# Project 18: Automatic text summarization 1

https://github.com/tHerttua/nlpProject

Jasmin Al Amir Teemu Herttua Armi Korhonen Niina Mäkinen

***Abstract*—Automatic text summarization is a tool to compress the main information of a document to a smaller space. A common approach is to extract the most important sentences from the original text using some kind of scoring algorithm. In this work we use the open-source summarization module PyTLDR and design a new algorithm based on named entities. The algorithms are evaluated using Daily Mail news articles and common ROUGE-N scoring. Our method matches and even exceeds the performance of the algorithms implemented in PyTLDR, especially in terms of precision.**

## I. Introduction

With the rapid growth of news articles and other texts on the internet, finding relevant information becomes more and more laborious. Summarization of documents has a growing demand to allow spending less time reading and searching for suited articles. Manually summarizing documents is slow and inefficient, which calls for tools for automatizing the task.

Automatic text summarization (ATS) is used to compress an input document to a shorter version which still contains the relevant information of the original text. Luhn [1] introduced the concept in 1958. He used word frequency and distribution to derive measures of significance for words and sentences. In 1969, Edmundson [2] combined the frequency approach with cue words, and also considered title and heading words and sentence location. It was concluded that using only statistical information is insufficient; semantic characteristics of language need to be taken account of.

More advanced techniques where e.g. relations between sentences and are considered were later developed, and machine learning approaches started gaining interest when the amount of available training data increased [3]. Current catalog of methods is extensive as the field has become more popular. A summary of the pros and cons of the current methods can be found in Tables 1 and 2 of El-Kassas et al. [4]. Many ATS systems utilize multiple methods, which allows the use of their respective strengths and eliminates their shortcomings.

Two approaches for ATS exist: extractive and abstractive. In extractive methods, the most important sentences in the input text are used for the summary. Abstractive methods generate the summary using words and sentences which are not in the original document. Hybrid approaches have also been used [4]. Since the abstractive approach is more complex, most of the research has been focusing on extractive methods.

The basic workflow of extractive summarization starts with forming a suitable representation of the input text. This is dependent of the extraction method, and can mean e.g. a graph or a matrix representation of the words and sentences. The sentences are then scored using some algorithm, and the sentences with the highest scores are extracted. The task is then to form the algorithm, and to design the representation type. Some post-processing of the sentences can optionally be done after extraction.

ATS is a challenging task, and current methods are not able to produce summaries similar to those written by humans [4]. Extractive techniques can produce incoherent summaries, where sentences seem disconnected. Using natural language processing is needed for producing more human-like summaries.

Coherence of text can be enhanced e.g. with respect to named entity. It refers to an abstract or physical object, which can be assigned a name. Recognizing named entities from documents can be difficult, and it is mostly approached using supervised machine learning techniques [5]. Several open-access tools have been developed for the task.

Developing of automated systems is dependent of an appropriate evaluation metric. Standard evaluation techniques used in ATS compare automatically and manually produced summaries and describe their similarity. While this is a valid approach, there is not necessarily one single way of making a good summary. The evaluation of the goodness of a summary can be subjective, and finding a functioning metric can be challenging. A common choice is to use the ROUGE framework [6], which has been shown to correlate well with the opinions of human evaluators [7].

PyTLDR[1] is an open-source Python module used for extractive document summarization. It includes three implementations of ATS: a TextRank based, a Latent Semantic Analysis based and a sentence relevance score based summarizer. In this work we evaluate the summarization algorithms provided by the PyTLDR module. We also propose a new algorithm based on recognizing named entities using the recognition framework in the spaCy[2] module. We use articles from the Daily Mail website[3] and the standard ROUGE-N evaluation metric to asses the performance of the methods. We develop a Graphical User Interface (GUI) for easy utilization of the software, with a focus on the Daily Mail website and its articles.

[1]https://github.com/jaijuneja/PyTLDR
[2]https://github.com/explosion/spaCy
[3]https://dailymail.co.uk

This report is organized as follows: In Section II the methods implemented in this work are described. The software and workflow are presented in Section III. The results and performance metrics are presented in Section IV and further discussed in Section V. Section VI contains the concluding remarks.

## II. Methods

### A. TextRank

TextRank [8] is a graph-based ranking model for text processing. It is a generalization of PageRank used to score sentences based on their relevance. It is fully unsupervized and does not require training corpora.

