Summarizer51: Final Report

Eric Ouyang
Frederick Widjaja

May 2nd 2014
Computer Science 51

Video Overview URL: http://youtu.be/0JgzphtXOYU

Overview

In this final project, we implemented algorithms to perform automatic keyword finding and text summarization. Specifically, our keyword finder used TextRank and our summarizer used TextRank and LexRank. We investigated and adapted techniques to parse free form text, analyze its content, and identify key words and sentences. To determine the part of speech of a given word, which enabled us to generate better keywords, we used the Brill Tagger algorithm.

Our program first parses a given text file. Using regular expressions, we split a body of text into sentences and words. Afterwards, we run all of the words through a tagging algorithm to determine its part of speech. Our tagger utilizes a large lexicon with parts of speech to determine the appropriate tag for most words. We supplement this lexicon with a set of general rules within English to infer parts of speech. Words that are not in the lexicon or do not fit the general rules are assumed to be nouns. These parsed forms are then passed along to the various algorithms.

The TextRank keyword finder travels through the tagged words and adds them into the graph with its position if they meet a certain set of criteria (in our case, being a noun or an adjective). Positions are kept track of within a Node class, which contains a list of positions. Afterwards, an adaptation of the PageRank algorithm is used to determine the relevancy of specific words. We define an (unweighted) edge in our graph to exist if two words are within a predetermined “co-occurrence” window. Finally, we run through the text again to determine whether or not highly ranked words in fact are highly ranked phrases, combining them as needed. The top keywords are then returned back.

The TextRank sentence extraction summarization algorithm, like the keyword finder, uses an adaptation of PageRank to determine the relevancy of sentences. Two nodes, representing sentences, have a weighted edge as a function of the number of words that occur in both sentences normalized by the length of the sentence.

The LexRank sentence extraction summarization algorithm uses many of the principles in TextRank, however implements a much more sophisticated similarity algorithm than just the number of common words.

Planning

We spent a lot of time planning our program upfront and intended to write it in OCaml. However, as described in our Functionality Checkpoint, we ran into significant issues with OCaml programming. After spending a lot of time debugging, we decided it was best for us to implement it in Java, especially since Java 8 has a lot of functional-esque features.

We were able to implement all of our planned core features and were able to implement additional features including implementing additional algorithms (LexRank), enabling various amounts of detail, and a command line interface with our program that allows users to direct set the filename to summarize as well the algorithm to utilize.

Design and Implementation

Throughout the process of creating our final project, we constantly kept abstraction in mind to minimize code waste and promote modularization. Specifically we:

- created interfaces for keyword and summary extraction so that future work on this program can include drop-in replacements for TextRank and LexRank
- created an abstract graph, which could be of various types (Sentences, Words, NGrams, etc.) as keys which could have various types as nodes (generic nodes, TextRank nodes, TextRank nodes with positions, etc.)
- utilized functional programming paradigms and higher-order functions such as *map* and *reduce*

There were times at which we recognized that we needed to factor out code. While helpful in the long-run, it was a bit annoying at the time to need to rework some code to meet new abstractions.

We worked side-by-side for most of the implementation and split up the work evenly. For key areas such as implementing the PageRank algorithm, we partner-coded.

Reflection

This was our first time using Java 8, and we were really impressed by the its new features. Concepts we learned throughout the term at applied to OCaml translated in a few key locations to our program.

We were pleasantly surprised by how elegant some of the code was for the TextRank summarization algorithms, especially when we used higher order functions.

However, we ran into difficulty implementing the TextRank keyword finder, especially because of the various complicating factors (needing to account for part of speech, recombine words into phrases, etc.)

If we had more time, we would like to have implemented a parser that could take a URL and parse the body of the page and return the summary.

If we were to redo this from scratch, we may have used a library for text parsing, such as OpenNLP, so that we would not have to reinvent the wheel, and focus on the core functionality instead.

Advice for future students

Using a library for text parsing, such as OpenNLP, would prevent you from re-inventing the wheel. While we found it a good learning experience to implement our own parser, part of speech identifier, etc., perhaps we would have been able to accomplish more if we had built upon others' work.

Through this program, it was valuable to learn a bit about how the core of the world's largest search engine, Google, works. Additionally, it showed us a bit about the difficulties that researchers in natural languages face. Best way to learn these things is by trying it out first hand!