Kasey Cox

December 2017

Enron Submission Free-Response Questions

A critical part of machine learning is making sense of your analysis process and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be about 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your evaluator looks at your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: [[Link](https://www.google.com/url?q=https://review.udacity.com/%23!/projects/3174288624/rubric&sa=D&ust=1512323512213000&usg=AFQjCNF6hXrO1wfgDdwruS-ClP44mGDClg)] Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that part of the rubric. If your response does not meet expectations for all rubric points, you will be asked to revise and resubmit your project. Make sure that your responses are detailed enough that the evaluator will be able to understand the steps you took and your thought processes as you went through the data analysis.

Once you’ve submitted your responses, your coach will take a look and may ask a few more focused follow-up questions on one or more of your answers.

We can’t wait to see what you’ve put together for this project!

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 create and deploy a machine learning algorithm that could help us identify persons of interest (“POI”) in the Enron scandal from publicly released employee email and financial data. The dataset used to develop the algorithm contains 146 data points (each representing an employee) and 21 features. Of the data points, 18 belong to POI and 128 belong to non-POI. Every feature, except for the POI/non-POI feature, has missing values. The median number of missing values is 60. This dataset can be divided into training and testing data to develop an appropriate ML algorithm.

Some outliers or strange data points had to be eliminated before proceeding, however. One major outlier explored during the class lectures was a data point called “TOTAL”. It is a byproduct of this data being derived from a spreadsheet. I eliminated it because it is not a valid data point and will not help me identify whether or not a new point belongs to a POI or non-POI. I discovered two other useless points while finding the number of missing values in the data. One data point, “LOCKHART EUGENE E”, had only missing values other than the POI identifier feature. I eliminated that since it contained no useful information. The second point, “THE TRAVEL AGENCY IN THE PARK”, was missing 18 values. I eliminated it because it does not belong to a person and therefore cannot help me determine if an employee is a POI or not.

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”, “intelligently select features”, “properly scale features”]**

Ultimately, I used `SelectKBest` from sci-kit learn to pick the 5 best features. Before getting to that step in selection I whittled down some of the features by other means. I removed "email\_address" from my features list because a person's email address is a unique string based on their name that has no bearing on POI status. I then checked how many missing values were present in each feature. Those with many missing values (almost 100 or more) were omitted because they don't provide enough examples to give trust-worthy generalizations. Finally I created two variables that I thought may better gauge email activity: “fraction\_from\_poi” and “fraction\_to\_poi”. I thought that perhaps an employee's email activity with POIs could reflect whether or not they themselves are POIs. “fraction\_from\_poi” was created by dividing “from\_poi\_to\_this\_person” by “to\_messages”. “fraction\_to\_poi” was created by dividing “from\_this\_person\_to\_poi” by “from\_messages”.

After removing undesirable features and creating the two new ones, I normalized my features. This was important because some of the financial features went into the millions while many of the email features were single to double digit numbers – something like salary would then completely dwarf email data. I then used `SelectKBest` on my feature list to pick the 5 best features. Initially I had it select 10 features while using a Gaussian Naive Bayes classifier, but the performance of the algorithm was not sufficient (precision: 0.33357, recall: 0.23100). When I had visually inspected different features by plotting earlier on, I noticed that there are not any especially distinct patterns (at least to the eye). I reasoned that perhaps this makes classification a bit messy and that it would be best to simplify matters. When I reduced K to 5, I saw improved recall and precision for the Gaussian Naive Bayes classifier (precision: 0.47400, recall: 35100). The top five features, in terms of predictive power, were 'salary', 'bonus', 'total\_stock\_value', 'exercised\_stock\_options', and 'fraction\_to\_poi' with the respective scores of 24.82, 24.18, 20.79, 18.29, 16.41 (ANOVA F-values). I stuck with the default scoring function because something like `chi2` doesn't make sense with numerical values.