The idea behind graph-based ranking is to decide the importance of a vertex in a graph based on the votes cast by other vertices. The votes are links between vertices, and the more links to a vertex are cast, the more important it is. The score $S$ of a vertex $V_i$ is defined as [9]

$$S(V_i) = (1-d) + d \sum_{j \in In(V_i)} \frac{S(V_j)}{|Out(V_j)|}, \quad (1)$$

where $0 < d < 1$ is a damping factor, $In(V_i)$ denotes the set of vertices pointing to $V_i$ and $Out(V_j)$ denotes the set of vertices $V_j$ points to. In TextRank, the strength of the connection between vertices $V_i$ and $V_j$ is included by using edge weights $w_{ji}$. Weighted score $WS(V_i)$ is defined as

$$WS(V_i) = (1-d) + d \sum_{V_j \in In(V_i)} \frac{w_{ji} WS(V_j)}{\sum_{V_k \in Out(V_j)} w_{jk}}. \quad (2)$$

The TextRank algorithm is iterative, starting with random values assigned to the vertices and iterating until convergence.

The vertices in a graph are complete sentences, and the highest-scoring ones are used in the summary. The vote casting between sentences is based on their similarity, e.g. word overlap. The overlap-based similarity of two sentences $S_i$ and $S_j$, defined as sets of words $w$, is [8]

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{log(|S_i|) + log(|S_j|)}. \quad (3)$$

The numerator is a normalization factor ensuring that long sentences are not favored. Other similarity methods can also be used.

The main steps of implementing graph-based algorithms in natural language processing are (from Mihalcea and Tarau [8])

1) Identify text units that best define the task at hand, and add them as vertices in the graph.
2) Identify relations that connect such text units, and use these relations to draw edges between vertices in the graph. Edges can be directed or undirected, weighted or unweighted.
3) Iterate the graph-based ranking algorithm until convergence.
4) Sort vertices based on their final score. Use the values attached to each vertex for ranking/selection decisions.

### B. Latent Semantic Analysis

Latent Semantic Analysis applies Singular Value Decomposition (SVD) to the text summarization task (e.g. [10], [11], [12], [13]). The SVD of a $n \times m$ matrix $\mathbf{A}$ is defined as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \quad (4)$$

where $\mathbf{U}$ is a $m \times m$ unitary matrix, $\mathbf{\Sigma}$ is a $m \times n$ matrix and $\mathbf{V}$ is a $n \times n$ unitary matrix. The elements of $\mathbf{\Sigma}$, placed on the diagonal, are called singular values of $\mathbf{A}$. The columns of $\mathbf{U}$ and $\mathbf{V}$ are the left- and right-singular vectors of $\mathbf{A}$. The SVD provides a way to express $\mathbf{A}$ as a sum of its constituents:

$$A = \sum_{k=1}^{l} \sigma_k u_k v_k^T, \quad (5)$$

where $l = min\{n, m\}$, $\sigma_k$ is the $k$th singular value and $u_k$ and $v_k$ are $k$th columns of $\mathbf{U}$ and $\mathbf{V}$, respectively. The magnitude of each singular value indicates the relevance of each part of the sum, and since they are ordered, SVD can be used to obtain the components in order of importance.

In case of text analyzing, the matrix $\mathbf{A}$ is formed such that columns represent sentences and rows are words or phrases. The matrix entries can be e.g. frequency of word in corresponding sentence or Tf-idf [11]. SVD captures patterns, and singular vectors can then be thought as concepts in the text and singular values as the relevance of each concept [13].

Gong et al. [13] suggested that the rows of the matrix $\mathbf{V}^T$ consists of these concepts and the columns are sentences. Each matrix entry denotes an index, which indicates how well the sentence describes the corresponding concept. In their ATS approach, a predefined number of important concepts was chosen. Then, the sentence with the highest index with respect to each concept was extracted for the summary.

Steinberger and Jezek [12] pointed out that the above method leaves out relevant sentences, which have a high but not the highest index. They also noted that the number of sentences one wants to include in the summary is the same as the number of extracted concepts, and the relevance of the concepts decreases with the number of sentences. They proposed a modified method, where a vector $\mathbf{s}$ with components

$$s_k = \sqrt{\sum_{i=1}^{r} \sigma_i^2 v_{k,i}^2}, \quad (6)$$

where $r$ is the dimension of the new space and $v_k$ is a sentence vector in $\mathbf{V}$, is calculated. In essence, sentence vectors are weighted with their corresponding singular value, and their length is calculated. The sentences with the highest $s_k$ are extracted for the summary. The parameter $r$ is set such that only the dimensions whose singular value is over some predefined threshold, usually $0.5\sigma_{max}$, are included.

