CS 202 Group Project
Alaskan Text Surveyor

Bryan Beus
Sarah Carter

March 9, 2020

Source Code Link:
`https://github.com/siddhartha-crypto/cs202_group_project.git`
Git Commit Messages
`https://github.com/siddhartha-crypto/cs202_group_project/commits`
This project took approximately 12.5 hours to complete.

1 Pitch

The Alaskan Text Surveyor scans text files and creates a search and classification structure that extends beyond normal "Search" functionality.

The results of the scan display various output that are useful for readers, such as library-science classifications for texts and passages within the texts.

Furthermore, the reader is able to enter search functions that allow the reader to discover associated meanings of large quantities of text.

For example, the project may allow the reader to search for passages that are "negative" or "positive" in tone, or other related tasks.

The limitations of the Alaskan Text Surveyor will become more clear as the project develops. Natural language processing is a

broad field, and the scope of this project and class only allow for a limited amount of functionality.

2 Project Iteration 1: Design

2.1 Overall Design

The Text Analyzer software performs two main actions.

The software first scans into its memory a few dozen preset text documents that establish a baseline analysis. These documents derive from disparate categories, such as biography, science, geography, and more, to create a broad overview of the field of literary analysis.

The software performs several types of analysis on the texts, including section breakdowns, word count analysis, content type, and more, and provides a final report of the software's findings.

Having completed the preset baseline documents, the Text Analyzer software allows the user to choose another text (or texts) on which the software is to perform a similar analysis.

This analysis performs all the same actions as before, and also creates a comparison between the user's chosen text and the preset texts.

These elements are displayed in a graphic user interface (GUI), allowing the user to visual the content of their indicated document.

2.2 Prior Art

Three main types of inspiration include search engines, machine-learning software for language processing, and the common grammar improvement features found in major word-processing software.

The databases and searching functions of popular search engines, such as Google.com and Amazon.com, feature text analyzers that perform analysis on all documents available to the user through these portals.

Machine-learning software that attempts to auto-complete what a user is currently typing into a search bar is another type of software that performs detailed analysis on text documents. Many

types of machine-learning software perform these functions; a popular example is TensorFlow.

The simplest example is the grammar-completion software that exists in common word processors, such as the Microsoft Office Suite's Word processor.

2.3 Technical Design

The Text Analyzer is built in C++. For the text analysis portion of functionality, the software uses the or the GUI, the software uses the ibrary.

The software needs several classes, including `Analysis`, `Display` , and report `Report`. These classes perform analysis on provided text, display analysis findings to the user, and store report data. Each class has its own header and source file. A `Miscellaneous` class can hold any additional functionality. The overall structure of the software is defined in the `main()` function.

Sarah is interested in focusing on the GUI aspect of the software, while Bryan is interested in focusing on the text analysis aspect.

