CSCI 455: Project 2

Brayden Faulkner and Christina Hinton

March 24, 2019

1 Introduction

The purpose of this program is to process a large number of HTML files, separate them according to their class (student, faculty, staff, department, course, project, other), remove the HTML tags and stop words, stem the words, and then finally find the TF-IDF weights of the top 1000 words. The program prints to console each document's highest weighted words and saves to file the TF-IDF weights.

2 Overview

We used an array of Dictionary<string, double> named "terms" to keep track of each class's list of words. We used another Dictionary<string, int> named "words" to keep track of the number of classes in which the word appears. For instance, if the word "computer" appears in all seven classes, the value in "words" would be 7, which would mean that "computer" would exist in each array in "terms." The program opens with the Overseer class. Overseer calls FileProcessor, which goes through each file in the directory. FileProcessor keeps track of which class (student, faculty, etc) a file belongs to, and allocates a new element of the Dictionary array each time a new class folder is opened. It then calls methods to remove HTML and stop words and stem the words. Number of word appearances are tracked in the array "terms" as each new word is stemmed. When a new word is added to "terms," we check to see if it exists in "words," and increment accordingly. We only check when a new word is added because we only want to increment a value in "words" once for each class. Finally, the Dictionary array is returned to Overseer, which sorts each class's list of key-value pairs according to the value and calculates the TF-IDF weights of the top 1000 most used words. The program prints the words with the highest weights to the console, and saves the TF-IDF representations to the file results.txt.

3 Libraries

We decided against using libraries, as many of the processes provided a better learning experience to do ourselves. For instance, we initially used a library for HTML removal, but it only bloated the process and gave us mixed results. We did, however, find a porter-stemmer algorithm that mirrored the results of the Python stemmer, and used that.

4 Conclusion

We switched to C# to help reduce the running time of the program. The running time went from around a minute to 30 seconds. Additionally, moving where each step takes place helped to lower the runtime.

5 Results

Predictably, the most used words from the previous project don't make much of an appearance in the results. We believe it is because the most used words (like "compute") are used frequently across all classes, and therefore would not help a user find a certain document if used as a query term. The list of terms now seem to correspond with their respective class. For instance, "course," uniquely has terms about coursework ("midterm," "reading," and "prerequisites"), while "other," seems mainly preoccupied with topics like "crime," "handgun," and "murder."

References

[1] Kamil Bartocha. "The Porter2 stemming algorithm."http://snowball.tartarus.org/algorithms/english/stemmer.html. Accessed March 3, 2019.

[2] Doug K. "How to Code a Search Helper Class to Clean Stop Words with C#." *Got To Know.* https://www.gotknowhow.com/articles/code-search-helper-class-clean-stop-words. Accessed March 3, 2019.