Sentences can belong to more than one main concept, which causes noise to the $\mathbf{V}^T$ matrix. Ozsoy et al. [10] proposed the Cross method as an extension to the approach of Steinberger and Jezek to overcome this issue. They added a pre-processing step for the $\mathbf{V}^T$ matrix, in which the average sentence score is

calculated for each topic. The cells with an index value equal or less than the average score are set to zero. The algorithm then proceeds essentially identically to the steps in Steinberger and Jezek.

PyTLDR implements the sentence length method by Steinberger and Jezek (hereafter referenced as Steinberger LSA for convenience) and the Cross method.

### C. Relevance sentence scoring

A simple method for extracting sentences is to score them by relevance, and then choose the highest scoring ones. In the relevance scoring algorithm described by Gong et al. [13], the document is decomposed into sentences. A weighted term-frequency vector is created for the document and for each individual sentence, and the inner products of the sentences with the whole document are calculated. The highest scoring sentence is extracted for the summary, and the terms it contains are removed from the document. The procedure is repeated until the pre-defined number of sentences is attained. The workflow is as follows (from Gong et al. [13]):

1) Decompose the document into sentences and form the candidate sentence set $S$.
2) Create weighted term-frequency vector$A_i$ for each sentence in $S$ and a weighted term-frequency vector $D$ for the whole document.
3) Compute the inner product between $D$ and each $A_i$.
4) Add the highest scoring sentence to the summary.
5) Delete the highest scoring sentence from $S$, and eliminate all the terms in the sentence from the document. Recompute $D$.
6) If the number of sentences in the summary has reached the pre-defined value, terminate. Otherwise go to Step 3.

### D. Named entity scoring

The new text summarization algorithm we developed uses the Python library spaCy for named entity recognition and the library TextBlob[4] to extract the sentences from the original article. The spaCy model used for named entity recognition is en_core_web_sm, which is an English multi-task CNN trained on OntoNotes. In our implementation, the named entities used to create the summary are those labeled as a person or an organization. The number of extracted sentences is given as user input.

Named entities are searched from the document text and added onto a list. If a named entity appears in the document text n times, it is also added onto the list n times. The input document is decomposed into a list of sentences. All the sentences are evaluated against the list of the named entities. When a named entity appears in a sentence, the score of that sentence will increase by 1 with each appearance. As a named entity can appear multiple times in the list, the appearance of that particular entity in a sentence gives a higher overall score for that sentence. This is deliberate, since if a named entity appears several times in the document, it is assumed that it holds more importance compared to the other named entities.

The highest scoring sentences are added into the summary in the order in which they appear in the document as not to disrupt any possible continuation flow in the text.

### E. ROUGE-N

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [6] is the most widely used evaluation tool in the field of ATS [4]. It is used to determine the quality of an automatically produced summary by comparing it to manually produced, "ideal" versions.

Several types of ROUGE measures exist. We use ROUGE-N, which is defined as the n-gram (sequence of n adjacent words in the text) recall. ROUGE-N recall $R$ between the generated and one reference summary is calculated as

$$R = \frac{N_{overlap}}{N_{reference}}, \tag{7}$$

where $N_{overlap}$ is the number of overlapping n-grams in the evaluated and reference summaries, and $N_{reference}$ is the number of n-grams in the reference summary. It describes how much of the reference summary is captured in the automatically produced version.

The downside of recall is that summaries can achieve higher scores by simply increasing the length. This can lead to redundant words and sentences, which is counterproductive to the purpose of summarizing. It is then necessary to include precision $P$ of the summary

$$P = \frac{N_{overlap}}{N_{generated}}, \tag{8}$$

where $N_{generated}$ is the number of n-grams in the generated summary. Precision and recall can be expressed jointly using the traditional F-score

$$F = \frac{2PR}{P + R}, \tag{9}$$

which is the harmonic mean of the two.