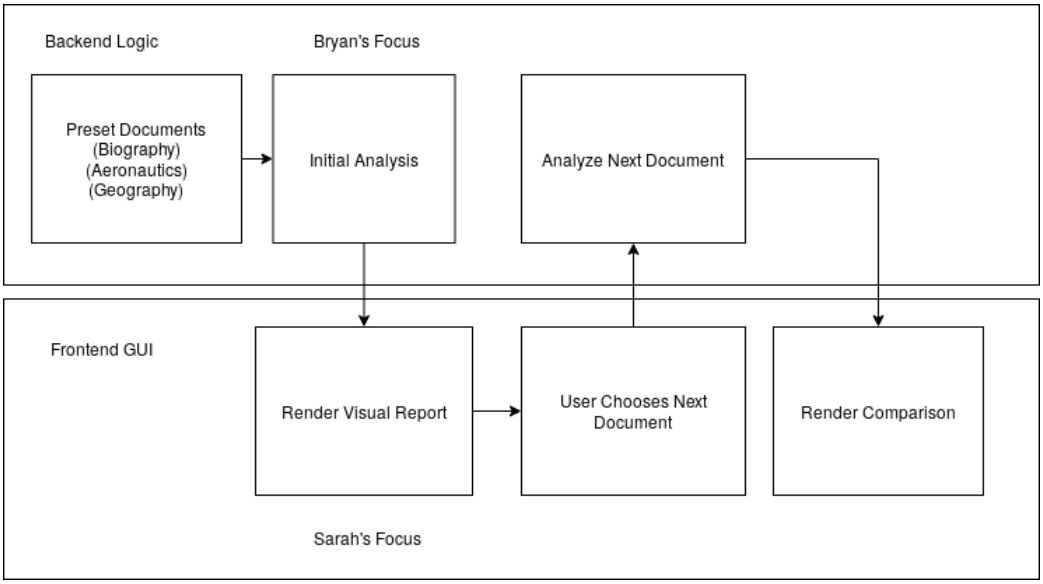

Figure 1: Text Analyzer Flow Chart

2.4 Required Libraries

The GUI aspect of the software relies on the FLTK library. This library is ideal for the software's purposes as the functionality appears to be user friendly from a developer's point of view. All functions necessary to perform the software are provided in the library.

The text-analysis aspect relies on the MeTA toolkit. This library includes many of the analysis features desired in the software, including topic models, classification algorithms, and more.

3 Project Iteration 2: Initial Prototype

In this section, each group member needs to give an update on the work they are doing. Write about the goals you achieved for your program and the goals you set for the next iteration.

3.1 Sarah Carter

My contributions thus far have included extensive research into the use of FLTK. I'm trying to learn how to make an interactive GUI that allows the user to browse their computer's directories for files they'd like to be analyzed. At the moment, I have imported a static output file from the analyzer that Bryan designed and displays its contents. I have learned a fair amount about various widgets I plan to use in the ultimate code, but for delivery at this time, a basic display of output was what I chose.

