A Google Chrome Extension for Summarising News Articles

Based on the TextRank algorithm

Ishmael Aqsar

1416344

Supervisor: Dr. Phillip Smith

College of Engineering and Physical Sciences
University of Birmingham

This dissertation is submitted for the degree of
Masters in Computer Science

Department of Computer Science March 2019

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 60 pages including appendices, bibliography, footnotes, tables and equations.

Ishmael Aqsar
1416344
March 2019

Acknowledgements

I would like to thank my supervisor, Dr Phillip Smith, for the patient guidance, encouragement and advice he has provided throughout my project. I have been lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly. I would also like to thank all those who have come before me in the field of computer science. None of this could have been possible without the work they have done and the contributions they have made to this exciting field. And last but certainly not least, I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are the ultimate role models.

Abstract

The following paper describes the design of a Google Chrome extension, which provides users with the ability to summarise news articles with a single button press. The implemented system uses cosine similarity of TF-IDF vectors values as a similarity measure between sentences in the TextRank algorithm and the linguistic features used in the construction were stop words, and stemming.

The system was evaluated by summarising four news articles online and comparing them to human created model summaries. Comparison between the system summaries and model summaries was made with the ROUGE-1 measure. The implemented system performed best with the search radius, n, set to 2, having an average precision of 42.7%, recall of 50.8%, and overall average F_1 score of 45.2%.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

ATS Automatic Text Summarisation

CSS Cascading Style Sheets

DOM Document Object Model

DUC Document Understanding Conference

HTML Hypertext Markup Language

ROUGE Recall-Oriented Understudy for Gisting Evaluation

TF-IDF Term Frequency-Inverse Document Frequency

UI User Interface

Chapter 1

Introduction

Online material is a crucial source of knowledge, but due to the large amounts of data, it becomes necessary to effectively summarise for it to be useful. The role of web browsers no longer solely consists of displaying static HTML content. Instead, web browsers must support interactions between users and highly dynamic web pages. In order to cope with the complexity of such pages, users may choose to personalise their browsing experience by installing browser extensions.

1.1 Problem to be solved

The problem this project looks to solve is:

Is it possible to produce a Google Chrome extension to effectively summarise online news articles?

To answer this question, the problem is divided into four parts:

1. Implementation of a summarisation algorithm.
2. Evaluation of the algorithm above.
3. Design of the Chrome extension itself.
4. Testing the performance of the extension.

1.2 Aims

The goal of this project was to build an extension for Google's Chrome browser, which implements a suitable automatic summarisation method to allow users to summarise the current news article they are reading in a user-friendly manner. The chosen method was then evaluated by comparing the constructed summaries with human created summaries. The comparisons were made with the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metric set, and specifically with the ROUGE-N measure, which measures *n-gram* (A sequence of *n* words from a text) overlap.[1]

1.3 Outline

This report is divided into eight chapters. The first chapter is this Introduction, and the remaining chapters are organised as follows:

- Chapter 2 provides further background information in the context of related existing work, commercial products, and research papers.
- Chapter 3 gives a high-level account of the structure of the work and describes its functionality.
- Chapter 4 provides detailed account of the implementation and testing of the work.
- Chapter 5 describes the evaluation process.
- Chapter 6 presents the results of the work and evaluates its success.
- Chapter 7 summarises the work with some conclusions and future directions in which the presented work could be extended.

To facilitate the reading process, a list of acronyms and other abbreviations used in this document is provided in the Nomenclature.

Chapter 2

Background

2.1 Chrome Extensions

Chrome extensions are zipped bundles of HTML, CSS, JavaScript and other web technologies, that extend the functionality of the Chrome browser. An extension's architecture will vary depending on its functionality, but many extensions will include the following components:[2]

- **Manifest file:** The one file every extension must have. The manifest gives the browser information about the extension, such as the most important files and the permissions it requires.
- **Background script:** Background scripts are an extension's event handler; they contains listeners for specified browser events and lie dormant until an event is fired.
- **UI (User Interface) elements:** Most extensions have either a page action (within the address bar), or a browser action (in the toolbar), which when clicked open a HTML page.
- **Content scripts:** Content scripts can read and modify the DOM (Document Object Model) of web pages the browser visits.

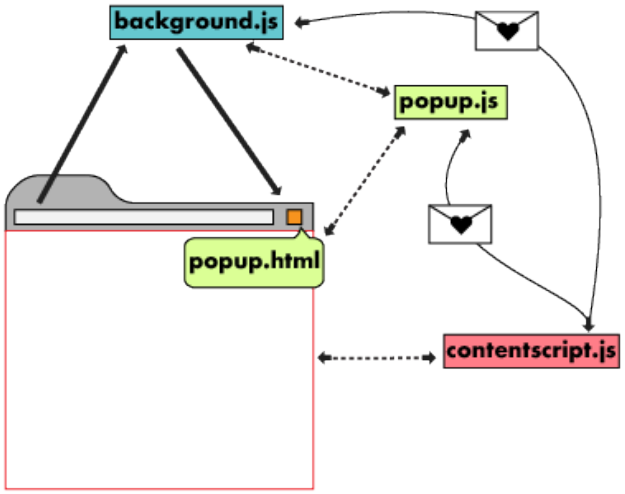


Fig. 2.1 An overview of Chrome extensions.[2]

2.2 Automatic Text Summarisation

Automatic text summarisation (ATS) is the process of producing a short and coherent summary from a large document, preserving its information content and overall meaning. The book "Automatic Text Summarization," provides six reasons why we need automatic text summarisation tools.[3]

1. Summaries reduce reading times.
2. When researching documents, summaries make the selection process easier.
3. Automatic summarisation improves the effectiveness of indexing.
4. Automatic summarisation algorithms are less biased than humans.
5. Personalised summaries are useful in question-answering systems as they provide personalised information.
6. Using automatic or semi-automatic summarisation systems enables commercial abstract services to increase the number of texts they are able to process.

Humans are generally good at this type of task since they are able to interpret the meaning of the document and then bring out the meaning and capture salient details in the new interpretation. Therefore, one can say the goal of ATS is to have the resulting summaries as good as those written by humans.

Evaluation of previous work

There has been plentiful research conducted in the field of ATS, with various approaches having been developed and applied in different domains. The approaches can be broadly divided into the following groups:[4]

- **Extractive summariser** - Involves selecting sentences from the document to make up the new summary.
- **Abstractive summariser** - Involves generating entirely new sentences to interpret the meaning of the document.

- **Single-document summarisation** - Produces a summary of a single input document.
- **Multi-document summarisation** - Produce a summary of multiple input documents.

- **Generic summary** - Summary simply gives important information from a document.
- **Focused summary** - Summary is produced in response to a user query.

Multi-document summarisation lies outside the scope of this project as the application will be employed per document (i.e. Single-document summarisation). Extractive summaries tend to give better results compared to abstractive summaries due to the fact that abstractive summarisation methods need to deal with more complex problems such as semantic representation, inference, and natural language generation.[5] Many approaches have been taken when it comes to extractive text summarisation, the four most extensively used methods are described in the following subsections.