## III. Software

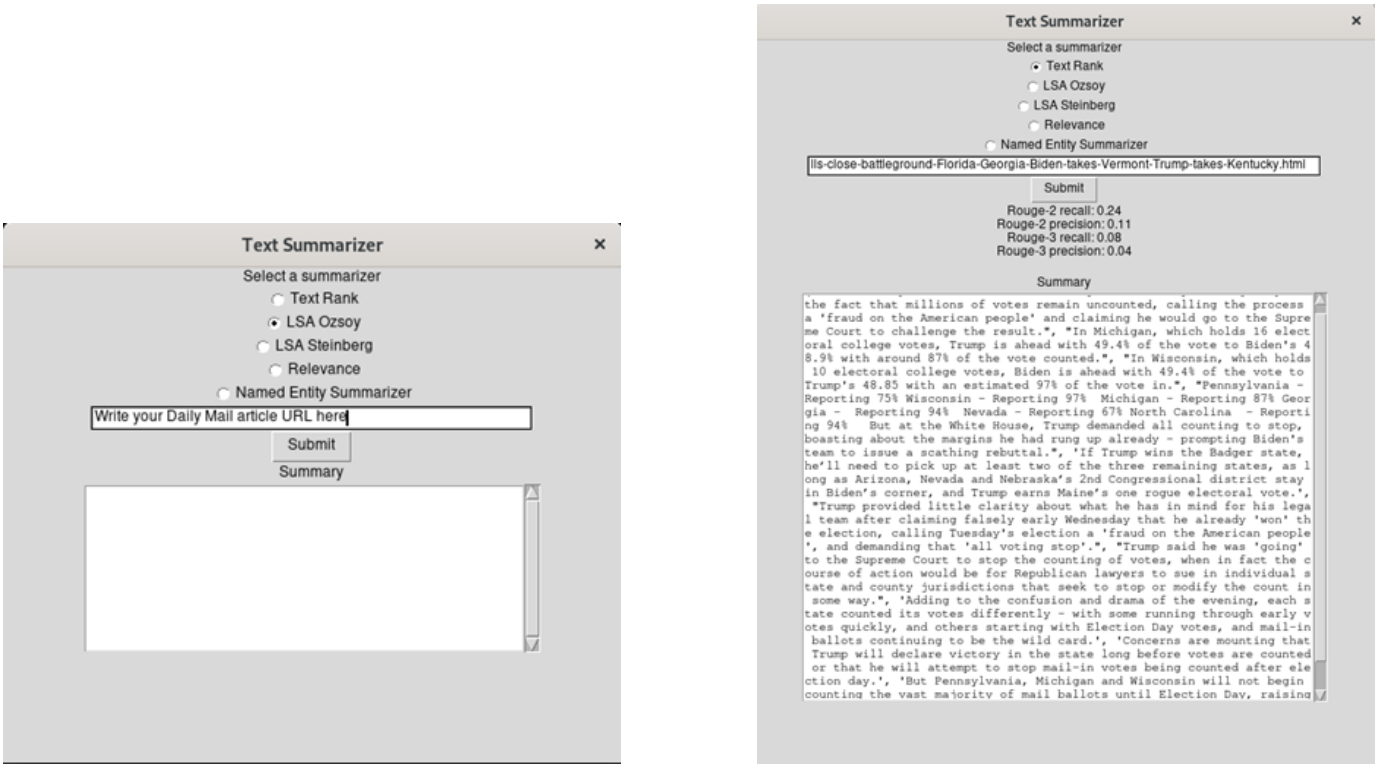
### A. Information retrieval

The information retrieval (IR) system for this program was made using the Python packages beautifulsoup4[5] and requests[6]. The module defines the methods needed for retrieving and parsing the information for further processing. A URL of a Daily Mail article or a path to a HTML file can be given as input and the method creates a beautifulsoup object from 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.

[4]https://textblob.readthedocs.io/en/dev/

[5]https://pypi.org/project/beautifulsoup4/

[6]https://pypi.org/project/requests/




(a) Blank view.

(b) Example input and output.

Fig. 1. The GUI for summarizers.

### B. Graphical user interface

Screenshots from the GUI for implementing the summarizers are shown in Fig. 1. The program takes three inputs from the user. First input is a 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 URL field, which defaults to help text ”Write your Daily Mail article URL here”. The URL can 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 Daily Mail URL. There is no input parsing implemented, so the user must take care writing the URL.

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

After the user has pressed ”Submit” button, the program fetches the article using its IR system, creates a summary of the article contents using the selected summarizer and calculates the ROUGE scores 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 upon pressing the ”Submit” button again.

