CS165A MP2

Steven Cabral
6801179
CS165A
M 4PM Section
03/09/15

For this project, the assignment was to create an AI model for determining whether a set of text (in this case a movie review) expresses a positive sentiment or a negative one. Specifically, the specification is to use decision trees to construct the evaluative model and pruning the tree to avoid overfitting. I chose term frequency to be the primary measure of information gain on which the decision trees were split.

My code revolves around using the Document class, which contains the label, words, and word frequencies for each document read in from the files. After being constructed, these Documents are fed into the buildTree function, where the many-branched Tree is constructed by splitting the documents to maximize information gain through term frequency. The tree function contains words at each junction (called split words) that are used to evaluate documents based on their containing such critical words. Additionally, there is a file called "Definitions.h," which contains all the parameters which can be tweaked to improve the speed and accuracy of the model.

To preprocess the input itself, first the raw input was read into separate strings as distinct words. Then, the punctuation is stripped, the letters are all set to lower-case, and the HTML break tags are removed as well. Finally, each word is run through a crude suffix stripper that removes only a few basic suffixes from the words.

The primary metric used was term frequency, measured according to the Okapi BM25 model:
$normalized\ TF = \dfrac{F\ *(k+1)}{F + k\ *\left(1 - b * \dfrac{docLen}{avgDocLen}\right)}$, where F is the term frequency and k and b are parameters that can be tweaked to improve the performance of the model. This model uses the normalized term frequency to decide which words are most significant to the evaluation of the documents' sentiments.

This implementation does no pruning after the tree has been constructed, but instead utilizes prepruning through a number of modifiable values. First, the term frequencies are selected between two different minimum and maximum thresholds. A minimum threshold prevents words from being used that might cause the tree to create overly-specific branches and avoids utilizing terms that are too infrequent to be of common use. The maximum threshold effectively eliminates branching the tree on words that are ubiquitous and so commonly used among all documents that they don't offer useful information (also known as stop words). Additionally, a coarseness value is introduced, which simply stops the tree from branching when the branch size gets too small as another measure of avoiding overfitting. Finally, the tree also utilizes a branching factor, which can change both the speed in which the tree is constructed as well as the quantity (and thus implicitly the quality) of split words chosen at each tree level.

Initially, the program was able to run in 40 seconds with 58% accuracy on testing data and 91% accuracy on training data. However, after significant tuning on the separate parameters of the model, the program can now achieve 74% accuracy on testing data and 90.6% accuracy on training data within 30 seconds. Additionally, the quality of the words the tree splits on has noticeably improved, moving from largely irrelevant words to words that are notably significant of a review's sentiment. For example, the current iteration of the tree has the following split words at the top of the tree (i.e. the most significant spot to split at): "worst", "awful", "waste", "lame", "terrible", "crap", "dull", "unless", "worse". All of these words are clearly indicative of a negative sentiment (with the possible exception of "unless"). However, note that "worst" and "worse" are both among the most significant words. One of the shortcomings of my algorithm is that the stemmer cannot effectively distinguish that these two words, along with many others, share the same stem when they appear not to for the tree builder. Another flaw with the method is that term frequency is a very limited metric for building a decision tree. Splitting the tree based on a single word almost always results in a very lopsided tree, which makes the tree less useful in practice. Additionally, the bag-of-words model loses all context which could be useful for a feature-based model.

In fact, the biggest challenge was in building the frequency-based model. Originally, I had implemented an entropy-based model, splitting around all the different words. However, that method requires simply too many splitting options, since there are so many words, and it necessitated that I find an information gain function that was easier to calculate, which turned out to be the normalized frequency model I am currently using. Even after I switched to a frequency model, however, speed was still an issue. To solve this issue, I increased the amount of branches allowed to the tree, while at the same time I also made this branching factor a parameter that could be changed. Giving one additional parameter that could be tuned gave a large increase in speed and, surprisingly, accuracy as well, which solved the speed problem quite nicely.