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 Gaussian Naive Bayes classifier. I also tried Decision Tree classifier with different values for `min\_samples\_split` as recommended by my parameter tuning with `GridSearchCV`. With the 5 best features (as described in the previous answer) and `min\_samples\_split` equal to 2, precision was 0.27428 and recall was 0.27250. Setting `min\_samples\_split` to 4 yield a precision of 0.266601 and recall of 0.25750; `min\_samples\_split` set to 5 gave a precision of 0.27428 and recall of 27250. These were worse than my Gaussian Naive Bayes algorithm.

I also tried a a Support Vector Machine classifier. Out of the box it tended to have better accuracy scores, but its precision and recall were not better. Trying to tune it with `GridSearchCV` was painfully slow so I opted not to continue with an SVM.

Before passing my algorithm and features to `tester.py`, I evaluated my algorithm and features simply with `train\_test\_split` and retrieved the score, precision, and recall.

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? What parameters did you tune? (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 items: “discuss parameter tuning”, “tune the algorithm”]**

Tuning parameters of an algorithm refers to the optimization of hyper-parameters with respect to the algorithm's performance. Algorithms present in sci-kit learn have hyper-parameters that you can set when creating your classifier. Typically they are unique to that algorithm and they frequently can improve the performance of the algorithm in question. If a model goes through too much tuning, the model will be get quite good scores on your training data, but will not generalize well to new data; the model will be overfitted. On the other hand, if it is not tuned much or at all, your algorithm may be too general and miss underlying patterns in the data.

As noted above, the Decision Tree algorithm has a hyper-parameter called `min\_samples\_split` which dictates the minimum number of samples required to further split an internal node. This is useful to tune because it can help protect against overfitting. When this hyper-parameter using `GridSearchCV` and gave it a parameter space of (2, 3, 4, 5, 6) for `min\_samples\_split`. 2 and 5 ended up giving the best results for a Decision Tree. When I was trying out an SVM algorithm, I also attempted to tune it with `GridSearchCV` since it has parameters that can be tuned like `gamma`, `C`, and `kernel`. For reasons stated in the previous answer, I opted to not use an SVM. The algorithm that I ended up using, Gaussian Naive Bayes, did not have any tunable parameters.

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 items: “discuss validation”, “validation strategy”]**

Validation is the process by which a trained model is evaluated with a testing data set that was not part of the model's training. It's vital for ensuring that your data isn't overfitted: when the algorithm memorizes the pattern you've given it and doesn't generalize well. A mistake that you can make is not withholding your test data from your training data. You want your testing data to be novel to the trained algorithm.

One popular way to do validation is with K-folds cross validation. This is a form of validation that attempts to maximize the learning and validation results from a given amount of data. This is done by splitting the data into different training and validation sets K times and taking the average metric results from those trails. At first I used this method, but then realized that it wasn't matching up with `tester.py`, which deploys Stratified ShuffleSplit cross-validation. K-folds is not appropriate for this data set because there are many more non-POI points than POI points. K-folds does not evenly distribute these labels during its trials, leading to unreliable results. Stratified ShuffleSplit cross-validation is the right way to go because it ensures this even distribution during each round of validation. Because this is deployed in `tester.py`, I did a simple “preview” validation (using `train\_test\_split`) in `poi\_id.py`, along with precision and recall metrics, before checking it with `tester.py`.

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”]**

I used three metrics to evaluate my algorithm's performance: accuracy score, precision, and recall. Accuracy score is the most intuitive of the bunch. It is the number of items labeled correctly out of all items labeled. While this makes sense, it is not good when your data inherently has a lot more of one label than any of the other labels. This is the case in this Enron scandal project. The data has very few POIs and many non-POIs. If the algorithm were to predict non-POI randomly, chances are that it would be correct because most data points are non-POI regardless of features. My algorithm's accuracy is 0.86153.

Because that is not sufficient, I also used precision and recall to evaluate my algorithm. Recall, sometimes called sensitivity or true positive rate, measures the number of relevant samples (say, POI, for example) correctly labeled out of all possible relevant samples (POIs). A low recall, therefore, misses a lot of POIs. My recall, according to the `tester.py` script, is 0.35100. Precision, also called positive predictive value, measures the number of correctly labeled items out of all items with that label (correctly or not). A low precision then is when many false positives exist. My algorithm's precision is 0.47400.