

Cracow University of Technology

Faculty of Computer Science and Telecommunications

Natural Language Processing - Erasmus
Academic Year 2024/2025

Text Summarization Application

Luca Bazzetto
Michele Zazzaretti

Abstract

This report presents the entire development process and implementation of a text summarization application built using Python. The main functionality is about reducing a given textual input to the most essential sentences using Natural Language Processing techniques. The application employs TF-IDF vectorization and Cosine Similarity to identify the most semantically significant sentences, to produce an extractive summary. For the graphical aspect, **Tkinter** was used to implement a graphical user interface that allows the users to interact with the program giving the possibility to either pasting the text or by uploading .txt files. This project demonstrates a practical integration of machine learning text analytics, and software design principles for effective information condensation.

Contents

1	Introduction	2
1.1	Aim	2
1.2	Scope	2
1.3	Methodology	2
2	Theoretical Part	3
2.1	Regular Expression	3
2.2	Tokenization and Stopwords	3
2.3	TF-IDF	3
2.4	Cosine Similarity	4
3	Practical Part	5
3.1	Environment and Libraries	5
3.2	Text Summarization Function	5
3.2.1	Text Preprocessing	5
3.2.2	Stopwords Removal and TF-IDF Vectorization	5
3.2.3	Cosine Similarity	6
3.2.4	Sentences Selection and Summary Generation	6
3.3	Graphical User Interface (GUI)	6
3.3.1	Style	7
3.4	File Handling	7
4	Summary	8
	Bibliography	9

Chapter 1

Introduction

1.1 Aim

The main aim of this project is to design and implement an automated text summarization tool. This tool extracts the most relevant sentences from a document, producing a condensed version that is both shorter and easier to understand. Such a tool can support tasks like document review, academic research, and content analysis.

1.2 Scope

The Scope of this project covers the design and development of a sentence-level extractive summarization system using natural language processing techniques. It focuses exclusively on the English language input and it utilizes static methods that are TF-IDF and Cosine Similarity, needed to compute the sentence relevance. The project scope includes the implementation of a (GUI) through the usage of the Python's Tkinter library that enables the user to have an interactive engagement throughout the application usage. Features, such as the possibility to upload the user's own text files, and also download the text after being summarized. The project scope excludes the abstractive summarization methods and instead focuses on the extractive ones.

1.3 Methodology

In order to achieve the goal, we adopted a methodology that combines:

- **Development:** Python serves as the primary programming language, leveraging its libraries, such as the one that has been used to develop the whole application, due to its extensive libraries that are cited below
- **Text Preprocessing:** This stage involved the usage of the Regular Expression (RegEx) Python library, for the purpose of splitting the whole text in sentences. it also involved tokenization and stopword removal using the Natural Language Processing Toolkit **NLTK** Python library
- **Feature Extraction:** This methodology involves the usage of the Term Frequency-Inverse Document Frequency (TF-IDF) model from the Scikit-learn python library to vectorize the text with the specific **TfidfVectorizer**
- **Cosine Similarity computation:** This methodology is employed to compare individual sentence vectors with the document's overall mean vector so that it will allow the identification of the most relevant sentences.
- **GUI development:** the user interface is constructed using the **Tkinter** library, to enable seamless interaction, including inputting or uploading text and also downloading the generated summary.

The whole project was created thanks to the usage of some crucial Python libraries that are fundamental for the purpose of summarizing.

Chapter 2

Theoretical Part

The Text Summarization is the name of the process that involves reducing a body of text to a shorter version that preserves its key topics, information and meaning. The goal of a text summarization is to extract the most important contents of any text and while doing that, it will be also necessary to eliminate redundancy and peripheral details. The summarization can be broadly categorized into two types:

- **Extractive Summarization:** Selects and combines key sentences or phrases from the original text without altering their wording. This method relies on ranking the importance of sentences based on statistical or heuristic measures.
- **Abstractive Summarization:** Generates entirely new sentences that convey the essential meaning of the original text. This technique is more complex and in general it requires deep learning or advanced language models.

In this case we developed an extractive summarization application. This approach was chosen due to its interpretability, simplicity, and suitability for integration with classical Natural Language Processing techniques. In order to construct a perfectly working text summarizer, it was necessary to study and learn from various theoretical concepts in Natural Language Processing (NLP). So in this section, we will explain the theoretical background of the primary techniques that were used in the project.

2.1 Regular Expression

Regular expressions, commonly referred to as RegEx, are powerful tools that are used for searching and manipulating strings based on specific patterns. They are extensively applied in text processing tasks due to their flexibility in handling unstructured data. In the project RegEx was used to implement a way to split the entire text into sentences.

2.2 Tokenization and Stopwords

Tokenization refers to the process of breaking text into smaller units, typically sentences or words. Sentence-level tokenization is crucial for extractive summarization, that is our case, as each sentence becomes a candidate for inclusion in the final summary.