Topic representation approach

The early work of ATS aimed to identify words that describe the topic of the input document. In his paper in 1958, Luhn used this approach to define the important words in a document as a combination of their frequency and their relative position in the sentence they appear in.[6]

Frequency-driven methods

A more advanced technique, which is utilised in many existing summarisation systems, is the Term Frequency-Inverse Document Frequency (TF-IDF) statistic. The TF-IDF value increases proportionally as the frequency of a word increases, but is offset by the frequency of the word in the corpus, which helps to control the fact that some words are more common than others.[7]

Graph-based techniques

Graph-based methods are influenced by Google's PageRank algorithm and they represent the document as a connected graph, with the sentences forming the vertices of the graph and the edges between them representing how similar the two sentences are.[8] The number of sentences a node is connected to, and the importance of those sentences affects its own importance and whether it is included in the summary or not.

Machine learning techniques

Machine learning approaches model the summarisation as a classification problem. A *naive-Bayes classifier* can be used to classify sentences based on whether they are summary-worthy or not.[9] Classifiers require training, and the training documents required are not always readily available.

Chapter 3

Design

3.1 Summariser

The summarisation algorithm utilised in this project will be an unsupervised graph-based extractive method based on the TextRank and LexRank algorithms.[10, 11] Both algorithms are derived from the PageRank algorithm and they essentially score sentences by their importance within a text.[12] The algorithms can be described as a three-step process with the following subsections describing these steps in detail.

3.1.1 Pre-processing

First, the text is pre-processed by removing stop words. Stop words refer to the most frequently occurring words in a language and are therefore deemed irrelevant. Once this is done, the remaining words are stemmed to reduce inflectional forms of a word to a common base form (e.g. car, cars, car's, cars' $\Rightarrow$ car).

3.1.2 Sentence ranking

A complete graph is then created where the vertices are the sentences of the text and the edges represent the similarity between the sentences. This is where TextRank and LexRank differ. TextRank implements a connected graph with edges between vertices representing the similarity between the corresponding sentences, no matter how similar they are, whereas LexRank implements an unweighted graph with edges existing between sentences only if the similarity is above a predefined threshold.[10, 11] This system goes with the TextRank approach and implements a fully connected graph. The similarity and ranking functions are described in further detail below.

Similarity

Sentence similarity in TextRank is defined simply as the number of terms overlapping between sentences. The approach taken in this system is more similar to that used in LexRank, which is the cosine similarity of tf-idf vectors. The tf-idf weight of a word increases proportionally with the number of times that word appears in the document but is offset by the frequency of the word in the corpus.[13]

The tf-idf weight is composed of two terms: the first computes the normalised Term Frequency (TF), the number of times a word appears in a document, divided by the total number of words in that document;

$$\{tf\}_{t,d} = \frac{f_{t,d}}{\sum f_d} \tag{3.1}$$

Where:
f_t is the frequency of term t in document d
$\sum f_d$ is the total number of terms in document d

The second term is the Inverse Document Frequency (IDF), computed as the logarithm of the total number of the documents in the corpus divided by the total number of documents where the specific term appears.

$$\{idf\}_{t,D} = \log \frac{N_D}{N_t} \tag{3.2}$$

Where:
N_D is the total number of documents in the corpus
N_t is the total number of documents including term t

This then gives the overall equation for the tf-idf weight for a term, t, as:

$$\{tf\text{-}idf\}_{t,d,D} = \{tf\}_{t,d} \times \{idf\}_{t,D} \tag{3.3}$$

Cosine Similarity will generate a metric that shows how related two vectors are by looking at the cosine of the angle between them rather than the magnitude.

$$\text{similarity} = \cos\theta = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} \tag{3.4}$$

Scoring

Once the graph has been constructed, each vertex in the graph is then ranked using the PageRank algorithm. Its central thesis is that a node is important if it is pointed to by other important nodes. Informally, it can be thought of as a model of user behaviour, where a surfer is surfing the web at random. The probability that the random surfer clicks on one link is solely given by the number of links on that page, which means that if the set of outgoing links from page u is O_u, the probability of going to each neighbour from j is $\frac{1}{|O_u|}$. The basic definition of the PageRank score can then be given as:[14]

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)} \tag{3.5}$$

Where:
$PR(u)$ is the PageRank of the current node u
B_u is the set of nodes having links to node u
$L(v)$ is the number of links present in node v

However, this has a problem where if a page does not have any outward links, the algorithm becomes stuck. To account for this, a damping factor d is introduced which is the probability to jump to any other node at each transition. This gives the final definition of the PageRank score as:

$$PR(u) = (1-d) \sum_{v \in B_u} \frac{PR(v)}{L(v)} \tag{3.6}$$

Where:
d is the damping factor and usually set to 0.85 [10]

3.1.3 Sentence selection

Once each sentence has been scored, the last step is to select the highest scoring sentences until the output summary reaches a desired length.

3.2 User Interface

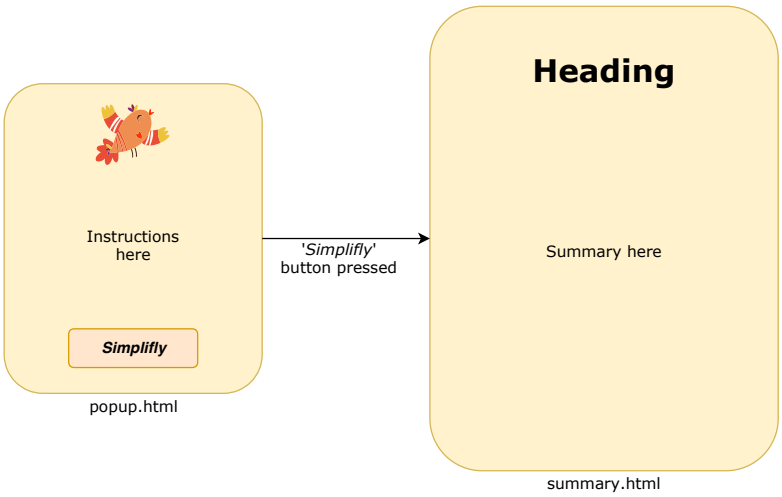


Fig. 3.1 A view of the extensions default pop-up screen and the summary screen.

The UI of the extension consists of a browser action (shown in Figure 3.2), which when clicked, displays the default pop-up screen showing the instructions on how to use the extension. The default pop-up will also contain a button which, once clicked, will switch the pop-up to the summary screen and display the summary of the currently visited article.

Fig. 3.2 The view of the extension in the browser environment.

Chapter 4

Implementation

4.1 Summariser

4.1.1 Architecture

Figure 4.1 shows the pipeline for the summarisation system built for this project. The *Sentence Extractor* split the input text using sentence tokenisation. The *Pre-processor* then removed stop words, and stemmed the sentences. The *Graph Builder* created a graph, with sentences as nodes and edges as the similarity between the sentences. The *Graph Builder* then scored the graph using PageRank, and returned an array of sentences with a score for each sentence. The *Sentence Selector* then picked the highest ranking sentences to create a summary.

