## Information Retrieval System

### Design choices

One of the requirements for the project was to assess the performance of summarizers on CNN or Dailymail articles. Ultimately we concentrated on the Dailymail articles for this project because the Rouge evaluation needs a human written summary as the reference text, and Dailymail articles included bullet points summarizing the keypoints of an article. Using these bullet points meant that we could fully automate the evaluation process. For CNN articles hand-written reference summaries would’ve had to be written.

### How it’s built

The information retrieval system (IR) for this project was made using Python packages ”beautifulsoup4” and ”requests”. The module defines the methods needed for retrieving and parsing the information for further processing. An URL of an article or path to a html file can be given as input and the method creates beautifulsoup object out of it. A method was created for parsing the the article and finding the bullet points. The method returns any bullet points it finds as a list. Another method was created for parsing the article content itself and returning the text as a string object.

## Graphical User Interface walk-through

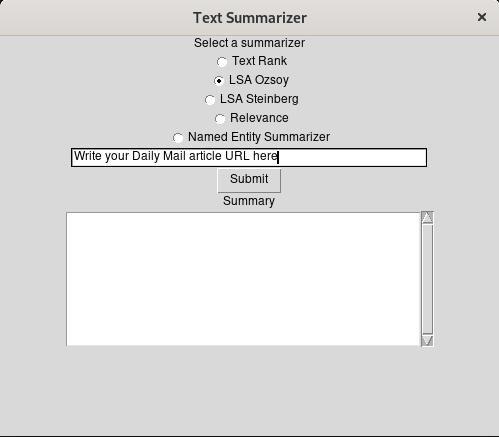


Figure 1: Graphical User Interface input

The graphical user interface (GUI) takes three inputs from the user. First input is radio button between the five different summarizers. The radio buttons are below ”Select a summarizer” help text, and user can select only one of the summarizers for each evaluation.

The second input is the Uniform Resource Locator (URL) field, which defaults to help text ”Write your Daily Mail article URL here”. The URL must not include quotation marks or any other special characters at the start of the text or at the end, and the URL must be a Dailymail URL. There is no input sanitization implemented for this version of the program, so the user must take care writing the URL.

The last user input is clicking the ”Submit” button. This button launches the program calculating the rouge scores and generates the summary. Under the said button there is a text label ”Summary” and under it there is a scrollable text field where the generated summary will appear.

Figure 2: Graphical User Interface output

After the user has pressed ”Submit” button, the program fetches the article using its IR system, creates a summary of the article contents using selected summarizer and calculates the Rouge values using the bullet points found in the article. Pressing the button then displays the output in two fields under the button. First output is the Rouge recall and precision values in percentages for both Rouge-2 and Rouge-3. This field is only revealed after calculating the values. The generated summary will be inserted in the previously blank text field. The Rouge values field and the summary field both clean up themselves upon pressing the button again and the user can try out the same article with all of the summarizer.

### Future development

Multi-linguality or even cross-linguality are important features, but left out of the scope for lack of time to implement them, This version of the program uses only english language but other languages are supported by the packages used with some tuning; future version could have drop down list from which the user can choose the language that an article uses. An option to automatically detect the language could be used as well, and user could select which language they want the output in for cross-linguality interface.

This version of the program focuses heavily on Dailymail, so in a future version the IR system would have an automatic detection functionality to determine which site is being processed, and based on the site, decide which methods to use for parsing.

## Program performance metrics

All the speed performance metrics were gathered using cProfile module and all the memory usage metrics were gathered using ”memory profiler” module. For all the testing one article was used (URL: <https://www.dailymail.co.uk/news/article-8911489/Polls-close-battleground-Florida-Georgia-Biden-takes-Vermont-Trump-takes-Kentucky.html>).

### Overall performance

The overall program running time was measured using cProfile and the times include every single operation done by the program including the gui and the backend processes. The program was opened, then the URL was entered as an input and the summarization process was ran. This excludes the time taken by the user inputs. Below results of the summarizers used in the project.

|  |  |
| --- | --- |
| Summarizer | Finishing time (seconds) |
| LSA (Ozsoy) | 13 |
| LSA Steinberg | 11 |
| Named Entity | 14 |
| Text Rank | 17 |
| Relevancy based summarizer | 10 |

The relevancy based summarizer performed the best averaging 10 seconds, and the summarizer performed the worst averaging 17 seconds.