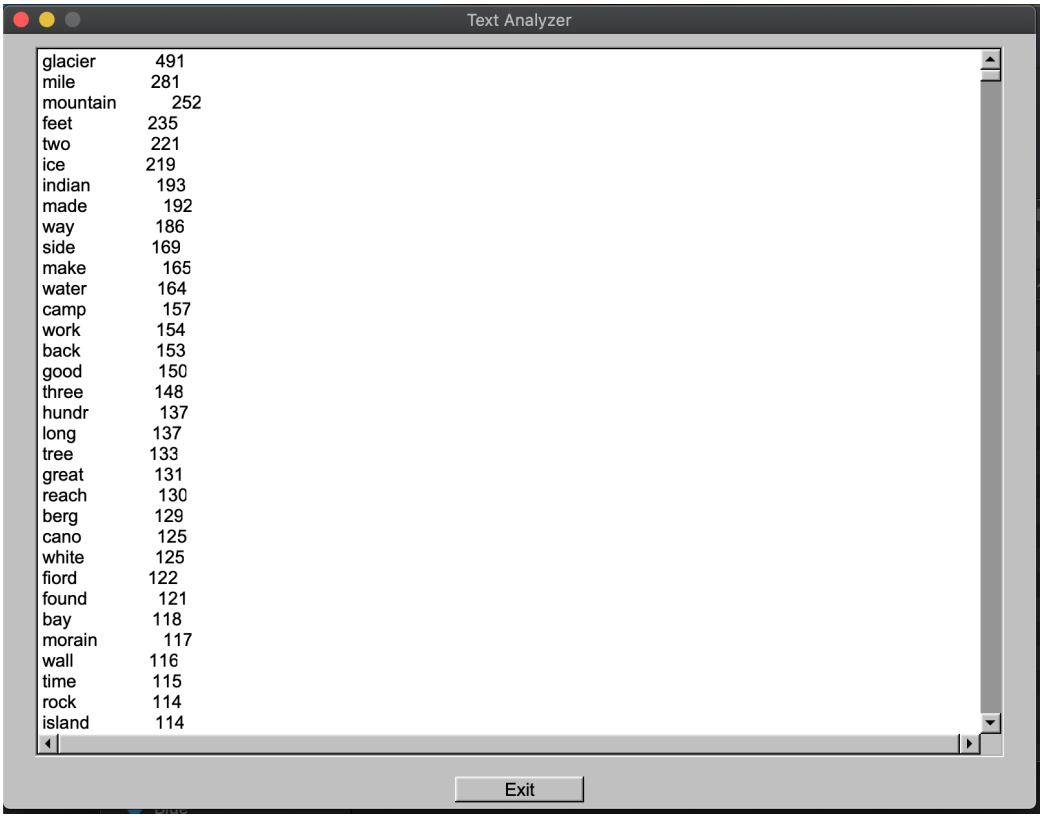

Figure 2: Sarah Carter Artifacts

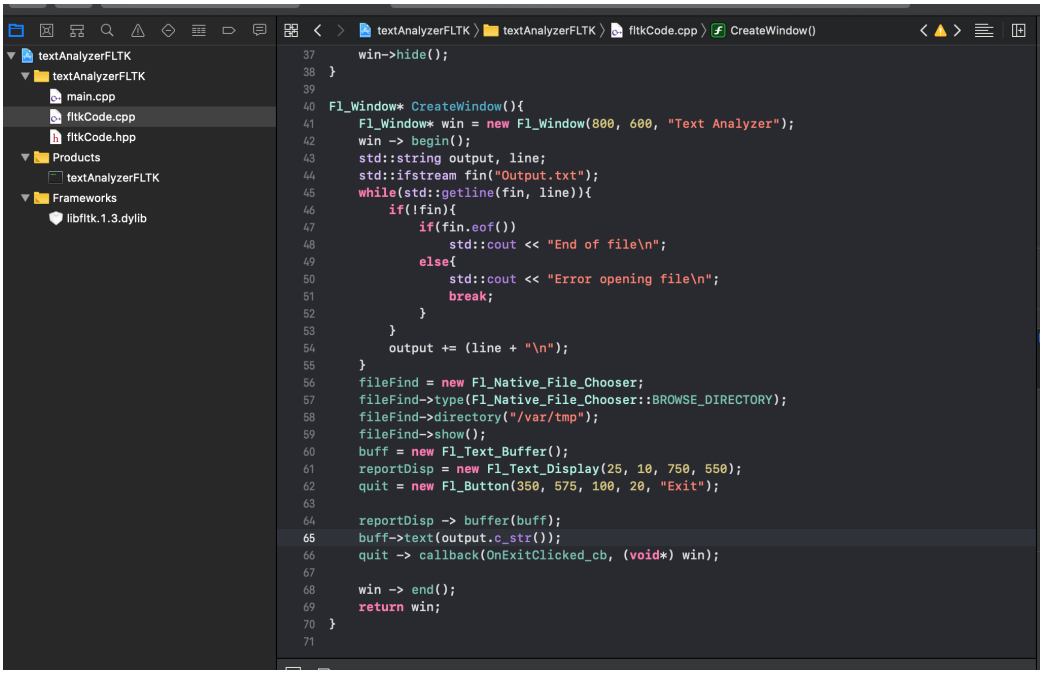

Figure 3: Sarah Carter Artifacts

3.2 Bryan

In this iteration of the project I focused on learning how to use the MeTA toolkit. The code is powerful, but not well maintained. No significant update has been released since Ubuntu 14.10.

In the process of building the program I had to find several workarounds, including building symbolic links to libraries in the Unix library that were present when the MeTA code were written but have since been removed.

The source code for MeTA assumes the user is familiar with the build process, and that the user expects to import the source code from the project into other toolsets.

