Matthew Bellissimo

**Enron Submission Free-Response 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?  [relevant rubric items: “data exploration”, “outlier investigation”]*

The goal of this project was to apply machine learning techniques to the Enron Dataset, in order to create a classifier that could predict whether, based off a set of features, an employee was a person of interest (POI) or not. The dataset in question contained 146 data points and 26 features. 18 out of the 146 data points were POI’s, and 1 point was an extreme outlier that I ended up removing, due to the fact that it was a spreadsheet quirk and it’s inclusion was an error. While there were some additional outliers in the dataset set (for instance, employees with bonuses over 5 million dollars, and a salary of over 1 million dollars), they represented valid data points, and actually included some POI’s, so they were left in.

1. *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.  [relevant rubric items: “create new features”, “properly scale features”, “intelligently select feature”]*

The process I used for feature selection used equal parts discretion and statistical logic. First, I excluded “email\_address”, which was a non-numeric feature, and “other”, which was poorly defined. Next, I used a correlation matrix to see which features were strongly correlated (>.85) with other features and got rid of them as well. I created two additional features, “fraction\_from\_poi”, and “fraction\_to\_poi”, by combining "from\_poi\_to\_this\_person","to\_messages", and "from\_this\_person\_to\_poi","from\_messages" respectively. I did this because I thought that looking at the proportion of messages to and from POIs would do a better job of providing context towards an association with guilty/suspected parties, than just looking at the absolute number alone, which I believed could be misleading. I removed then removed the component features in order to reduce redundancy. Finally I looked at which features had a high number of missing values. If these features were not at least moderately correlated with POI labeled employees, I chose to exclude them as well.

Once I excluded all the features I knew I would not use, the next step was to scale them. I used MinMaxScaler for this process. Scaling was important here because the remaining features I was looking at all had different units. For instance, salary was being measured in dollars, and had a highly disparate range, while fraction\_to\_poi was a fraction, and thus by definition could not exceed a value of one. Scaling made it so that when I fit my classifier, I would be doing a better job of looking at features as apples to apples. The last step I performed for feature selection was to use the function SelectKBest, with a k value set to four, so that I would only be looking at the four most important features. The final features that were selected and put into my classifier based off these steps were:

|  |  |  |
| --- | --- | --- |
| **Feature Name(s)** | **KBest Feature Score(s)** | **DT Feature Importance(s)** |
| fraction\_from\_poi: | 2.337903 | 0.0985505 |
| loan\_advances: | 2.180175 | 0.00460426 |
| shared\_receipt\_with\_poi: | 7.167587 | 0.3850618 |
| fraction\_to\_poi: | 14.400092 | 0.51178343 |

I was exceedingly happy to notice that my two created features were included in final classifier, as it proved my hunch for their relevance to be correct. As a final note, I used k=4 as the parameter for select k beset as it was the one that provided my the best success metrics after a series of trial and error.

1. *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?  [relevant rubric item: “pick an algorithm”]*

I ended up using a Decision tree for my classifier. This was after trying a number of other algorithms and getting unsatisfactory results. For instance, the first algorithm I tried was a Naïve Bayes, and despite playing around with feature selection a bit, I couldn’t get any results over .3, I also that the nature of the algorithm was inappropriate for this type of problem since we were dealing with a lot of continuous data. The next algorithm I tried was an SVM, but that also turned out to be a bad solution as was prone to over-fitting, and thus it was not well suited to generalizing rules which could be applied to new previously unseen data. Decision tree’s ended up working really well, since it was flexible enough to be tuned appropriately, and did a much better job than SVM at predicting new data. I ended up getting over .54 scores on all of my key metrics(precision,recall,f1).

1. *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).  [relevant rubric item: “tune the algorithm”]*

Tuning the parameters of an algorithm basically means specifying how, and under what constraints you want your classifier fitted, with the goal of achieving the best results with your dataset. For instance, one of the parameters, on the decision tree that was actually really helpful was class\_weight. This parameter basically allowed me to specify how much extra weight I want to assign towards getting certain class values correct. So for instance, in the Enron dataset, we have only 18 out of 145 data points that were POIs. Since we really only really cared about finding out which data points were indicative of guilt, adding extra weight to this class made a lot of sense, and was able to boost my success metrics tremendously. Other parameters that I included I in my tuning were min-sample-split, which specified the minimum number of samples required to split an internal node, and criterion, which specified whether splits should focus on reducing impurity or gaining information. Because there were so many different combinations of features I could use, I ended up tuning my model with GridSearchCV, which is a function that essentially auto tunes my classifier with all of the possible parameters in mind, in order to get the best result. By default, GridSearch uses kfold cross validation, but, again because of the skewed dataset, I specified that it should use a form a Stratified Shuffle split instead. I also told it to focus on optimizing f1 using the micro calculation (Metrics counted globally by counting the total true positives, false negatives and false positives).

1. *What is validation, and what’s a classic mistake you can make if you do it wrong? How did you validate your analysis?  [relevant rubric item: “validation strategy”]*

In machine learning, training data is the data you train your classifier with, while testing data is data that you’ve set aside so that you can test the accuracy of your trained classifier. While ideally you would want to maximize the size of both sets of data, in practice, this can be really difficult sometimes. For instance, in the case of the Enron data set, because we have so few data points, separating ANY data is very hard as you are risking either improperly training the classifier, or testing it on an insufficiently sized sample. With cross validation, you get around this problem, by partitioning your data in k sets and running your classifier multiple times, each time rotating which of your partitions is the testing set, and average the result. While this can take up more computational power for training and test, the benefit generally is more accurate results. There are also ways where you can validate your data the wrong way. For example, the Enron data set is very imbalanced. This can be a problem if the partitions you are randomly creating don’t contain enough (or any) of the less frequent class, as it makes it more difficult for the classifier to make/learn correct associations. Another problem is if the data has been previously ordered in some way, so that the partitions are abnormally dissimilar to each other (Imagine 10 partitions, 5 made up of all POI data points, and 5 made up of all not poi data points). With these issue in mind, I chose to use a method known as Stratified Shuffle split when doing my cross validation. This means that when I create my partitions the data is shuffled beforehand, with the caveat being that each partition has to maintain the same class distribution as the initial dataset. A final note was that when I was using GridSearch to tuning my classfier, I made sure the seed I used for stratified shuffle split was different from my final tester.py stratified shuffle split so that randomness would be mainted.

1. *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. [relevant rubric item: “usage of evaluation metrics”]*

Two key metrics that I used as a measure of success were precision and recall, which are defined below.

Precision: TP/(TP+FP)

Recall: TP/(TP+FN)

In layman’s terms Precision can be defined as the average probability of a relevant call being accurate, in our case how many predicted poi labels were accurate, whereas represents how the fraction of actual poi’s out the total poi’s we’ve identified correctly.

I was able to achieve average scores of 0.54512, and 0.60100 for precision and recall respectively. Given that the metric for success in this project was .3 and the level of imbalance in the dataset, I judged these scores as representing a pretty decent success with the classifier.