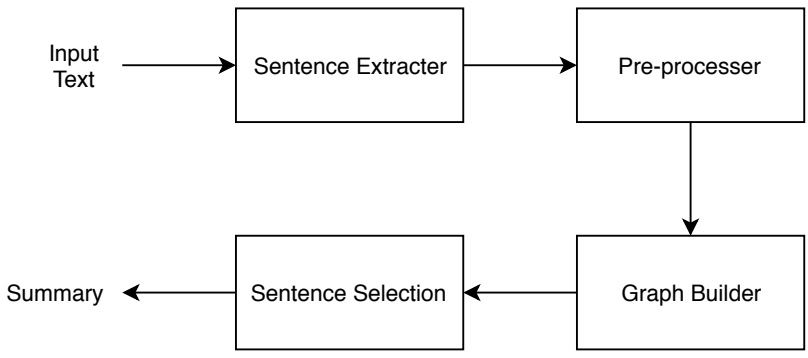


Fig. 4.1 An overview of the summarisation system showing all the individual components.

4.1.2 Sentence extractor

To represent text as a graph, with sentences as nodes, the text needs to be split into individual sentences. A naive approach would be to always split at periods, however, this would mean even single sentences like "The U.S.A" would be split into three sentences ("The U.", "S.", "A"). Libraries for this specific task exist, such as in Pythons NLTK, however, such libraries are unfortunately not available in the browser environment and instead regular expressions (regex) were used.

The pseudocode below describes the regex expression used in this project to split the text into sentences. In line 2, the replace function captures the following: {. or ? or !} followed by zero or more white spaces (\s*). This accounts for spaces following a punctuation mark which matches the English language grammar. The previous tokens only match if the next character is within the range A-Z (?=[A-Z]). Most English language sentences start with a capital letter. The matching tokens are then replaced with the pipe character 'l' and then in line 3, the text is then split along the pipes.

Algorithm 1 Sentence Tokeniser

```
1: function TOKENISE(text)                                  ▷ input text
2:     text ← string.replace(/([.?!])\s*(?=[A-Z])/g, "$1|")
3:     array ← text.split("|")
4:     return sentences                                     ▷ Array of sentences
```

4.1.3 Pre-processor

Next, the array of sentences from the extractor goes into the pre-processor where stop words are removed and the remaining words are stemmed. The stop words were removed by going through every word in every sentence and seeing if it was in the predefined list of stop words.[15]

Algorithm 2 Pre-processor 1

```
1: function REMOVESTOPWORDS(sentences, stopwords)
2:     for each sentence in sentences do
3:         sentence ← sentence.remove(stopwords)
4:     return sentences                    ▷ Array of sentences without stop words
```

Once the stop words were removed, the sentences were then passed into another function which stems all the words using the Porter Stemmer algorithm. This was done using a Node.js library 'porter.js'.[16]

Algorithm 3 Pre-processor 2

```
1: function STEMMER(sentences)
2:     for each sentence in sentences do
3:         sentence ← sentence.stem()
4:     return sentences                ▷ Array of stemmed sentences without stop words
```

4.1.4 Graph builder

It is now possible to build a graph from the pre-processed array of sentences. The graph was built by iterating through each sentence and the sentences adjacent to it and adding a directed edge in both directions with the value of the similarity between them. The number of adjacent edges to iterate through is called the search radius, *n*, and the TextRank reference paper recommends a value between 2 and 10.[10] An optimum value for the search radius is found in Chapter 6.

The tf-idf values of each sentence were calculated using the Node.js library 'node-tfidf' which returned an array of tf-idf values corresponding to the documents in the corpus.[17] The corpus used consisted of the article title, the article body, and a set of stemmed news articles from BBC News.[18] The cosine similarity between the two arrays was then calculated with the Node.js library 'cosine-similarity'.[19]

Algorithm 4 Graph Builder

```
1: function BUILDGRAPH(sentences)
2:     N ← number of sentences
3:     for i ← 0 to N do
4:         for j ← i to N do
5:             graph.addEdge(i, j, similarity(TFIDF(i), TFIDF(j)))
6:             graph.addEdge(j, i, similarity(TFIDF(i), TFIDF(j)))
7:     rankings ← graph.rank(0.85, 0.0001)      ▷ damping factor and threshold values
8:     return rankings                  ▷ Array of sentences with respective scores
```

Once the graph was built, the nodes were then ranked using the PageRank algorithm. This was done using the Node.js library 'pagerank.js' which returned an array of scores for the corresponding sentences.[20]

4.1.5 Sentence selector

After the sentences were scored with PageRank, the length of the goal summary is calculated. In this system, the summary length (T) is 20% of the original articles and must have a minimum size of 5 sentences. The array of sentences is sorted by scores and then the top T

sentences are kept. The array is then sorted by position in the original article and iterated over to produce the summary.

Algorithm 5 Sentence Selector

```
1:  function SELECTSENTENCES(scoredSentences)
2:      if Math.floor(sentences.length ÷ 5) < 5 then
3:          T ← 5
4:      else
5:          T ← Math.floor(sentences.length ÷ 5)          ▷ 20% of original size
6:      sentences ← sentences.sortByScore()               ▷ Sort by PageRank score
7:      sentences ← sentences.slice(0, T)                 ▷ Reduce array to size T
8:      sentences ← sentences.sortByPosition()            ▷ Sort by position in text
9:      summary ← ""
10:     count ← 0
11:     while count < sentences.length do
12:         summary ← summary + sentences(count)
13:         count ← count +1
14:     return summary
```

4.2 Extension

4.2.1 Build

All required libraries are installed using the npm package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js. Npm also allows us to access and install a range of available packages by browsing its registry; this may be useful for development. Webpack is a module bundler which handles all the applications assets and is used to bundle all the files in this project into a single zip file which is compatible with the Chrome web browser.

4.2.2 Permissions

To protect users private information, the extensions permissions have been limited to the critical information necessary to implement the extension. Therefore, the extension only has two permissions declared: activeTab, and storage. The activeTab permission will grant the extension temporary access to the currently active tab, only when the user invokes the extension. Access is cut off when the user navigates away from or closes the current tab. The storage permission allows the extension to store, retrieve, and track changes to user data.

