various parameters you can use. It's helpful to understand thoroughly what each of them means so you can better assess the impact they will have for the data set you've explored and understood, given the type and ranges of the features in it. In the end, I chose the "criterion=entropy" as my only real parameter needed. This is used to measure the quality of a split in the decision tree as it runs. I attempted to use 'min_samples_split' and "max_depth" in my wanderings, but wasn't able to get a better performance with regard to accuracy, precision, or recall.

If this were a real-life scenario and someone's judicial fate hangs in the balance, it would be important to be precise, but recall would be equally important. If you miss a number of the positive predictions, even if you got the ones you captured correct, you could miss out on some of the most important POIs you were trying to identify.

5. Validating your findings would be key in order to confirm that what your model thinks to be true, is in fact true, and what it thinks to be false is in fact false. In this case, as mentioned before, the cost could be very impactful on the lives of the persons in this data set. You can perform validation by a proper training / test split on your data. If you train the model on data which is known, and test it on another portion of that same known data, you can find out whether it is correctly identifying positives and negatives, and to what degree these are true. This allows you to make assumptions about the number of times your model will predict correctly, and how often it will capture the positives and negatives respectively. Once this is done, you can go into "the wild" into unknown datasets with a degree of certainty on how well your model will perform in classifying or predicting it's targets. If you do validation incorrectly, you have taught your algorithm poorly to identify its targets, and it will replicate this inaccuracy in production.

6. In this particular data set, where you have employees of Enron which in reality either were or were not involved in the scandal, your model's Precision & Recall are critical performance indicators.

Precision tells you the ratio of times your algorithm predicted correctly that a POI is a POI. Precision is calculated in this way:

$$\frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

Recall tells you the number of times you correctly identified a person of interest, and hints at the number of times you missed them as well. It takes the total number of actual positives (including those you deemed negatives), and creates a ratio against the correct & positively identified targets in your output. It is calculated in this way:

$$\frac{\text{True Positives}}{\text{Trues Positives + False Negatives}}$$

My decision tree algorithm, with the particular features and modified data set, performed as follows:

```
Accuracy: 0.82747
Precision: 0.35686
Recall: 0.36650

F1: 0.36162    F2: 0.36453

Total predictions: 15000
True positives:  733  False positives: 1321
False negatives: 1267 True negatives: 11679
```

The F1 & F2 scores represent a combined precision and recall evaluation known as a harmonic mean. In our case, the F2 score would be more important to the Enron data set as it would be important to correctly identify and not miss the identification of persons of interest (more weight applied to the recall).