Stopwords are commonly occurring words that carry little semantic weight (e.g., “the”, “is”, “and”). These will be all filtered out using a predefined list provided by the Natural Language Toolkit **NLTK**. Removing the stopwords improves the quality of the vector representation by reducing the dimensionality and computational noise. It is important to filter out the stopwords before performing the vectorization.

2.3 TF-IDF

TF-IDF is a statistical measure that reflects how important a word is to a document in a collection or corpus. The formula is:

$$TF\text{-}IDF(t, d) = TF(t, d) \cdot IDF(t)$$

where:

- $TF(t, d)$ is the frequency of the term t in the sentence
- $IDF(t)$ is the log-scaled inverse of the number of sentences containing t

In other words, TF-IDF can be described as these two components:

- Term Frequency (TF): Measures how frequently a word appears in a sentence.
- Inverse Document Frequency (IDF): This component reduces the weight of overly frequent terms across all sentences, as they are considered less informative.

The resulting TF-IDF vector will represent each sentence as a vector of weights corresponding to the importance of each word, of course, in the context of the entire document. In the application, “TfidfVectorizer” from “scikit-learn” was used to perform this transformation efficiently.

2.4 Cosine Similarity

The Cosine Similarity is a widely used metric for measuring the similarity between the vector representation of a sentence and the average vector of the entire document that are two non-zero vectors.

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

It calculates the cosine of the angle between them, producing a value which range can be between -1 and 1:

- A value of 0 indicates entirely dissimilar vectors
- 1 indicates identical vectors (in terms of direction, not magnitude)

In this project the cosine similarity was used to compare each sentence vector with the mean document vector. Sentences that are most similar to the overall document representation will be considered more important and central to the text’s meaning and will be therefore selected for the summary.

Chapter 3

Practical Part

3.1 Environment and Libraries

The application was developed in Python, a widely used programming language. Several essential libraries were required to implement the text summarizer, including NLTK, Scikit-learn, Tkinter, and NumPy.

- **NLTK**: this library is fundamental for all the Natural Language Processing projects, since it includes a lot of fundamental functions such as the tokenization and the stopwords
- **re**: Regular Expression (RegEx) library is used for the purpose of splitting the text into sentences.
- **sklearn**: needed for the implementation of both the TF-IDF vectorization and the Cosine Similarity.
- **Tkinter**: Graphical User Interface implementation.
- **NumPy**: this library is needed for the creation of big matrices and multidimensional arrays, as well as a lot of basic mathematical functions that were requested to implement some specific parts of our projects.

3.2 Text Summarization Function

The focus of this project is about this precise function, which consists of some important processes to go through, in order to achieve our intended goal.

3.2.1 Text Preprocessing

The first process that we will be going through is splitting the input text that the user provided into sentences, and this will be done with the usage of the regular expression, so the **re** library. It could have been done easily with the split function, but for the sake of preserving the semantics, we decided to adopt another solution that did not delete the dots. And this is done with the following lines of code:

```
sentences = re.findall(r'[^\.\!\?]+\.[\!\?]', text)
return [s.strip() for s in sentences if s.strip()]
```

After generating the sentences for analysis, the next step involves

3.2.2 Stopwords Removal and TF-IDF Vectorization

In order to make the sentences ready for the semantic analysis, we will download and apply the standard list of English Stopwords, available on the **nltk** library, by executing in the function this line of code:

```
nltk.download('stopwords', quiet=True)
stop_words = nltk.corpus.stopwords.words('english')
```

By doing this, all those common, semantically-light words (e.g., the , is, in) will be excluded from the analysis to improve the signal-to-noise ratio in the text representation. Subsequently in the function we will convert all the sentences into TF-IDF vectors by using the **TfidfVectorizer** class from the **sklearn** library. So first we will define the vectorizer, using our just defined stopwords.

```
vectorizer = TfidfVectorizer(stop_words = stop_words)
```

After this we will perform the actual vector creation:

```
return vectorizer.fit_transform(sentences)
```

So now each sentence has become a numerical vector that represents the significance of each term relative to the entire set of all the sentences.

3.2.3 Cosine Similarity

Once vectorized, the core of the summarization algorithm relies on identifying the most accurate and representative sentences. This will be achieved by computing the mean TF-IDF vector of all sentences by executing this line of code:

```
doc_vector = np.asarray(tfidf_matrix.mean(axis=0))
```

Now each sentence's cosine similarity to this mean vector will be then calculated with the below line of code and also thanks to the **cosine_similarity** class from the **sklearn** library as well. The assumption is that the sentences closer to the “average” document vector content are the ones that are more central and suitable for inclusion in the summary.

```
return cosine_similarity(tfidf_matrix, doc_vector).flatten()
```

3.2.4 Sentences Selection and Summary Generation

To generate the summary, the number of sentences will be selected by the user itself, there is the possibility for him to choose whichever percentage he prefers from a range between 10 and 90 percent of the whole input text's number of sentences. This method ensures that the summary is concise while retaining the essential information.

```
n = max(1, int(len(sentences) * ratio))
```