4.2.3 Use case

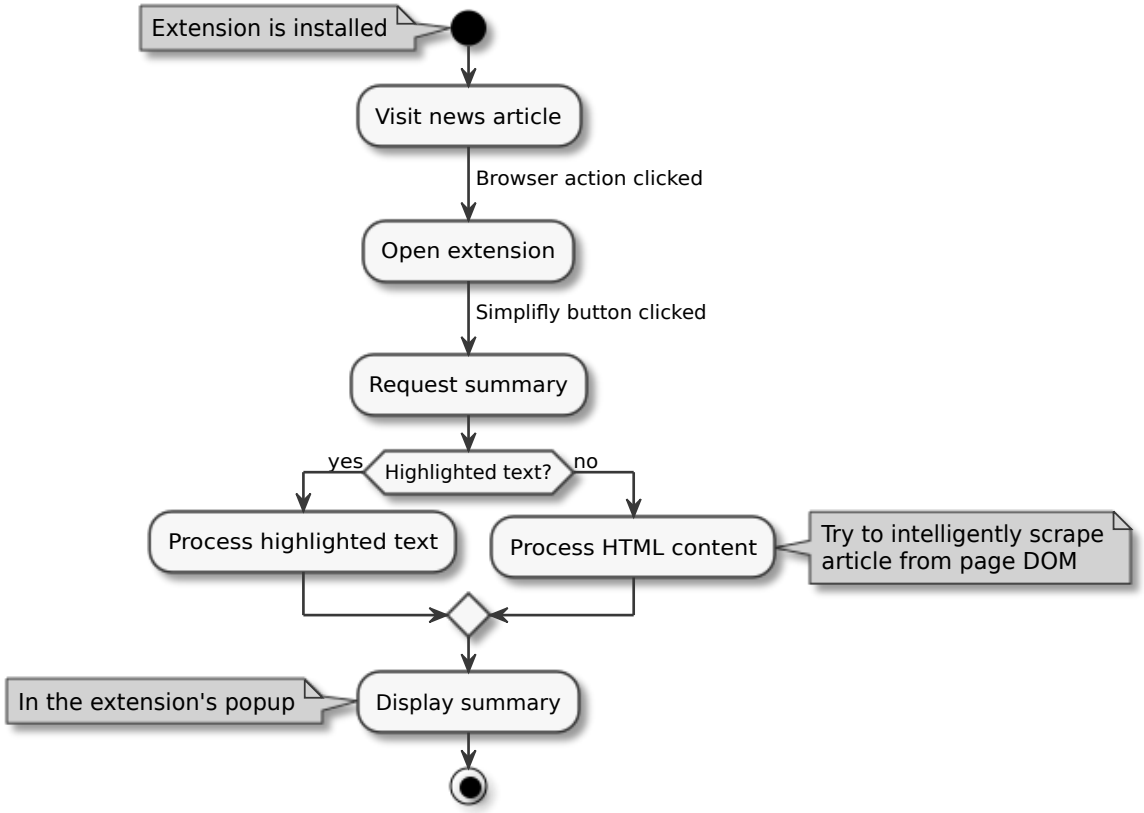


Fig. 4.2 Activity diagram for the summarisation of an online news article.

Figure 4.3 shows the sequence diagram for a typical use case. Once the user clicks the summarise button, an event is triggered and a message is passed to the content script requesting the currently active tab's content. The content script then checks to see if there is any text highlighted by the user. If there is then the content script grabs the highlighted content assuming it is the article's content and stores it along with the current tabs title in the browsers local storage as a JSON object.

If there is no text highlighted, then the content script tries to scrape the article using jQuery. This is done by selecting all <p> tags within the HTML source code not belonging to a class, i.e. <p class="foo"> would not be selected. This was done because it was found the majority of news websites store the contents of an article in classless <p> tags. It was also found that in the majority of cases the article's heading is always the first heading in the HTML source code, i.e. <h1>. Using this information, the article heading and its contents are extracted from the page and stored in the browsers local storage as a JSON object.

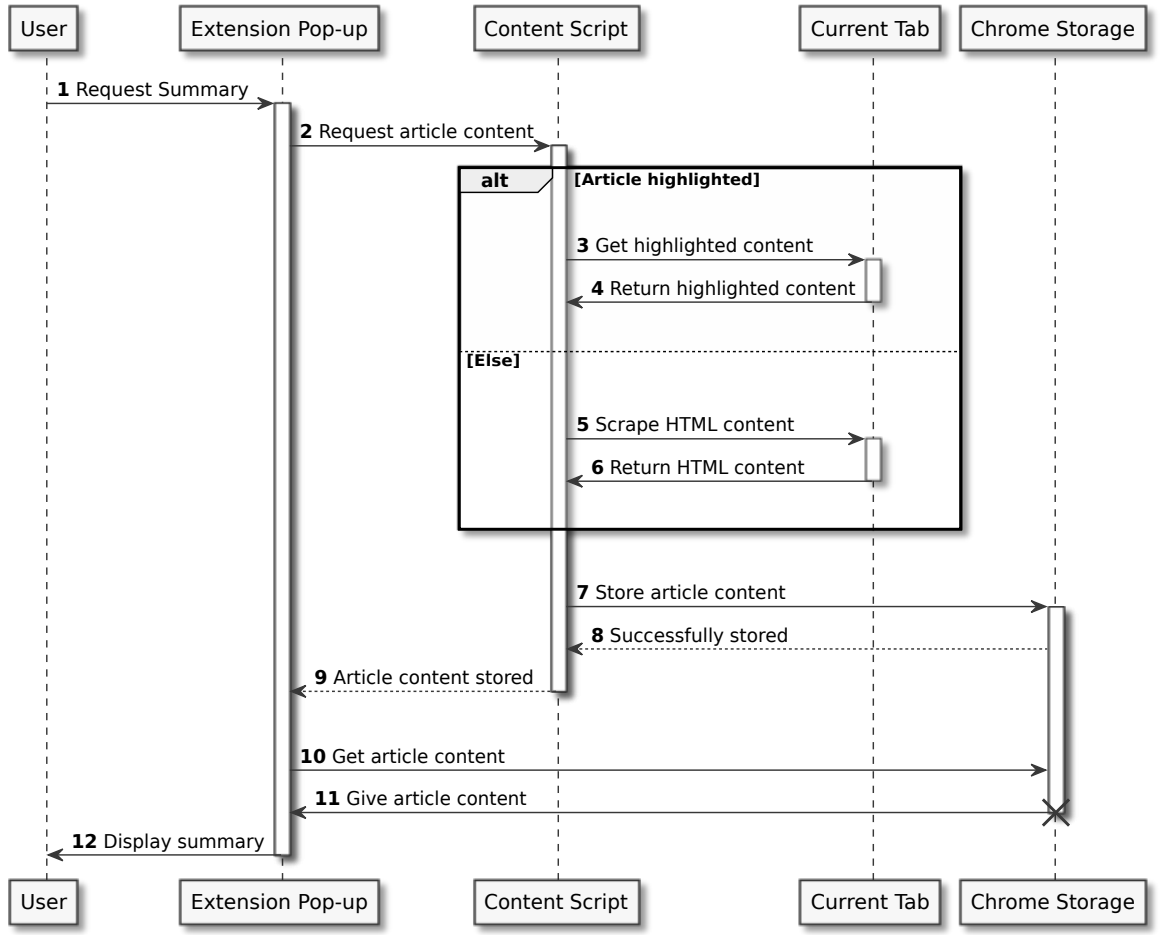


Fig. 4.3 Sequence diagram for the summarisation of an online news article.

Once the article is stored, the content script sends a message back to the pop-up informing it that the article is ready to be accessed. The pop-up then retrieves the information from the browser's local storage and clears the storage to avoid build up. The article's heading and its content are then passed into the summarisation function which returns a summary of the given article. The pop-up then opens a new window within itself displaying the summarised article.

