

**BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
SPECIALIZATION COMPUTER SCIENCE**

DIPLOMA THESIS

Enhancing Romanian Text Analysis: A Comparative Study of Keyword Extraction Techniques

**Supervisor
Lect. PhD. Lupea Mihaela**

*Author
Gruia Marc Bogdan*

2023

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ**

LUCRARE DE LICENȚĂ

Îmbunătățirea Analizei Textelor în Limba Română: Studiu Comparativ Privind Metodele de Extragere a Cuvintelor Cheie

**Conducător științific
Lect. Dr. Lupea Mihaiela**

*Absolvent
Gruia Marc Bogdan*

2023

ABSTRACT

This bachelor thesis presents a comparative analysis of keyword extraction techniques for the Romanian language. The aim of the study is to evaluate and compare the performance of three popular algorithms, namely RAKE, TextRank, and TF-IDF, in extracting relevant keywords from Romanian texts. To facilitate easy access and utilization of these techniques, a web-based application is developed, featuring a REST API and a user-friendly website.

The web application allows users to input a text document and select one or more of the above mentioned algorithms for analysis. The selected algorithm processes the text and extracts keywords and phrases based on their respective methodologies. The extracted results are then presented to the user, providing valuable insights into the most significant terms within the input Romanian text. The user has the ability to compare the results with a set of manually selected keywords, scoring the efficiency of each chosen method with a well defined metric.

Contents

1	Introduction	1
1.1	Problem statement	1
1.2	Related work	2
1.3	The purpose of this paper	2
1.4	The structure of this paper	3
2	Theoretical background	4
2.1	Keyword extraction	4
2.1.1	The need for different approaches	4
2.1.2	Classifying the algorithms	5
2.2	The Romanian language	7
3	Technologies used	9
3.1	Web programming	9
3.2	Frontend technologies	10
3.2.1	Hypertext Markup Language	10
3.2.2	Cascading Style Sheets	10
3.2.3	Javascript and Typescript	11
3.2.4	React.js	11
3.3	Backend technologies	12
3.3.1	Python	12
3.3.2	spaCy	12
3.3.3	Django	12

4	RoMiner - different approaches for keywords extraction in Romanian texts	14
4.1	The project	14
4.1.1	What we try to acheive	14
4.1.2	Added value	15
4.2	Software requirements	16
4.3	Keywords extraction	18
4.3.1	Preprocessing	18
4.3.2	The spaCy project	20
4.3.3	Available algorithms	20
4.4	Implementation details	24
4.4.1	System Architecture	24
4.4.2	Accessing the application	27
4.4.3	How we store your data	28
4.4.4	User Manual	29
4.4.5	Measuring the performance	30
5	Conclusions	32
	Bibliography	34

Chapter 1

Introduction

1.1 Problem statement

In the era of information overload, effectively extracting meaningful information from large volumes of text has become a priority in many cases. Keyword extraction [FNAD19], a subtask of natural language processing, plays a vital role in facilitating text understanding, information retrieval, and content analysis. By automatically identifying the most important terms and phrases within a text, keyword extraction techniques enable efficient summarization, categorization, and indexing of textual information.

Keyword extraction finds applications in various domains, including information retrieval, document classification, sentiment analysis, and text summarization. It assists search engines in retrieving relevant documents based on user queries, enhances the accuracy of document categorization by identifying key topics, aids sentiment analysis models in capturing crucial sentiment-bearing words, and provides concise summaries of lengthy documents.

However, while keyword extraction has been extensively studied for major languages, its application and evaluation in the context of the Romanian language remain relatively limited. The linguistic characteristics and unique syntactic structures of the Romanian language pose distinct challenges for keyword extraction techniques. Therefore, there is a critical need to explore and compare different techniques to identify the most effective approaches specifically tailored for the Romanian language.

1.2 Related work

In this subchapter, we present an overview of the existing literature and related work on keyword extraction methods, aiming to understand the current state-of-the-art techniques and their possible limitations.

