Alex Berry, Aumit Leon, Abdi Mohamed

CS451 Homework 4 Report

Over the course of implementing this spam classifier, we implemented a number of ideas to improve the performance of the SVMlight classification methods. This included identifying misclassified examples, extracting the 10,000 most frequently used words, and trying different C values. In addition to extracting more informative features and tuning our C value, we also took steps to preprocess our data: this included removing words with lengths less than 2 characters, removing punctuation and most special symbols, removing "stop words" (words that carry little value in actual conversation and make for poor features), and applying stemming at the feature extraction step as well as the SVM file creation step. To speed up testing different models, we wrote `automate.py` to run `svm_learn/svm_classify` with varying values for C. Each strategy had varying levels of success but we are happy to say that our final model has a consistent accuracy of 98.62% and precision/recall values of 97.68% and 99.17%, respectively.

To begin, we attempted to examine the examples that our original model misclassified. In order to find these examples, we needed to create a way of locating them in the datasets, as the report generated by running `svm_learn/svm_classify` only returns the number of misclassifications. We wrote `find_misclass.py` which goes through the labels for the validation set and saves the paths of the misclassified examples to a file called `misclass.txt`. This enabled us to manually examine emails for any common patterns which we could then add to our feature vector.

The next strategy we tried was increasing the size and quality of the feature vector. This meant formulating a way to extract the most frequently used words in the email data set and to rank them by usage. We added a function to `spamsvm.py` called `featExtractor()` which uses a dictionary to aggregate words and frequencies. After several attempts, we decided that adding the top 10,000 most frequently used words to the feature vector gave us the best increase in accuracy, improving it from the default 81.44% to 97.50% and precision/recall from 82.54%/74.64% to 96.84%/97.65%. With more tuning, we got this accuracy to 98.62% with a 80-20 split in the 8000 training examples.

Modifying C was the biggest step-- we simply kept retraining the model and reclassifying using different values of C, until we found a value that gave us some sort of upper bound. The addition of pre-processing steps such as identifying stop words, removing words of length less than 2, and stemming had arguably biggest impact in performance after solid feature extraction and identification of C.

A large part of this project was the review of literature, here are just 2 of the papers that we consulted: http://www.ijste.org/articles/IJSTEV1I11008.pdf, https://link.springer.com/content/pdf/10.1007/978-1-4020-6264-3.pdf#page=388