4.2.4 Tests

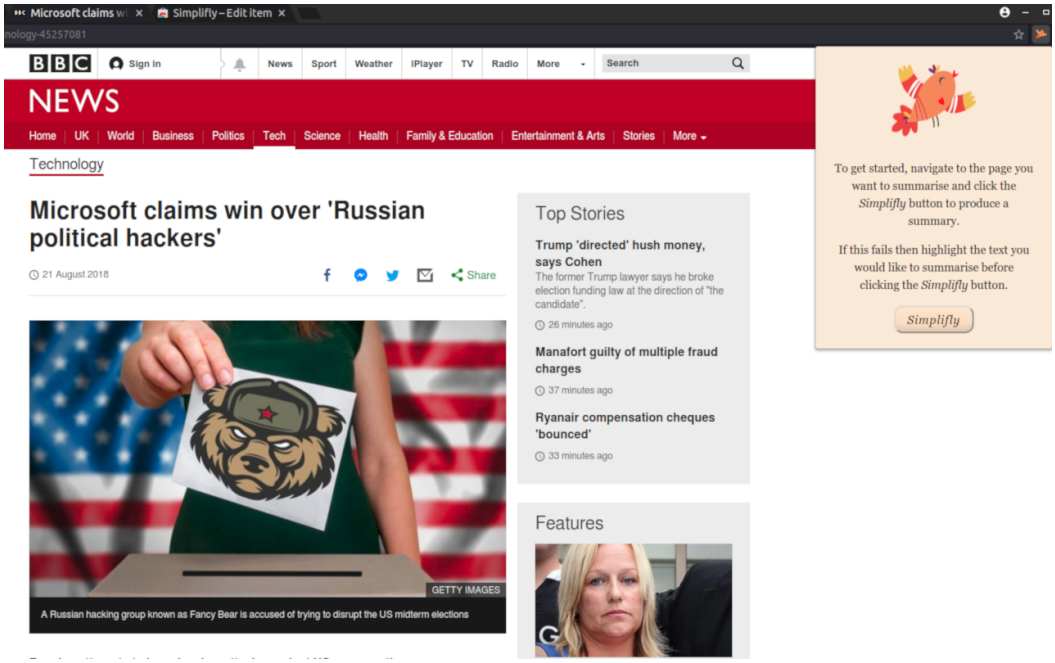


Fig. 4.4 Default pop-up screen.

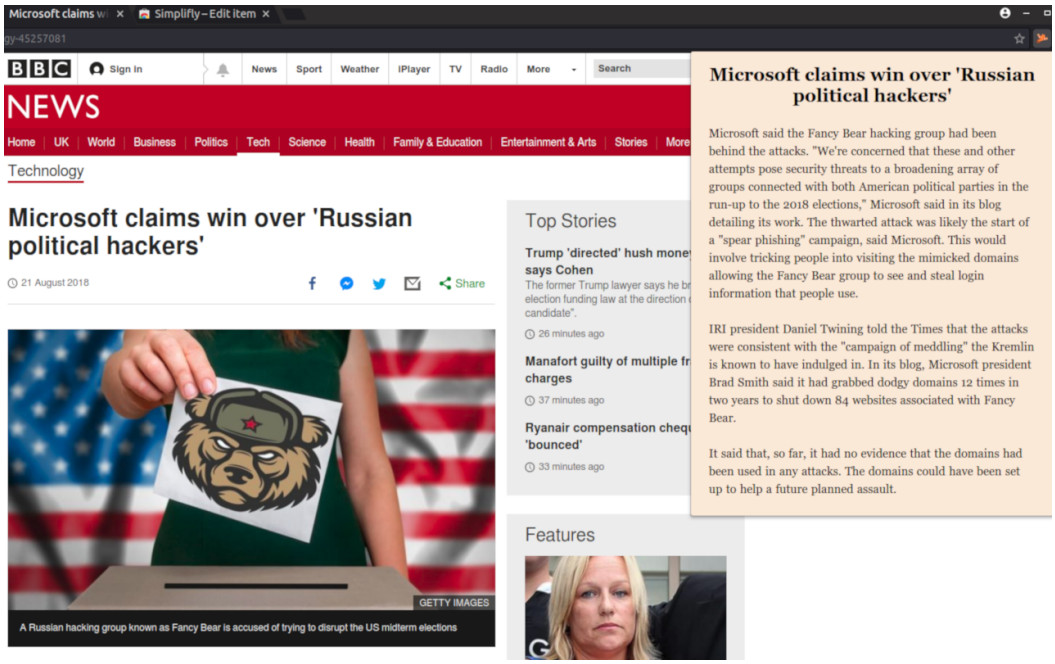


Fig. 4.5 Summary screen.

Chapter 5

Evaluation

Different values were tested for the search radius, n, of the algorithm to try and find an optimal value. The original paper describing TextRank recommends a value between 2 and 10, therefore values within this range are tested.[10] These values, along with two other popular summarisation systems called SMMRY, and Text Summarizer, are evaluated by comparing their output summaries with human (model) summaries using the ROUGE-N measure.[21, 22] This is explained in more detail throughout this chapter.

5.1 Data

The Document Understanding Conference (DUC) 2002 data-set for evaluation of ATS systems is ideal when it comes to testing since it contains evaluation documents with a set of corresponding gold-standard abstractive summaries from human judges.[23] This data-set, however, requires payment and was therefore not used in this evaluation. Instead, three individuals were asked to construct what they thought was the best summary for the following four news articles:

- Article 1: https://reut.rs/2wZQ3cf
- Article 2: https://bit.ly/2QbLPax
- Article 3: https://cnet.co/2CDkviC
- Article 4: https://bit.ly/2wXKlsu

5.2 Comparison

There are two different types of evaluation; intrinsic and extrinsic, and for this project, an intrinsic evaluation was chosen.[24] This is because intrinsic evaluation focuses on the performance of the summarisation system itself, and also the reference paper of TextRank also uses intrinsic evaluation. To assess the performance of each summariser, the set of summaries generated by them (system summaries), were compared to the human-generated abstract summaries (reference summaries).

Since the system summaries are in the form of extracts and the reference summaries are in the form of abstracts, the ROUGE-N metrics from the ROUGE metric set was used which makes it possible to compare extracts with abstracts. For this project, ROUGE-1, which refers to the overlap of unigrams between the system summary and reference summary, was used since it has shown to be a reasonable choice for extractive summarisation systems.[25]

5.2.1 Pre-processing

Before the ROUGE-1 scores were calculated, both the system summary and the reference summary were stemmed. This was because the system summaries were extractive, but the reference summaries were abstractive, meaning the same piece of information present in the system summary might also be present in the reference summary but with a different inflexion. The stop words were also removed from both summaries to allow for a more accurate scoring of the system summaries produced.

5.2.2 Measures

ROUGE-N uses three measures; recall, precision, and F_1 score.