## IV. Evaluation

### A. Data set

We use news articles retrieved from the Daily Mail website as data for the evaluation. The Daily Mail articles commonly start with a small summary in the form of bullet points. We use this bullet point summary as the human-made reference summary needed for the ROUGE-N scores.

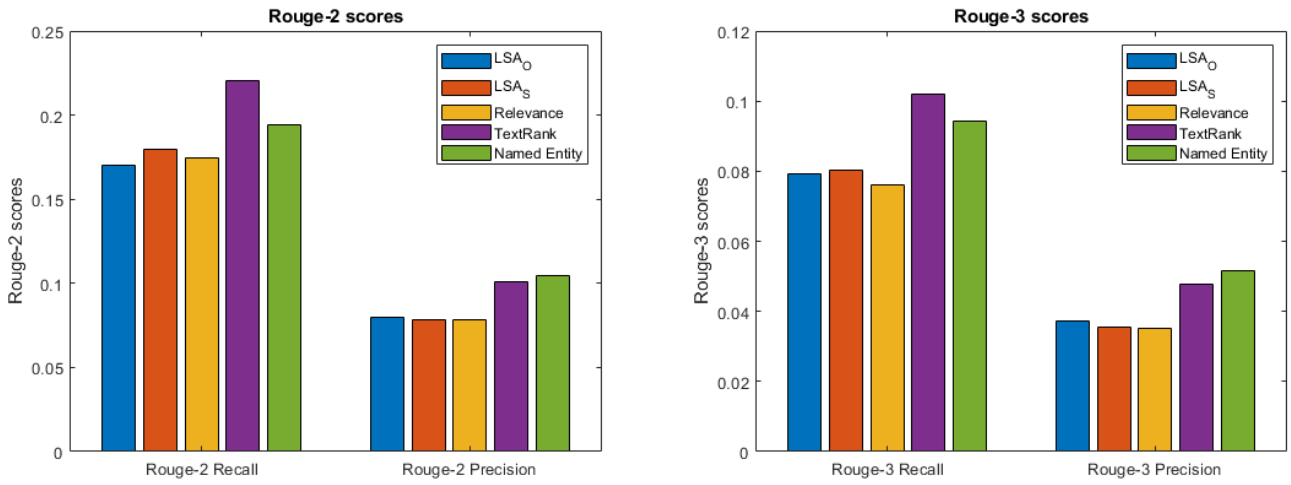
For the automatic processing of the data set, we developed a script that creates the summaries using both PyTLDR and the newly developed named entity summarizer. This process bypassed the use of the GUI for faster processing of large amount of data. The script also measures the time each algorithm took to produce the summaries.

We tested 73 different documents fetched from the Daily Mail website. The data set used was extracted from the website manually. The links to the documents were stored in a text document and automatically processed with the Python script.

When testing the algorithms, we created summaries with an equal amount of sentences as the human-made reference summaries.

### B. ROUGE scores

The means of the ROUGE-2 and ROUGE-3 scores from the 73 generated summaries are plotted in Fig. 2. The LSA




(a) (b)

Fig. 2. The ROUGE-2 and ROUGE-3 scores of the implemented summarizers. $\text{LSA}_O$ refers to the Cross method, and $\text{LSA}_S$ to the Steinberger version.

approaches and relevance scoring have virtually the same scores. TextRank performs the best in terms of ROUGE recall. The named entity scoring method seems to have the best precision, but the difference between named entity and TextRank is very small.

The F-measures of the algorithms calculated from the ROUGE scores seen in Fig. 2 are seen in Fig. 3. The named entity algorithm has the highest measures, but only with a narrow margin to the TextRank algorithm. The Cross method performs slightly better than the Steinberger LSA. Relevance scoring algorithm has the lowest performance, but the margin to the Steinberger LSA is narrow especially with ROUGE-2.

Overall, precision scores are quite low indicating that there is some redundancy in the produced summaries. This is also seen in the lengths of the reference and produced summaries. The mean length of the reference summaries was 76 words, and the mean length of the produced summaries was 165 words. There was not a large variation between the different algorithms in terms of amount of words in the produced summary.

The reference summaries used for the articles were bullet point lists, which are usually very compact. Full sentences tend to be longer and have more fillers, and extractive methods collect them as such from the articles. This could lead to low precision of the summaries. Since longer sentences in the generated summaries likely contain more fillers than the short bullet point references, the rather low recalls can also be partly explained by the same effect. Since this is likely a data set related issue, it does not affect how the methods compare against each other.