Linking all the libraries between my own source code and the MeTA source was too advanced for my current abilities, but I was able to build the project simply as a standalone program. This was the source of the output data for this first iteration of the software.

I suspect that with continued effort, we will be able to import the MeTA source code so that we can use it inline with our own source code. This is what the MeTA library is built for, and it most powerful in this capacity.

For now, we ran some analytics on John Muir's famous observations of Alaska, and displayed this in the FLTK output.

In the current state, the program is split into several separate sections. The MeTA submodule is currently standalone. The `data` directory contains a separate program that further cleans the output data. The FLTK section displays data.

This allows the audience to see data on the texts immediately and to view the building concept.

In the next iteration, we hope to be able to incorporate the source code together, run the analytics on several sets of texts, and display the observations in FLTK, as per our original goal. 6

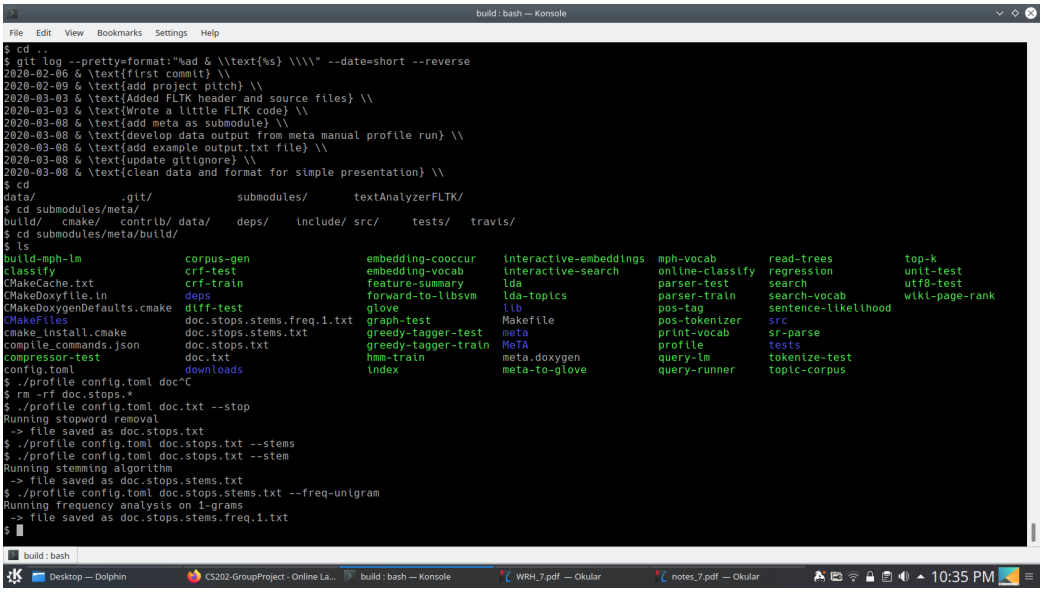

Figure 4: Using MeTA to Process John Muir's Observations on Alaska

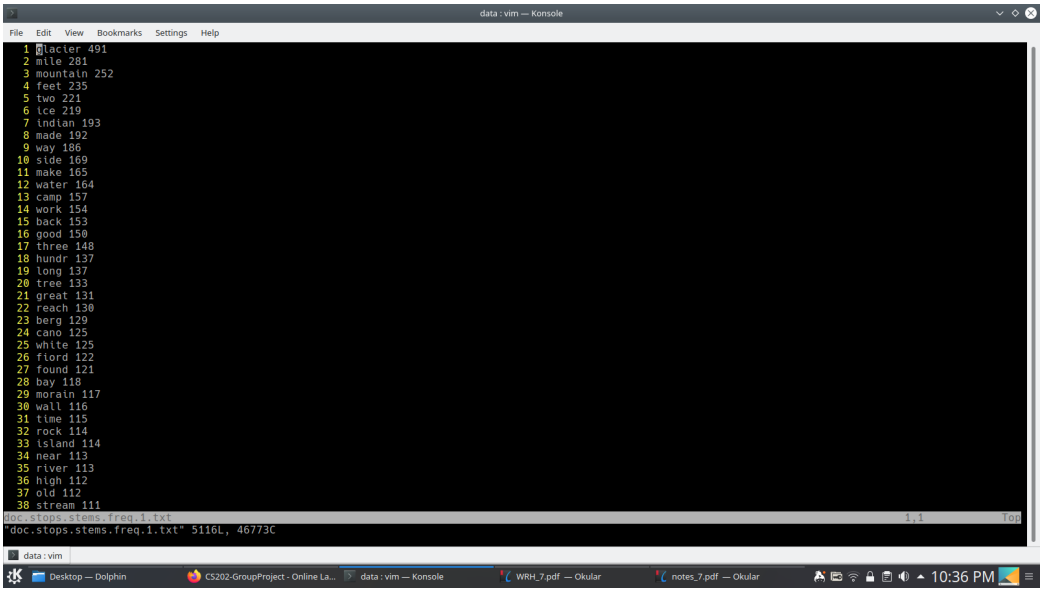

Figure 5: Raw MeTA Output

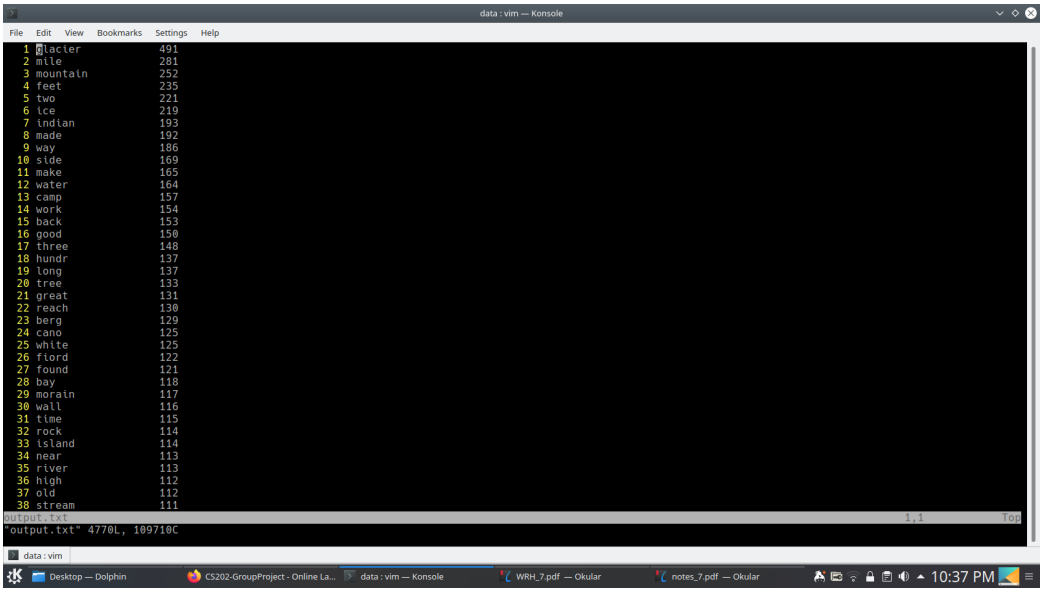

Figure 6: Cleaned MeTA Output

3.3 Git Commit Messages

2020-02-06 first commit
2020-02-09 add project pitch
2020-03-03 Added FLTK header and source files
2020-03-03 Wrote a little FLTK code
2020-03-07 Added directory browser to FLTK window. Doesn't work yet.
2020-03-08 Created exit button; trying to add a file browser button to choose files to import.
2020-03-08 add meta as submodule
2020-03-08 develop data output from meta manual profile run
2020-03-08 add example output.txt file
2020-03-08 update gitignore
2020-03-08 clean data and format for simple presentation