Recall

Recall, in the context of ROUGE, refers to how much of the reference summary is the system summary recovering/capturing. For ROUGE-1, it is computed as:

$$\text{Recall} = \frac{(\text{words in reference summary}) \cap (\text{words in system summary})}{\text{words in reference summary}} \tag{5.1}$$

Precision

Precision, in the context of ROUGE, refers to how much of the system summary is relevant/required. For ROUGE-1, it is computed as:

$$\text{Precision} = \frac{(\text{words in reference summary}) \cap (\text{words in system summary})}{\text{words in system summary}} \tag{5.2}$$

F_1 score

The F_1 score is the harmonic mean between precision and recall, with the purpose to get a weighted average of the two. The F_1 score is defined as:

$$F_1 = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \tag{5.3}$$

Upper bound of the ROUGE scores

Since the reference summaries will be abstractive, it will be impossible to achieve a perfect score as all the system summaries will be extractive. An experiment on the DUC 2002 data-set in 2012 aimed to calculate the upper bound for ROUGE-1 scores of extractive summarisation systems. This was done by selecting sentences from the reference summaries and then picking the sentences in the source text that most resembled those sentence to build a system summary. This experiment gave an idea of the largest possible score extractive summarisation systems can achieve. The upper bound F_1 score was calculated as 0.5657.[26]

Chapter 6

Results

The following tables show the performance of the summarisation systems in the four test articles, and Table 6.5 giving the overall average scores across all four tests.

System	Recall	Precision	F_1
Simplifly, $n = 2$	**0.7208**	**0.4149**	**0.5267**
Simplifly, $n = 4$	**0.7208**	**0.4149**	**0.5267**
Simplifly, $n = 6$	**0.7208**	**0.4149**	**0.5267**
Simplifly, $n = 8$	0.8000	0.3590	0.4956
Simplifly, $n = 10$	0.8000	0.3590	0.4956
SMMRY	0.8545	0.3780	0.5241
Text Summarizer	0.6108	0.3685	0.4597

Table 6.1 The average ROUGE-1 scores for Article 1

System	Recall	Precision	F_1
Simplifly, $n = 2$	**0.4195**	**0.4216**	**0.4206**
Simplifly, $n = 4$	0.4343	0.3397	0.3812
Simplifly, $n = 6$	0.3937	0.3408	0.3653
Simplifly, $n = 8$	0.3082	0.3083	0.3082
Simplifly, $n = 10$	0.2561	0.3114	0.2811
SMMRY	0.3535	0.2546	0.2960
Text Summarizer	0.2897	0.3891	0.3321

Table 6.2 The average ROUGE-1 scores for Article 2

System	Recall	Precision	F_1
Simplify, $n = 2$	0.2808	0.4128	0.3342
Simplify, $n = 4$	0.3350	0.3822	0.3570
Simplify, $n = 6$	0.2371	0.3563	0.2847
Simplify, $n = 8$	**0.3893**	**0.5599**	**0.4592**
Simplify, $n = 10$	0.2825	0.4675	0.3522
SMMRY	0.4872	0.4297	0.4566
Text Summarizer	0.3251	0.4557	0.3795

Table 6.3 The average ROUGE-1 scores for Article 3

System	Recall	Precision	F_1
Simplify, $n = 2$	0.6096	0.4602	0.5245
Simplify, $n = 4$	0.4750	0.4749	0.4750
Simplify, $n = 6$	0.4750	0.4749	0.4750
Simplify, $n = 8$	0.4750	0.4749	0.4750
Simplify, $n = 10$	0.5427	0.5223	0.5323
SMMRY	0.7054	0.4404	0.5423
Text Summarizer	**0.6336**	**0.6131**	**0.6232**

Table 6.4 The average ROUGE-1 scores for Article 4

System	Recall	Precision	F_1
Simplify, $n = 2$	0.5077	0.4274	0.4515
Simplify, $n = 4$	0.4913	0.4029	0.4350
Simplify, $n = 6$	0.4567	0.3967	0.4129
Simplify, $n = 8$	0.4931	0.4255	0.4345
Simplify, $n = 10$	0.4703	0.4150	0.4153
SMMRY	**0.6002**	**0.3757**	**0.4548**
Text Summarizer	0.4648	0.4566	0.4486

Table 6.5 The average ROUGE-1 scores across all articles

Discussion

The proposed algorithm in this paper was shown to perform best when the search radius, n, was set to 2. The best summariser overall was the SMMRY summariser which had a recall that was 0.0925 higher than the proposed system, precision was actually lower than the proposed system by 0.0517, and the F_1 score was 0.0033 higher than the proposed system.

The results show that the proposed system is comparable to competing systems, with it having an overall F_1 score only 0.0033 lower than the highest ranking system. Looking at the results, it is clear this is due to the low recall scores, implying that the proposed system misses out on information deemed important by the reference summaries. However, it is worth noting that human summaries are susceptible to bias. The proposed system does, however, have the highest precision scores, implying that most of the information it does contain is actually relevant information. With such a small data-set, however, it is difficult to consider these results accurate but they do give a good indication of potential scores the systems could receive.

As mentioned in Section 5.2.2, there is an upper bound to the ROUGE-N score since the reference summaries are abstracts whereas the system summaries are extracts, so a perfect score will be impossible to obtain. However, the score is still useful when comparing different algorithms, especially when running them on the DUC 2002 data-set.

Ethics

There were no ethical issues directly involved in the research during this project. However, the possibility of text becoming misleading is a risk since important points could be excluded from a text if the summarisation is done without human supervision, which may change the entire meaning of a text to something other what the author intended, spreading incorrect information.

Chapter 7

Conclusion

The Google Chrome extension designed in this paper allows users to quickly generate summaries of the news articles they visit. Although it has been designed with English articles in mind, the extension can be extended to work with any language. While this system overcomes issues with existing summarisation extensions, such as the inability to produce a summary without first highlighting the text to summarise, it still suffers the same problems with splitting sentences correctly. The performance of the underlying summarisation system itself was comparable to competing systems and real-time performance is a lot faster as it produces summaries almost instantly.

7.1 Future Work

Pre-processing

A relevant work for the future would be to investigate the pre-processing more thoroughly this project only utilised stop words and stemming for pre-processing. Other pre-processing steps such as lemmatisation rather than stemming and also using PoS (Part of Speech) filters could also impact the results.

Evaluation

A major drawback to this project was the lack of a large data-set to test the systems. In future, it would be interesting to see the results if a data-set such as DUC 2002 was used to evaluate the system summaries.

New technologies

Having to operate in the browser environment meant the system was limited in what it could use in terms of libraries. However, recently Google has announced TensorFlow.js, which is designed to allow users to define, train, and run machine learning models entirely in the browser. This is a major development as it means machine learning techniques can be used in future systems which would definitely solve some major problems such as incorrect sentence tokenisation.

References

[1] Chin Yew Lin. "ROUGE: a Package for Automatic Evaluation of Summaries". In: July 2004. URL: https://www.microsoft.com/en-us/research/publication/rouge-a-package-for-automatic-evaluation-of-summaries/.

[2] Google. *What are extensions?* Online. [Accessed: 05-08-2018]. URL: https://developer.chrome.com/extensions.

[3] Juan Manuel Torres-Moreno. *Automatic Text Summarization*. First. Cognitive Science and Knowledge Management. Wiley-ISTE, 2014.

[4] D Jurafsky and J Martin. *Speech and Language Processing An Introduction to Natural Language Processing*. Second. Computational Linguistics, and Speech Recognition. Pearson, 2008.

[5] Mehdi Allahyari et al. "Text Summarization Techniques: A Brief Survey." In: *CoRR* abs/1707.02268 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#AllahyariPASTGK17.

