Enron Project Report

This dataset is emails from the employees of the infamous Enron Corporation. Enron was known for its accounting fraud known as the Enron scandal. The goal of this project was to see if we could build an algorithm to see if we could detect a “poi” or someone involved in the fraud based on emails and financial data. The data can be used by the algorithm to make predictions. There were two outliers in the data, “Total” and “The Travel Agency in the Park”. Obviously these are not employees at Enron. I initially discovered about them by reading forum posts. I had initially overlooked them and was getting 14% accuracy in my Naive Bayes algorithm with errors in the dataset. After removing those two errors the accuracy jumped to 87%.

There are a total of 146 total data points in the dataset. For each data point there are 21 initial features. We added another feature as seen in the project notebook. There only 18 people with POI status. I manually looked deeper at the features of these people to see if I could spot any patterns. My basic observations were that none of the POIs have a director fee. Most of them have a large bonus. None of them had restricted_stock_deferred. They seem to have a high shared receipt with POI. They also seem to have fairly high total_payments and total_stock_value. I also looked at columns that had missing or NaN values. I observed that 4 features that had over 100 NaN values. These features were deferral_payments, director_fees, restricted_stock_preferred, and loan_advances. Considering our dataset has only 142 data points, these features will not be very helpful to us.

I also created a new feature called unusually_high_bonus. During my manual look at the POI data features, I noticed most POIs had very high bonuses. So I created a new feature that segregated people based on whether they received a bonus above or below 8 million dollars. Testing the features in my GaussianNB classifier without the new feature I got an accuracy score of around 88%. After adding the new feature and running it on the same classifier my accuracy dropped to 84%, so my hypothesis was incorrect. I would assume this is because there were people who received a high bonus that weren’t POI.

For subsetting data I started with a GaussianNB classifier and included all the features. This resulted in a recall of 25% and precision of 40%. I then removed director_fees since it had a lot of NaN values and my precision increased to 28% and recall dropped to 37%. It was okay that recall dropped because it was still above 30%. Next I removed other and the precision increased again to 29%. Recall jumped to 41%. After that I removed salary to see what would happen and my precision dropped back down to 28% and recall also fell to 39%. This was the wrong direction so I added salary back to the features_list. I then took another guess and removed to_messages. This did increase my precision to over 30%. My final feature list is in the notebook and I achieved a precision of around 31% and recall of 42%.

The features I ended up choosing, were simply just picked by trial and error. I literally added all of the columns to the features list and ran the test classifier. As previously mentioned, I achieved a precision of around 25% and a recall of around 40%. I first tried to be logical and take out features that might lead to an inaccurate precision, but that didn’t work because features that I thought would increase precision turned out to actually decrease it. This process was sort of a black box, because I didn’t fully understand what was happening or how it worked. I just started randomly picking features to

remove to improve the precision because my recall was already above 30%. I thought there would be a more systematic way to do this but after reading the Udacity forum post “confused-about-feature-selection-and-outliers” (resources), I realized this was the way to do it.

Tuning a classifier is adjusting the values of the parameters of an algorithm or function. The main goal of tuning a classifier is to optimize and get the best possible performance from your model. I did tune the max_depth parameter for the Gridsearch algorithm. I had initially started out with a range of 1 to 10 which led me to an accuracy score of 79%. I then changed the range from 5 to 10. This only increased my accuracy by 2 percent. Changing the range to 1 to 15 did not increase my accuracy. Changing the max depth to 5 to 15 led me to an accuracy score of 84%. This shows that tuning parameters is important because it directly led to a 5% increase in accuracy on a small dataset. I would imagine tuning parameters would have a even more significant impact on larger datasets. I did not have to tune the parameters for the GaussianNB algorithm for this project. Using the GausianNB algorithm was very straight forward and easy, I just entered the training features and training labels, which were already given in the starter code in the fit function and was able to get around 80% accuracy. After going through the Deep Learning Nano Degree I’m very familiar with the importance of tuning hyper parameters. In that Nano Degree I would spend around 20-30% of my time just tuning the hyper parameters, which included Epochs, Keep Probability, Batch Sizes, Convolutional Layers and Pooling Layers. By not tuning hyper parameters the accuracy of the model will be low.

Validation is making sure your model can generalize well to the test data. A classic mistake is to use testing data for training. This is a huge error and will ensure that your model can’t be generalized to other datasets. This is commonly referred to as overfitting.

For this project I did not have to use feature scaling. Both the Naïve Bayes and Decision Tree classifiers work without feature scaling.

The evaluation metrics used in this project were accuracy, precision and recall. These already came with the starter code in the tester.py file. As already mentioned the accuracy I got when running the GausianNB method was around 80%. After running the test_classifier function on my features list I got a recall of about 30% and a precision of 42%. From what I understand Precision is the proportion of true positives to true positives and false positives. Recall is the proportion of true positives to true positives and false negatives. In other words, precision measured how correctly the algorithm predicted someone was a POI, when they were actually a POI. From our algorithm, this was done only 30% of the time. Recall measured of the total number of POIs how many did our algorithm mark as POI. This was less than half for both our algorithms.

This algorithm was validated using stratified shuffle split with 1000 folds in the tester.py file. Stratified Shuffle split is a merge between StratifiedKFold and Shuffle Split. We choose stratified Shuffle split since this is a small dataset and we want to make our validation models robust. It also keeps our test set consistent and robust by having the same ratio of POIs and Non-POIs. This helps in ensuring the accuracy and proper training of our model. Comparing the two classifiers I found the Naïve Bayes classifier had a significantly larger recall score and only a slightly less precision score. Therefore it would be better to use the Naïve Bayes classifier on this dataset.