In-browser Text Importance Detector

Monday 30th May, 2022 - 00:53

Paulo Ricardo Botelho Oliva
University of Luxembourg
Email: paulo.botelho.001@student.uni.lu

This report has been produced under the supervision of:
Luis Leiva
University of Luxembourg
Email: luis.leiva@uni.lu

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1. Introduction

This Bachelor Semester Project is all about finding important sentences in a text. The main objective of the project is to create a browser extension that highlights the most important information of an article's web page. Moreover, this project also addresses the topic of automatic text summarization, as well as extractive text summarization algorithms.

2. Project description

2.1. Domains

2.1.1. Scientific. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis.

2.1.2. Technical. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis.

2.2. Targeted Deliverables

2.2.1. Scientific deliverable. This BSP's scientific deliverable aims to answer the following research question:

How can we detect important sentences in a web document?

To help answer this question, we will need to answer the following secondary questions:

1. How can we automatically detect text blocks on a web page?

2. How can we extract relevant sentences within a text block?

To answer our main scientific question, we will investigate the concept of automatic text summarization. Automatic text summarization is the technique of creating a shorter version of a text using a computer program. This shorter version of the text (i.e., the summary) contains the most relevant information of the original text.

Automatic text summarization methods are very much needed to deal with the ever-increasing volume of text data available online, both to help identify relevant information and to consume relevant information more quickly. There are many reasons why one would need automatic text summarization. For example, summaries reduce reading time and should be less biased than summaries made by a human person [1].

In general, there are two main kinds of text summarization techniques, extractive text summarization and abstractive text summarization. Extractive text summarization consists in selecting the most important parts of a text and extracting them to create a shorter summary of the same text. Abstractive text summarization consists in gathering the main ideas of a text and using those abstract ideas to generate a summary of the text, where the summary isn't just a copy of the old text, but actual new sentences.

Given the scope of this BSP, our scientific deliverable will focus on extractive text summarization. There exist many approaches to extractive text summarization, like statistical approaches, lexical chain based approaches, graph-based approaches, cluster-based approaches, and fuzzy logic based approaches, among others.

We will focus on graph-based text summarization, as it is commonly used by many algorithms to classify sentences by their importance on the source text. For example, Google's PageRank is a graph-based ranking algorithm used by Google to rank web pages [2]. This algorithm is the basis of other well-known graph-based text summarization algorithms like TextRank [3] and LexRank [4].

The graph-based method of text summarization is an unsupervised technique in which sentences or words are scored using a graph, therefore their convenience for this BSP. In a nutshell, the basic goal of graph-based methods is to extract the most relevant sentences from a block of text. Thus, the scientific deliverable will focus on explaining how graph-based text summarization algorithms work.

To answer the secondary questions, the notion of text blocks should be explained. In the context of web pages, a text block is considered a group of paragraphs. Critically, we need to identify the most important text blocks in a web page (the main content) as well as unimportant text blocks, such as texts in the header or the footer of the page. Unimportant texts should not be considered for summarization.

A key challenge is that automatic detection of the main text blocks on a web page is not reliable, since every web page has a different structure and layout. Thus, detecting blocks of text is a hard task. The scientific deliverable of this BSP will study this question more thoroughly. As there does not seem to be a trivial solution, we will try to find the best compromise solution.

2.2.2. Technical deliverable. The technical deliverable of this project focuses on creating a browser extension that detects the most important sentences on a specific web page and highlights them, so that the user only needs to read the highlighted sentences instead of the whole web page.

A browser extension is a small software package that adds features and functions to a browser. Extensions are built using well-known web technologies such as HTML, CSS, and JavaScript. Extensions rely on their own set of APIs, which are browser-dependent. For example, they allow developers to change the browser's default behaviours or inject custom stylesheets of JavaScript content before a web page is loaded.

The extension should be built using JavaScript and should work on the Google Chrome browser, given its large market share of 65-70% [5] [6], without breaking the functionality of existing websites. The extension will automatically highlight the most important sentences of an article or blog post in a specific web page.

In short, the goal of this technical work is to implement text summarization techniques in a browser environment, while creating a simple extension that is easy to use and that works on most websites.

3. Prerequisites

Before starting the project, a basic understanding of a few topics is required. In particular, a basic understanding of web technologies, such as HTML, CSS, and JavaScript, is required.

3.1. Scientific prerequisites

There are no scientific prerequisites for this Bachelor Semester Project.

3.2. Technical prerequisites

The only required technical competency before starting to work on this Bachelor Semester Project is possessing a basic amount of knowledge about HTML, CSS and the JavaScript programming language.

4. Scientific Deliverable

4.1. Requirements

The scientific deliverable produced in this project consists in answering the question "How can we detect important sentences in a web document?". In this case, a web document is any web page presented to us in the browser while browsing the Internet.