[6] H. P. Luhn. "The Automatic Creation of Literature Abstracts". In: *IBM Journal of Research and Development* 2.2 (Apr. 1958), pp. 159–165. ISSN: 0018-8646. DOI: 10.1147/rd.22.0159.

[7] Juan Ramos. *Using TF-IDF to Determine Word Relevance in Document Queries*. 1999. URL: https://www.bibsonomy.org/bibtex/2e757decbf13ecf55dee0b5eae56f9ccd/reynares.e.

[8] Amy N. Langville and Carl D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ, USA: Princeton University Press, 2006. ISBN: 0691122024.

[9] Julian Kupiec, Jan Pedersen, and Francine Chen. "A Trainable Document Summarizer". In: *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '95. Seattle, Washington, USA: ACM, 1995, pp. 68–73. ISBN: 0-89791-714-6. DOI: 10.1145/215206.215333. URL: http://doi.acm.org/10.1145/215206.215333.

[10] R. Mihalcea and P. Tarau. "TextRank: Bringing Order into Texts". In: *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelonaand Spain, July 2004. URL: https://www.bibsonomy.org/bibtex/20d96cef99428b30c4a5ac67241b34e45/lee_peck.

[11] Günes Erkan and Dragomir R. Radev. "LexRank: Graph-based Lexical Centrality As Salience in Text Summarization". In: *J. Artif. Int. Res.* 22.1 (Dec. 2004), pp. 457–479. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622487.1622501.

[12] L. Page et al. “The PageRank citation ranking: Bringing order to the Web”. In: *Proceedings of the 7th International World Wide Web Conference*. Brisbane, Australia, 1998, pp. 161–172. URL: citeseer.nj.nec.com/page98pagerank.html.

[13] Cambridge University Press. *Tf-idf weighting*. Online. [Accessed: 28-08-2018]. URL: https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html.

[14] Kaushal Kumar. “PageRank algorithm and its variations: A Survey report”. In: 14 (Jan. 2013), pp. 38–45.

[15] Onix Text Retrieval Toolkit. *Stop Word List 1*. Online. [Accessed: 08-08-2018]. URL: http://www.lextek.com/manuals/onix/stopwords1.html.

[16] jedp. *porter.js*. Online. [Accessed: 28-08-2018]. URL: https://github.com/jedp/porter-stemmer.

[17] duyetdev. *node-tfidf*. Online. [Accessed: 09-08-2018]. URL: https://github.com/duyetdev/node-tfidf#readme.

[18] Derek Greene and Pádraig Cunningham. “Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering”. In: *Proc. 23rd International Conference on Machine learning (ICML’06)*. ACM Press, 2006, pp. 377–384.

[19] kgryte. *Cosine Similarity*. Online. [Accessed: 09-08-2018]. URL: https://github.com/compute-io/cosine-similarity#readme.

[20] alixaxel. *pagerank.js*. Online. [Accessed: 10-08-2018]. URL: https://github.com/alixaxel/pagerank.js/.

[21] SMMRY. *SMMRY*. Online. [Accessed: 30-08-2018]. URL: https://smmry.com/.

[22] Text Summarization. *Text Summarizer*. Online. [Accessed: 30-08-2018]. URL: http://textsummarization.net/text-summarizer.

[23] NIST. *DUC 2002 GUIDELINES*. Online. [Accessed: 30-08-2018]. URL: https://www-nlpir.nist.gov/projects/duc/guidelines/2002.html.

[24] Inderjeet Mani. *Summarization Evaluation: An Overview*. 2001. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.2078.

[25] Chin-Yew Lin. “Looking for a few good metrics: ROUGE and its evaluation”. In: (Sept. 2018).

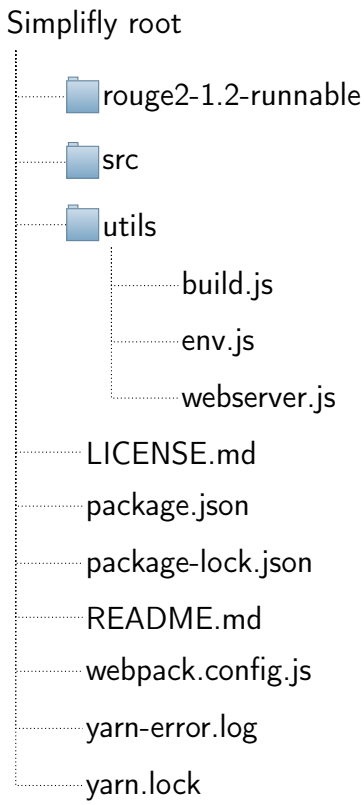
[26] Jonatan Bengtsson and Christoffer Skeppstedt. “Automatic extractive single document summarization: An unsupervised approach”. MA thesis. Göteborg, Sweden: Chalmers University of Technology, 2012.

Appendix A

Git Repository

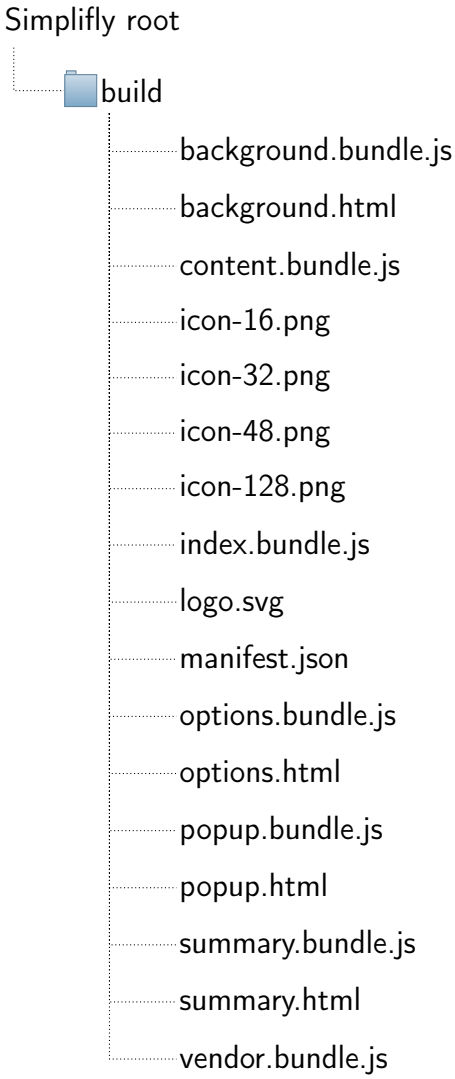
https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2017/ixa444

Structure



The README.md file contains a link to the Chrome Web Store to download the extension.

https://chrome.google.com/webstore/detail/simplifly/jioeflccmbeaeelfklklmbdjcbdeccnh



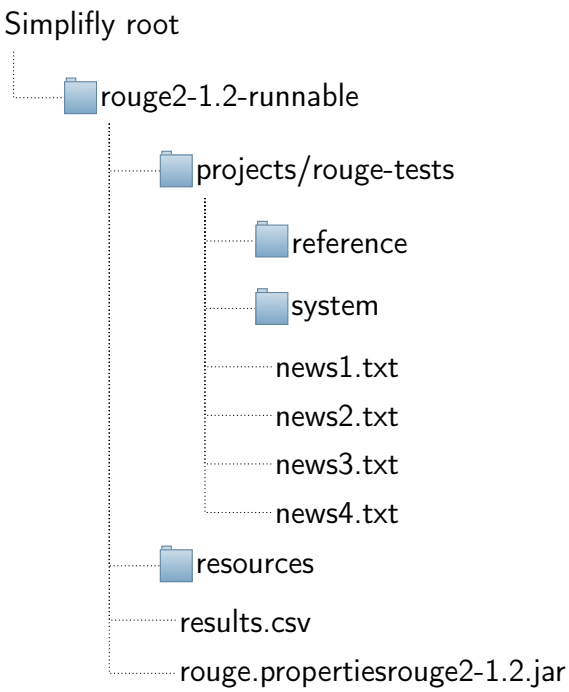
Build folder is the web-packed form of the extension and is used for testing. When uploading to the Chrome Web Store, the contents of the build folder are zipped and uploaded.

