CS294-1 Programming Assignment 1

Richard Hwang, David Huang

February 13, 2013

1 Our Solution

We followed advice in the assignment statement, splitting the text with the regex "[\\s.,();:!?&\"]+" and indexing each word except for certain categories of stopwords. After experimenting with filtering out various categories of stopwords, we discovered that filtering out 1) articles (e.g. "the", "a", "an"), 2) auxiliary verbs (e.g. "be", "does") and 3) non-directional prepositions (e.g. "about", "during", "of") yield the best F-measure later on.

We then constructed an array of two sparse matrices for the review documents (one for positive reviews and the other for negative reviews). In particular, we have the indexed terms as rows and the documents as columns, resulting in two $|W| * |D|$ matrices where $|W|$ denotes the size of our dictionary and $|D|$ denotes the number of positive/negative reviews (1000 each in this case).

Next we fed the aforementioned sparse matrices into our $train()$ function wherein we produced two $|W| * 1$ matrices of log-probabilities (again one each for positive and negative reviews) to serve as our model for classification.

We now have a model with which we could apply our $classify()$ function to a given document (preprocessed into a sparse matrix representation). Using a multinomial model where we treat each occurrence of the same word in a document as independent terms with the same probability, we compute maximum a-posteriori (MAP) estimates for the given document being positive or negative. After comparison, we output the more likely outcome as an integer (1 for positive, -1 for negative).

Finally, $validate()$ performs n-fold (in this case n = 10) cross validation on our model. We compute precision and recall rates for each fold and compile them into an overall average f-measure for the aforementioned model across 10 folds. $validate()$ then outputs this f-measure.

2 Smoothing/Backoff

Since for zero-count terms in our dictionary, taking the logarithm of 0 would result in floating-point arithmetic error, we decided to apply additive smoothing with $\alpha = 1$.

3 Performance

For some reason, indexing the words was intolerably slow. We tried using foreach, for loops, String.split() and StringTokenizer, and for loops and split turned out to be the fastest combination. It still took at least 10 minutes to index all the words, though. Constructing the word-document matrix, however, was very fast: about 7 seconds. Building the model, that is, computing all the log probabilities took 3.4 seconds. Finally, validation took 13.6 seconds. Placing a flip and flop around our entire main method, we recorded 0.021549 GFlops on a Macbook Pro with Intel Core Duo.