MEMORIAL UNIVERSITY OF NEWFOUNDLAND
FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

Text Summarization

Computer Science 4750

Jacob House Noah Gallant

Friday, November 30th, 2018

Contents

1 Topic

Motivation to study computerized text summarization — referred to by the term *automatic abstracting* by those in the field — stemmed from curiosity about the mechanisms used in an automatic abstraction PowerShell script found years ago on the Internet [1]. Further research has shown automatic abstraction's usefulness in many interesting fields, including but not limited to the legal and medical professions, scholarly research, and search engine result sorting and summarization.

2 Background Research

3 Our Implementation

The implementation chosen mainly followed the graph method. Three primary files were created containing the three main classes of the software. These classes were the node, the edge, and the paragraph class. Combined together these are called by a summarizer Python file which has the role of opening the summary file and ensuring the correct format is used before performing the summarization.

3.1 Design

3.2 Classes

3.2.1 Text Class

The role of the `Text` class is to perform all operations on the text. This includes any pre-processing required, the initial object creation, determining further relations, as well as using the determined relationship information to form a reasonably accurate and concise summary.

When first initializing the text, the text to be summarized must be passed into the constructor of the `Text` class. This text is then saved as an object variable for future use, and proceeds to undergo pre-processing. Pre-processing is done by the `preProcessing()` method, and is tasked with splitting the text into sentences.

preProcessing() Pre-processing is a crucial step in achieving a high quality summary. Since this is the first modification made to the text, the method in which pre-processing is conducted can have a significant bearing on the quality of the summary. The goal of the pre-processing in this application is primarily to split the text into sentences. This is done by considering all sentence terminating symbols including '.', '!', and '?'. This also

considers if a sentence contains a quote, and any other situation in which a terminating symbol should be ignored.

The secondary operation of the pre-processing in our application is to replace special Unicode characters with their plain text equivalent. Many common text editors use Unicode characters as opposed to the plain text characters for many text symbols. Some of these symbols include quotation marks, accents, en dashes and em dashes.

Processing Sentences

Once the pre-processing is complete

Creating the Dictionary

After all the sentence nodes have been created, and a list of words in each sentence node has been processed, The `Text` object will then proceed to loop through each sentence node to create a complete dictionary of all the words found in the text. This dictionary will not only store the available words, but will also store all the sentence node in which the word is contained in. The goal of creating this dictionary is to decrease the complexity in creating the relations between sentences. By creating a dictionary which contains words and the nodes they are contained within, we must simply loop through all the words and create relations when a word is contained within more than 1 node.

3.2.2 Node Class

The `Node` class is the applications representation of a sentence. The `Node()` constructor takes in a sentence as a string, and will immediately proceed to split the string into words which are saved in a list.

To reduce the complexity of the code, instead of performing lemmatizing in the pre-processing stage as might be expected, it was decided to perform this step at the same time the words in the sentence were being divided. This was decided so as to prevent accessing a word more than was required.

The final task of the `preProcessing()` method is to use a lemmatizer to remove word endings. When comparing words later in the application to determine relevant sentences, this will improve the results as it will more accurately represent similar words, rather than counting the same words with identical endings as different words and therefore misrepresenting the relations in the text.

3.2.3 Edge Class

4 Optimization

References

[1] Prateek Singh. *PSSummary*. 2018. URL: https://github.com/PrateekKumarSingh/PSSummary.

Appendix

A Demonstration