Summarizer51

CS51 Final Project
Draft Specification

Eric Ouyang
Frederick Widjaja

April 13th 2014

Overview

Our final project aims to investigate and implement various text-summarization algorithms in OCaml. Specifically, we aim to develop techniques to parse free form text, analyze its content, and identify key phrases and words. With information being provided to us in increasingly large volumes through email and news, we think it's important to be able to consume information more efficiently. A summarization algorithm enables individuals the flexibility to gain a basic understanding of information without needing to spend the time to read through all extraneous information. While we do not aim to eliminate the need for reading full-text, our algorithm hopes to give individuals the tools to decide what may be most important to read.

Features

Core features

- Input mechanism
    - enables users to easily provide to the program the text that they want to summarize
    - two ways of input:
        - direct input via console
        - file input (provide a filename)
- Text parser into useful data that our algorithm can utilize
    - an algorithm to turn unstructured free-form text into a useful data structure that the keyword and relevancy algorithms can utilize
- Keyword generator
    - uses data structure representing the text and ranks the importances of words within the text
- Sentence/phrase relevance ranking algorithm
    - an unsupervised algorithm that takes a data structure representing text and ranks the importance of key sentences/phrases
- Output mechanism
    - enables users to easily view the program output
    - two ways for output:
        - direct output to console
        - file output (provide a filename)

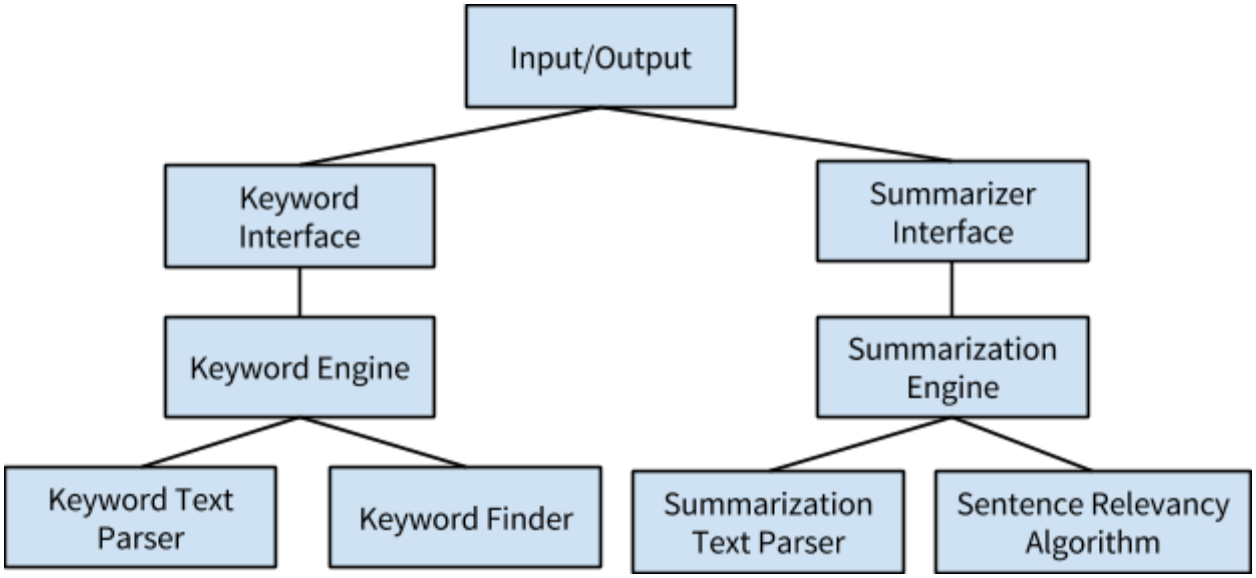
Extensions (in order of priority)

1. Implementation/hybridization of multiple algorithms
    a. compare and contrast different algorithms, perhaps including a supervised learning algorithm
    b. combine various ranking strategies to form an overall rank for keywords and/or sentences/phrases within the text
2. Parameter for amount of detail

    a. allows users to specify the number of keywords and/or the number of sentences/phrases they want the program to output
3. Web-based interface for the application
    a. create a basic web server API that can provide the output of the program as JSON
    b. create a basic javascript-based web app that can take input text, send it to the server, receive and parse JSON output, and present to the user
4. Input from online sources
    a. allow users to provide a URL (e.g. of a news article)
    b. program will parse HTML, strip unnecessary content, and run algorithm on the content of the website

Technical Specification

Application Flow Diagram



Application Components

Input/Output

- get_input: filename -> stream
- get_keywords: stream -> set of words
- get_summary: stream -> set of sentences
- print_keywords: set of words -> standard output
- print_summary: set of sentences -> standard output

Word type: string

Sentence module
- type t
- make: word list -> t
- get_nth_word: t -> int -> word
- iter: t -> (word -> unit) -> unit

Paragraph module
- type t
- make: sentence.t list -> t
- get_nth_sentence: t -> int -> sentence.t
- iter: t -> (sentence.t -> unit) -> unit

Text module
- type t
- make: paragraph.t list -> t
- get_nth_paragraph: t -> int -> paragraph.t
- iter: t-> (paragraph.t -> unit) -> unit

Keyword module signature (public)
- make: stream -> Keyword module
- get_keywords: Keyword module -> set of words

Keyword module helpers (private)
- parse_stream: stream -> Text.t
- calculate_keyword_rankings: Text.t-> set of records containing words and rankings

Summary module signature (public)
- make: stream -> Summary module
- get_summary: Summary module -> set of sentences

Summary module helpers (private)
- parse_stream: stream -> Text.t
- calculate_sentence_rankings: Text.t -> set of records containing sentences and rankings