This question has been split into two parts: the first part is about automatic text detection in a web page, and the second part is about extracting the most important sentences from a text block. Thus, the scientific deliverable requires these two questions to be answered, ultimately leading to a conclusion for the question "How can we detect important sentences in a web document?".

4.2. Design

4.3. Production

4.4. Assessment

5. Technical Deliverable

5.1. Requirements

The requirements are the characteristics that are required of the completed technical deliverable. The software developed for the technical deliverable must meet a number of functional and non-functional requirements.

5.1.1. Functional Requirements.

- Gathering the text on a web page.
- Generating a summary of the gathered text.
- Highlighting the important text on a web page.
- Running at least on Google Chrome.

Gathering the text

The browser extension must be able to gather the text of any web page. The extension must contain a function that gathers

the text of a web page. The function must be able to gather the text of any web page.

Generating a summary

The extension must allow the user to generate a summary of the text gathered from the web page. The summary must be generated using the most important sentences of the text, with the help of an extractive text summarization algorithm. The user must be able to configure the summary to be generated. The parameters of the summary must be configurable, depending on the summarization algorithm used. Such parameters are, for example, the length of the summary generated.

Highlighting the important text

The extension must be able to highlight the important text on the web page, based on the summary generated by the summarization algorithm. The most important sentences of the text must be highlighted after the user has generated a summary.

Running at least on Google Chrome

The browser extension must be able to run at least on Google Chrome. Other browsers may be supported, but the extension must be able to run on Google Chrome without errors.

5.1.2. Non-functional Requirements.

- Providing a high-quality summary.
- Maintaining the functionality of existing websites.

Providing a high-quality summary

The summary generated by the summarization algorithm must be high-quality. A high-quality summary should allow the user to quickly understand the content of the web page.

Maintaining the functionality of existing websites

The browser extension must not break the functionality of existing websites. Thus, if the user decides to use the extension on a website, all the website's features must remain functional.

5.2. Design

This section explains the design of the application developed in the technical deliverable. The software is written using VS Code, although any other text editor would be enough for the task. The browser extension is written using HTML, CSS and JavaScript in conjunction to make the extension.

5.2.1. Libraries used. This project mainly depends on 3 different sets of libraries to work:

- Summary.js
- textrank.js
- naiverank.js

Summary.js is "a lightweight paragraph summarizer library" that can be configured to appease to the user's preferences. It uses the LexRank algorithm to score sentences and gives the best scoring sentences back as a summary.

Textrank.js is a library that can be used to rank sentences in a text. It uses the TextRank algorithm to rank sentences. It depends on the Pagerank.js library.

Naiverank.js is a library that can be used to rank sentences in a text. It uses the NaiveRank algorithm to rank sentences. Unlike the TextRank library, it does not have any dependencies.

5.2.2. Project Structure. An extension is basically a bundled package of different HTML, CSS and JavaScript files. This extension consists of 3 main parts: the content script, the pop-up and the manifest file.

Content Script

A content script is a file that runs in the context of a web page. A content script can read the details of the web pages visited by the browser and make modifications to them. It has access to the same Document Object Model APIs as normal injected scripts, while also having access to special browser extension APIs.

The extension makes use of this content script to be able to:

- Gather the text from the current web page
- Highlight the most important sentences in the current web page

Pop-up

A pop-up is an HTML document that is displayed to the user when they click on the extension icon. A pop-up is comparable to a web page in that it can include links to stylesheets and script tags, but it does not support inline JavaScript. In the end, our pop-up consists of 3 different files:

- popup.html
- popup.js
- popup.css

In our extension, popup.html provides the user with a way of interacting with the extension functionalities and settings. It does this by acting as a user interface for the browser extension. It allows the user to select which algorithm to use when generating a summary of the web page, and, depending on the algorithm, to customize the parameters of the generated summary, like the number of sentences included in it.

Moreover, the popup.js file contains all the JavaScript code to make the pop-up work as intended. It makes use of the external libraries previously described to generate a summary of the text gathered by the extension.

Finally, the popup.css file contains the CSS style rules to properly display the pop-up to the user.

Manifest file

Any extension needs a manifest file to be able to work. The manifest file provides basic metadata such as the extension's name, version number, and the permissions it requires. It also

contains references to other files in the extension such as the extension icon and, more importantly, defines where the content script is able to run.

5.2.3. Script interoperability. Until now, we have described the different parts of our browser extension. However, because of the way the extension has been designed, we must talk about how the scripts interact with each other to provide the expected functionalities of the extension to the user.

As previously mentioned, content scripts run in the context of a web page rather than the extension. In the other hand, our pop-up script runs in the context of the pop-up and cannot interact with the current web page as they're running in separate environments thanks to our browser. Since the content script is the one in charge of gathering the text from the web page, the pop-up script cannot do anything without the text that the content script gathers from the web page. Because of this, we require a way for both of the scripts to communicate with each other.