Speed comparison was made against three different existing article summarizers, that are hosted on web. The web applications are Resoomer (<https://resoomer.com/en/>), Esummarizer (<http://esummarizer.com/main/summarize>), and SMMRY (<https://smmry.com/>). While there is no exhaustive documentation on how the web hosted summarizer applications are built, these results can be used as indication on how well the project’s program performs.The same article, that was used for testing the project program, was also used for the existing summarizers. The article was entered, and timer was started upon launching the summarization progress. Below results of the existing web hosted summarizers.

|  |  |
| --- | --- |
| Application name | Finishing time (seconds) |
| Resoomer | 16 |
| SMMRY | 2 |
| Esummarizer | 3 |

Resoomer performed by far the worst and roughly as well as text rank summarizer and named entity summarizer. The two other summarizers performed almost five times better than any of the summarizers used in the project.

### Information retrieval

Speed and performance evaluation for the IR were tested using the aforementioned URL to an article. The article was retrieved five times and the average was calculated.

On average, the IR system takes 0,8 seconds to perform all the operations, which means opening the article, parsing the article content and finding the bullet points. Most time is spent loading the article from the site, which takes 0,5 seconds on average, and the rest of the time is used for the actual parsing. Both of the parsing methods iterate over the article once finding all the needed data meaning this process could be enhanced only by using a faster alternative to beautifulsoup package.

Memory usage is completely dependant on size the article, and for this article the html site was 3,8 megabytes and the beautifulsoup object was 2,4 megabytes, and in total this uses 6,4 megabytes worth of memory. Article content parsing and finding the bullet points don’t increment or decrement the value.

### Summarizers

Considering only the methods used by the summarizer we get the following results

|  |  |
| --- | --- |
| Summarizer | Total time (seconds) |
| LSA (Ozsoy) | 0.195 |
| LSA (Steinberg) | 0.165 |
| Text Rank | 0.9 |
| Relevance based summarizer | 0.17 |
| Named Entity | 1.09 |

The test shows that the Steinberg LSA summarizer performed the best and the named entity summarizer performed the worst. Both LSA summarizers and the relevance based summarizer took roughly fifth of the time that named entity summarizer and text rank summarizers used.

As named entity summarizer was made for the project it’s taken into a closer inspection. The other summarizers were only used in the project without any modifications in the code and assumed to work optimally.

The cProfile results show that the whole process takes 1.092 seconds and the most time is spent in preprocessing the text namely in the ”list\_named\_entities” method which takes 0.87 seconds. Further inspecting the code shows that the data is only iterated over only once, which means there is no needless redundancy.

### Rouge evaluation speed

The rouge score depends on the lenght of the summary and the amount of the bullet points in the article meaning that the speed of the rouge evaluation is different between articles and the summarizers. The results below were gathered using the same article that has been used in the other tests.

|  |  |
| --- | --- |
| Summarizer | Time taken (seconds) |
| LSA (Ozsoy) | 1.95 |
| LSA (Steinberg) | 1.865 |
| Text rank | 1.993 |
| Relevance based summarizer | 1.712 |
| Named Entity | 1.8 |

As the table shows there is fairly little difference between using the different summarizers and the rouge evaluation for the summaries generated takes 1.864 seconds on average, and the longest time is spent in the function ”ngram\_creation\_and\_counting” which is called twice and uses 0.6 seconds per call. Inspecting the function reveals that it uses two nested for-loops and calls the method ”lower” in the nested loop. This means the ”lower” method 2(xy) times where x is the number of reference tuples and y is the number of summary tuples (VIITTAUS KOODIIN: RougeEvaluation.py moduuli, metodi ngram\_creation\_and\_counting). For this method optimization could be done but otherwise the rouge evaluation seems to perform well, and this would only cause issues with very large amount of articles.

### Memory usage

The memory usage is dependant on the size of the article. For the article considered in these tests, the memory usage for fetching the article with the IR system was 3.9 Mb and the soup object generated from it required 2.4 Mb. Creating the summaries did not increment the amount of memory except the named entity summarizer needed 2.5 Mb more memory for the summary and reference tuples it creates. The rouge evaluation incremented the memory usage by roughly 3 Mb. In total the memory usage was incremented by 11.8 Mb or 9.3 Mb across the program depending if the named entity summarizer was used.

### Conclusion on performances

Considering the use of the graphical user interface is to perform the task on a single article the performance speed is satisfactory. The backend modules are fast enough to challenge the web applications considered above. Out of the methods, only the named entity summarizer’s data preprocessing method could be enhanced slightly. Overall the breakdown of the components suggests that the program performs fairly well and should rather need a different environment to be ran in such as part of web application.