The build folder and its contents are created from the command:

```
$ npm run build
```

For testing purposes, it was nice to have the build folder update automatically whenever a change occurred in the source rather than having keep running the previous command. This was done by starting a local web-server using the following command:

```
$ npm start
```



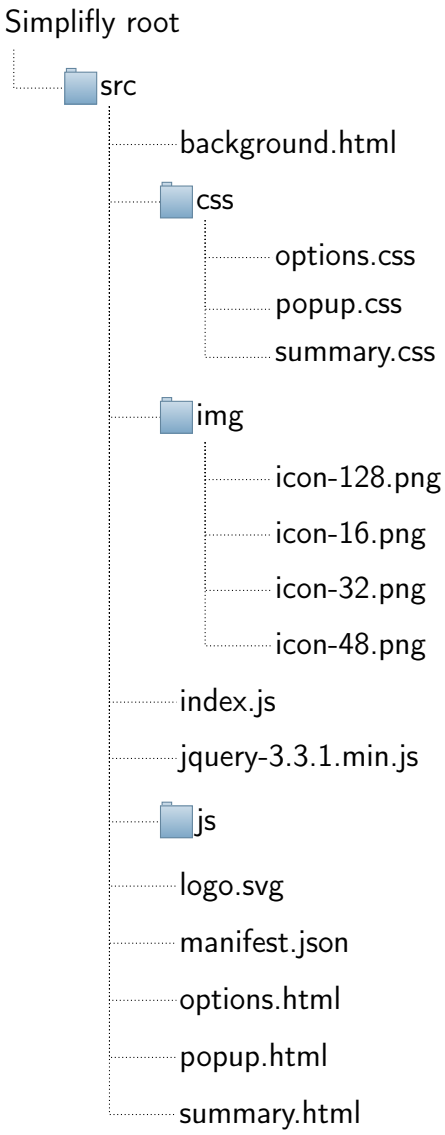
This directory contains all the information required to run the ROUGE automatic evaluation. The 'reference' directory contains reference (model) summaries to compare the system summaries to. The 'system' directory contains all the summaries generated by the different systems.

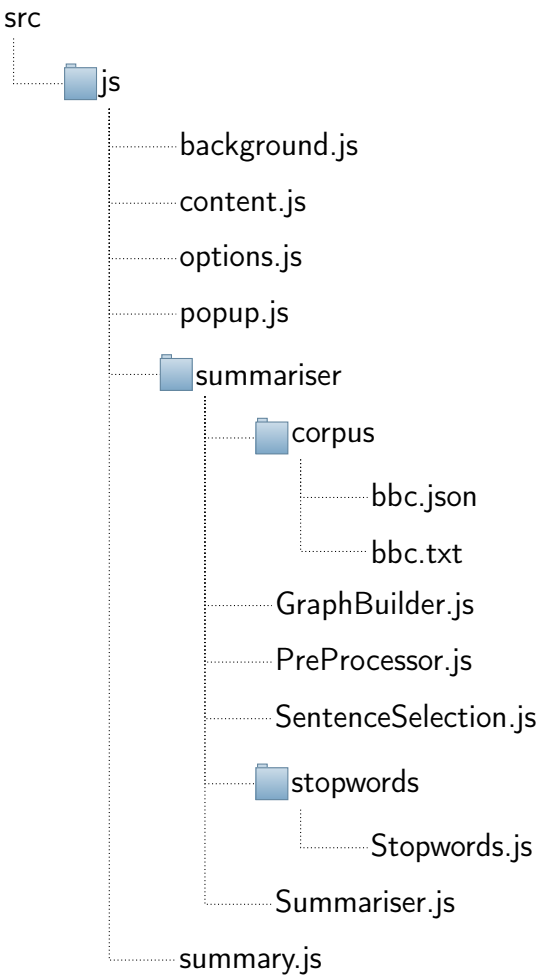
Systems are numbered as follows:

1. Simplify, $n = 2$
2. Simplify, $n = 4$
3. Simplify, $n = 6$
4. Simplify, $n = 8$
5. Simplify, $n = 10$
6. SMMRY, https://smmry.com/
7. Text Summarizer, http://textsummarization.net/text-summarizer

To run ROUGE and generate results.csv, run the following command within the rouge2-1.2-runnable directory:

```
$ java -jar rouge2-1.2.jar
```


Simplifly root
src
background.html
css
options.css
popup.css
summary.css
img
icon-128.png
icon-16.png
icon-32.png
icon-48.png
index.js
jquery-3.3.1.min.js
js
logo.svg
manifest.json
options.html
popup.html
summary.html


src
js
background.js
content.js
options.js
popup.js
summariser
corpus
bbc.json
bbc.txt
GraphBuilder.js
PreProcessor.js
SentenceSelection.js
stopwords
Stopwords.js
Summariser.js
summary.js

Appendix B

Project Proposal

A Web Extension to Summarise News Articles

Ishmael Aqsar, 1416344

Aim
To produce a web extension to summarise online articles with the click of a button.

Plan

1. Begin research. (1 weeks)
   (a) Look at past summarising techniques, comparing the pros and cons of each.
   (b) Look at writing up a draft literature review.
   (c) Look at what tools will be needed to develop the actual summarising algorithms.
   (d) Look at how web extensions are developed.
2. Requirements Analysis and Design (1 week)
   (a) Look at functional and non-functional requirements of the system.
   (b) Begin planning the structure and behaviour of the code using UML diagrams.
   (c) Begin planning the structure and behaviour of the code using UML diagrams.
   (d) Design Prototypes
3. Begin working on the back-end of the extension. (4 weeks)
   (a) Start writing the code needed for the core of the extension.
   (b) Make notes of the work as it progresses.
   (c) Test the code often.
4. Begin working on the front-end of the extension. (2 weeks)
   (a) Start writing the code needed for the actual browser extension.
   (b) Make notes of the work as it progresses.
   (c) Test the code often.
5. Finalise the extension. (2 weeks)
   (a) Test the extension thoroughly to make sure there are no flaws.
6. Begin working on the report. (2 weeks)
   (a) Include any references used.