"Message passing" is used to execute communication between extensions and their content scripts. Thanks to "message passing", extension components, such as the pop-up script and the content script, can listen for and reply to messages sent by one another using the browser's extension APIs.

In our extension, the content script uses "message passing" to send the text to be summarized to the pop-up. In return, the pop-up sends the generated summary back to the content script, allowing the content script to highlight the important sentences in the web page.

5.2.4. User Interface. The pop-up's UI is defined using HTML and CSS. The user interface contains three buttons, one for each of the used summarization algorithms. When the user clicks in one of the buttons, an input box appears that allows the user to select how many sentences should be in the summary.

Moreover, if the user selects the LexRank algorithm, an extra input box is displayed enabling the user to specify the amount of keywords to factor when generating a summary. The user can detect the most important sentences in the web page by clicking the "Detect" button.

Finally, there are two radio buttons bellow the "Detect" button which allow the user to select the highlight mode for the extension. The first mode highlights individual sentences, while the second one tries to highlight the entire paragraph where an important sentence is found.

5.3. Production

This section describes the production of the browser extension developed in the technical deliverable. The production of the extension involves three parts: the creation and development of the extension different components, the implementation of the user interface as detailed in the design section before, and the implementation of the features detailed in the requirements.

5.3.1. Project Development. As stated in the design section, the extension is written using VS Code, since there is no need for a fully fledged IDE in this project. Any other basic text editor could also be used instead of VS Code. The extension can be loaded in Google Chrome as an unpacked extension (a folder containing the extension files) in the browser extension settings.

As previously stated in the design, the project consists of three main parts: the pop-up, the content script and the manifest. The main code of the project is contained within the `content_script.js` and the `popup.js` files.

5.3.2. Content Script.

5.3.3. Pop-up.

5.3.4. Manifest file.

5.3.5. Message passing.

5.3.6. Non-required features.

5.4. Assessment

Acknowledgment

The author would like to thank the BiCS management and education team for the amazing work done.

6. Conclusion

7. Plagiarism statement

I declare that I am aware of the following facts:

- As a student at the University of Luxembourg I must respect the rules of intellectual honesty, in particular not to resort to plagiarism, fraud or any other method that is illegal or contrary to scientific integrity.
- My report will be checked for plagiarism and if the plagiarism check is positive, an internal procedure will be started by my tutor. I am advised to request a pre-check by my tutor to avoid any issue.
- As declared in the assessment procedure of the University of Luxembourg, plagiarism is committed whenever the source of information used in an assignment, research report, paper or otherwise published/circulated piece of work is not properly acknowledged. In other words, plagiarism is the passing off as one's own the words, ideas or work of another person, without attribution to the author. The omission of such proper acknowledgment amounts to claiming authorship for the work of another person. Plagiarism is committed regardless of the language of the original work used. Plagiarism can be deliberate or accidental. Instances of plagiarism include, but are not limited to:
  1) Not putting quotation marks around a quote from another person's work
  2) Pretending to paraphrase while in fact quoting
  3) Citing incorrectly or incompletely
  4) Failing to cite the source of a quoted or paraphrased work
  5) Copying/reproducing sections of another person's work without acknowledging the source
  6) Paraphrasing another person's work without acknowledging the source
  7) Having another person write/author a work for oneself and submitting/publishing it (with permission, with or without compensation) in one's own name ('ghost-writing')
  8) Using another person's unpublished work without attribution and permission ('stealing')
  9) Presenting a piece of work as one's own that contains a high proportion of quoted/copied or paraphrased text (images, graphs, etc.), even if adequately referenced

  Auto- or self-plagiarism, that is the reproduction of (portions of a) text previously written by the author without citing that text, i.e. passing previously authored text as new, may be regarded as fraud if deemed sufficiently severe.

References

[1] J.-M. Torres-Moreno, *Automatic text summarization*. John Wiley & Sons, 2014.

[2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[3] R. Mihalcea and P. Tarau, "Textrank: Bringing order into text," in *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004, pp. 404–411.

[4] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of artificial intelligence research*, vol. 22, pp. 457–479, 2004.

[5] NetMarketShare, "Browser market share," 2020. [Online]. Available: https://netmarketshare.com/browser-market-share.aspx

[6] StatCounter Global Stats, "StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share," 2022. [Online]. Available: https://gs.statcounter.com/

[7] BiCS(2021), "BiCS Bachelor Semester Project Report Template," University of Luxembourg, BiCS - Bachelor in Computer Science, Tech. Rep., 2021. [Online]. Available: https://github.com/nicolasguelfi/lu.uni.course.bics.global

8. Appendix