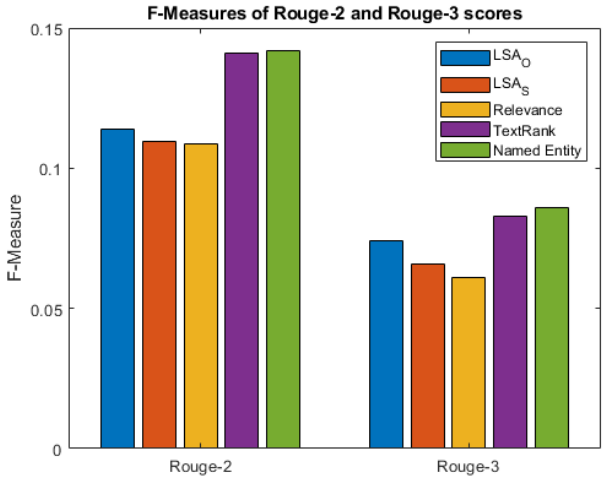



Fig. 3. F-measures of the summarizer algorithms. $\text{LSA}_O$ refers to the Cross method, and $\text{LSA}_S$ to the Steinberger version.

### C. Speed performance

All the speed performance metrics were gathered using the cProfile module and all the memory usage metrics were gathered using the memory profiler module. One article[7] was used for all performance testing.

*1) 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. This includes 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

[7]https://www.dailymail.co.uk/news/article-8911489/Polls-close-battleground-Florida-Georgia-Biden-takes-Vermont-Trump-takes-Kentucky.html

TABLE I
RUNTIMES OF THE OVERALL PROGRAM USING DIFFERENT SUMMARIZING ALGORITHMS.

| Algorithm | Runtime |
|---|---|
| TextRank | 17s |
| LSA (Steinberg) | 11s |
| LSA (Ozsoy) | 13s |
| Relevance score | 10s |
| Named entity | 14s |

once, finding all the needed data. Hence this process could be enhanced only by using a faster alternative to the beautifulsoup package.

*2) Graphical user interace:* The overall program running time was measured using cProfile. 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. The running times of the program using the five different summarizers are collected in Table I. The relevance scoring summarizer performed the best averaging 10 seconds, and the TextRank 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[8], Esummarizer[9], and SMMRY[10]. 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 program presented here performs. The same article that was used for testing our program was also used for the existing summarizers. The article was entered, and timer was started upon launching the summarization progress. The results of the existing web hosted summarizers are collected in Table II.

Resoomer performed by far the worst, roughly the same as our program when using TextRank summarizer and named entity summarizer. The two other summarizers performed almost five times better than any of the summarizers used in our software.

[8]https://resoomer.com/en/
[9]http://esummarizer.com/main/summarize
[10]https://smmry.com/

TABLE II
RUNTIMES OF EXISTING SUMMARIZERS HOSTED ON THE WEB.

| Application | Runtime |
|---|---|
| Resoomer | 16s |
| SMMRY | 2s |
| Esummarizer | 3s |

TABLE III
RUNTIMES OF THE SUMMARIZING ALGORITHMS.

| Algorithm | Runtime |
|---|---|
| TextRank | 0.900s |
| LSA (Steinberg) | 0.165s |
| LSA (Ozsoy) | 0.195s |
| Relevance score | 0.170s |
| Named entity | 1.090s |

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. 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 a part of a web application.

*3) Algorithms:* The performance of the algorithms was also tested separately, i.e. without the GUI. Runtimes of the algorithms are collected in Table III. The test shows that the Steinberger LSA 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 TextRank summarizers used.

As the named entity summarizer was developed for the project it is taken into a closer inspection. The other summarizers were only used in the project without any modifications in the code and are 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 method which creates the list of named entities in the document. The method takes 0.87 seconds. Further inspecting the code shows that the data is iterated over only once, which means there is no needless redundancy.

*4) Memory usage:* The memory usage is dependent 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. The soup object generated from the article required 2.4 Mb. Creating the summaries did not increment the amount of memory, except in case of the named entity summarizer which needed 2.5 Mb more memory for the summary and reference tuples it creates. The ROUGE-N 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.