Now we will choose the most similar sentences and sort them in the order they originally appeared in the document, so that the coherence will be preserved.

```
top_indices = scores.argsort()[-n:][::-1]
```

And after doing so, the last but not least thing that remains to do is joining the sentences again to generate the summary text.

```
return ' '.join([sentences[i] for i in sorted(top_indices)])
```

3.3 Graphical User Interface (GUI)

The application's interface was designed using Python's tkinter library with ttk (Themed Tkinter) widgets to ensure cross-platform consistency and modern aesthetics. The GUI adopts a single-window 3.1 with three core components:

1. **Input Section:** A scrollable text area supporting direct text input, pasting, or file loading. Users can upload .txt files via the Load File button.
2. **Control Panel:**
 - Summary Length Slider: Replaces static percentage buttons with an interactive slider (10–90%) for dynamic length adjustment.
 - Generate Summary: Triggers the summarization process using the selected ratio.
3. **Output Section:** A scrollable text area displaying the generated summary, with options to Save Summary (as .txt) or Clear All inputs/outputs.

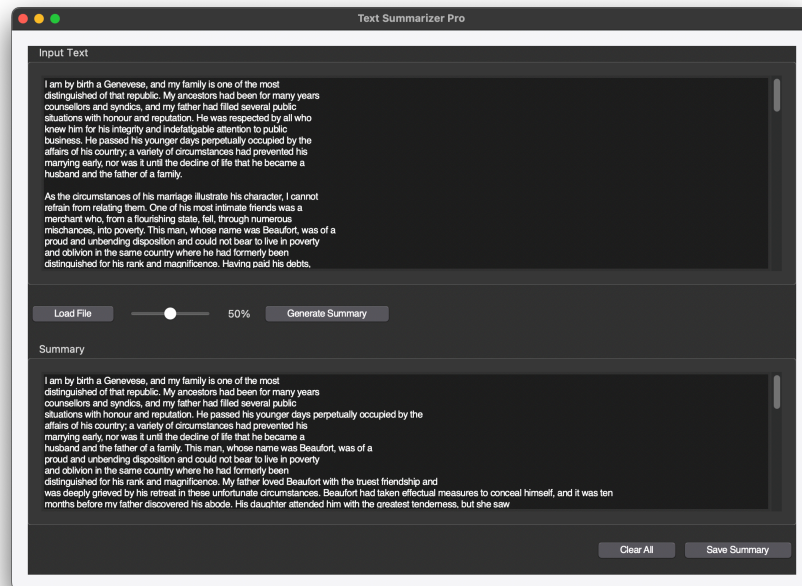


Figure 3.1: Main Frame

3.3.1 Style

An important part for the Graphical User Interface includes the consistent font choices, the colour schemes, and padding enhanced ability. The buttons are styled with a dictionary-based styling pattern to simplify the design consistency.

3.4 File Handling

The FileHandler class manages file operations:

- `load_text()`: Uses `filedialog.askopenfilename()` to read .txt files with UTF-8 encoding.
- `save_summary()`: Writes summaries via `filedialog.asksaveasfilename()`, ensuring proper file extension handling.
- Try-except blocks with messagebox alerts prevent crashes during file I/O.

Chapter 4

Summary

The project successfully developed an extractive text summarization tool capable of condensing a given textual input into a shorter version while preserving the most relevant information. This goal has been successfully achieved through the integration of several key components:

- Sentence segmentation using Regular Expression
- Vector representation via TF-IDF
- Sentence ranking using Cosine Similarity

The implementation also included a Graphical User Interface (GUI) that was built with Tkinter, which enhances the accessibility and the user interaction. Users are provided with the option to input text manually or upload it from a file of their own, after which summary is generated and can also be saved locally. These features enhance the application's versatility and user-friendly, with a potential for further extensions or integrations into more complex systems. From a theoretical point of view, the project incorporated core concepts of Natural Language Processing (NLP), such as the Regular Expression (RegEx) for the text splitting, the stopword elimination, the vector space modeling and techniques for retrieving information. Some of the encountered challenges included ensuring the sentence coherence in the summarized output and also maintaining the performance when working with large text inputs. These issues were mitigated thanks to ordering through each sentence and the efficient use of the cosine similarity measure. In conclusion, the project not only fulfilled its original goal but also served as a valuable opportunity to apply both theoretical knowledge and practical programming skills.

Bibliography

- [1] H. Lane, C. Howard, and H. M. Hapke. *Natural Language Processing in Action: Understanding, Analyzing, and Generating Text with Python*. Manning Publications, 2019.
- [2] NLTK Team. NLTK Documentation. <https://www.nltk.org/>, n.d.
- [3] Python Software Foundation. Tkinter Documentation. <https://docs.python.org/3/library/tkinter.html>, 2023.
- [4] Scikit-learn. scikit-learn: TF-IDF Vectorizer Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, 2023.
- [5] M. Summerfield. *Programming in Python 3: A Complete Introduction to the Python Language*. Addison-Wesley, 2 edition, 2010.