## V. DISCUSSION

The automatic generation of precise, relevant and coherent summaries is a prominent problem in natural language processing. Methods for the task have been developed since the 1950s. Due to the explosive growth of information on the internet, the need for such systems is continuously growing.

Extractive techniques are the most popular choice. A variety of methods have been developed, and the existing algorithms are occasionally revamped and extended by other researchers. Problems concerning e.g. coherence of text and the scoring heuristics are still yet to overcome.

We have developed a software for automatically summarizing documents using various extractive approaches. We utilize the Python PyTLDR module and its TextRank, Latent Semantic Analysis and relevance scoring methods. As an extension, we have developed a new algorithm based on named entities. The named entity summarizer scores sentences based on the appearance of keywords (the named entities) in them.

The summarizing methods were evaluated using ROUGE-2 and ROUGE-3 scores and news articles from the Daily Mail website. The scores were low overall, but this could be explained by the fact that the reference summaries were bullet point lists and not meant to be coherent text as such. Our named entity algorithm exceeded all the algorithms in terms of ROUGE precision. The TextRank had higher recall scores, but only slightly. From the PyTLDR algorithms, TextRank performed the best both in terms of precision and recall. LSA and relevance based algorithms had similar performance compared to each other.

Performance evaluation showed that the methods which scored well in terms of ROUGE-N scores were also the slowest to compute. Our named entity method was the slowest, with a small margin to the TextRank. The LSA and relevance based algorithms were the fastest. As the named entity summarizing algorithm is rather inefficient in comparison to the other algorithms, it should be optimized first before any future use. We did not consider optimization to be of the upmost importance and prioritized creating a working algorithm we could test against the other algorithms.

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

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

The named entity approach could be especially suited for summarizing documents concerning e.g. businesses. In case of successful optimization, it could be useful as part of a product aimed at companies who need to quickly find out what is being written about their product or brand on the Internet.

## VI. Conclusions

We have developed a software for automatic document summarization. In addition to using an existing Python module and its methods, we developed a new extractive summarization algorithm based on named entities. Our algorithm outperforms most of the pre-existing methods in terms of ROUGE-N scores, but the performance comes at the cost of a slightly higher runtime. The software is built around the Daily Mail website, but could easily be extended to function on arbitrary sites.

## References


[1] H. P. Luhn, "The Automatic Creation of Literature Abstracts," in IBM Journal of Research and Development, vol. 2, no. 2, pp. 159-165, Apr. 1958

[2] H. P. Edmundson, "New Methods in Automatic Extracting," Journal of the ACM, vol. 16, no. 2 oo. 264-285, Apr. 1969

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

[4] W. S. El-Kassas, C. R. Salama, A. A. Rafea, H. K. Mohamed, "Automatic Text Summarization: A Comprehensive Survey," Expert Systems with Applications, vol. 165, 113679, Mar. 2020

[5] R. Jiang, R. E. Banchs, and H. Li, "Evaluating and combining name entity recognition systems," In Proceedings of the Sixth Named Entity Workshop, pp. 21-27, Aug. 2016

[6] C. Y. Lin, "Rouge: A package for automatic evaluation of summaries," in Text summarization branches out, pp. 74-81, Jul. 2004

[7] R. Sun, Z. Wang, Y. Ren, and D. Ji, "Query-Biased Multi-document Abstractive Summarization via Submodular Maximization Using Event Guidance," International Conference on Web-Age Information Management, 2016, pp. 310–322

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

[9] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," Computer Networks and ISDN Systems, vol. 30, pp. 107-117, 1998

[10] M. Ozsoy, I. Cicekli, and F. Alpaslan, "Text summarization of turkish texts using latent semantic analysis," In Proceedings of the 23rd international conference on computational linguistics (Coling 2010), pp. 869-876, Aug. 2010

[11] M.G. Ozsoy, F. N. Alpaslan, I. Cicekli, "Text summarization using Latent Semantic Analysis," in Journal of Information Science, vol. 37, no. 4, pp. 405-417, Jun. 2011

[12] J. Steinberger, K. Jezek, "Using latent semantic analysis in text summarization and summary evaluation," Proc. ISIM, vol. 4, pp. 93-100, 2004

[13] Y. Gong, X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 19-25, Sept. 2001

[14] Y. Mao, L. Liu, Q. Zhu, X. Ren, and J. Han, "Facet-Aware Evaluation for Extractive Text Summarization," arXiv, arXiv-1908, Aug. 2019