There are three main algorithms studied in this paper. TextRank [MT04] is a graph-based approach inspired by the PageRank [YD15] algorithm. It's usually incorporated in text summarization algorithms. RAKE [RECC10] uses statistical measures such as word frequency and co-occurrence to efficiently extract keywords from text. TF-IDF [SW10] is a simple but efficient method, also based on statistical measurements. The efficiency of these methods has been widely tested before [GKMM18] [Man20], but not specifically for the Romanian language. Another, more advanced algorithm is YAKE [CMP⁺20] (Yet Another Keyword Extractor) - an algorithm that combines statistical measures and linguistic features to extract keywords, distinguishing itself from TF-IDF, TextRank, and RAKE by considering contextual relevance and utilizing a hybrid approach. FRAKE [ZSFDB21] is also an algorithm worth mentioning here, a relatively new approach that shows signs of improvement of up to 16 percent in F-Score testing [wik06]. These approaches were tested at large for the more popular languages of the world, like English [Li21] or Chinese [Pan23], but not for the smaller ones like Romanian. As such, there is an argument to be made that there is a lack of research in this field.

1.3 The purpose of this paper

This bachelor thesis aims to address this research gap by conducting a comprehensive comparative analysis of keyword extraction techniques for the Romanian language. The primary objective is to assess the performance and effectiveness of three prominent algorithms: RAKE, TextRank, and TF-IDF, in extracting keywords from Romanian texts. The study leverages a web-based application, equipped with a REST API and a user-friendly website, to facilitate the analysis and evaluation of these algorithms.

By aggregating and analyzing data obtained from these different techniques, valuable insights will be generated to enhance the understanding and applicability of keyword extraction for the Romanian language. The comparative analysis will provide a comprehensive understanding of the strengths and weaknesses of each algorithm, their suitability for Romanian language processing, and the factors influencing their performance.

The findings of this research will contribute to the development of optimized keyword extraction techniques for the Romanian language, enabling more accurate and efficient information retrieval, text summarization, and content analysis. Moreover, the web-based application developed as part of this study will serve as a valuable resource for researchers, linguists, and professionals working with Romanian textual data, offering them an accessible platform to extract and analyze keywords using various algorithms.

1.4 The structure of this paper

This paper is structured in 3 chapters, each one subsequently split into subchapters and subsubchapters. The next chapter, Chapter 2, aims to describe the problem that we try to solve. it explains why there is a need for different keyword extraction algorithms, and why they may or may not work best on the Romanian language and it's particularities. Next, Chapter 3 will give us an overview over the tech stack **ref aici** used in building the application. Alongside the actual technologies, it will also provide our motivation to use them and not other alternatives. Lastly, the final theoretical chapter, Chapter 4 will describe how the application was built, from the initial requirements, to the systems architecture and all the way to the expected use-cases.

Chapter 2

Theoretical background

2.1 Keyword extraction

The task of keyword extraction involves automatically identifying and extracting the most significant words from a piece of text, in order to capture its core context.

2.1.1 The need for different approaches

Keyword extraction is a complex task that requires consideration of various linguistic factors, text characteristics, and application contexts. As a result, the need for different approaches in keyword extraction arises to address the challenges and nuances associated with the task. This subchapter explores the reasons behind the necessity for diverse algorithms and highlights why certain approaches may perform better in specific scenarios.

Different languages exhibit unique linguistic patterns, grammatical structures, and semantic rules. Consequently, keyword extraction algorithms must account for these language-specific characteristics to accurately identify and extract keywords. For example, languages with rich inflectional systems may require algorithms that consider morphological variations, while languages with distinct word order patterns may benefit from algorithms that prioritize syntactic information. By developing and adapting algorithms to address these language-specific challenges, more accurate and contextually appropriate keywords can be extracted.

Texts can vary widely in terms of length, genre, domain, and style, necessitating different approaches in keyword extraction. Short texts, such as tweets or headlines, often require algorithms that can capture the essence of the content with limited context. On the other hand, longer texts, such as research articles or legal documents,

may benefit from algorithms that can identify key concepts and themes within the document. Additionally, different domains may have specific terminologies and jargon, requiring algorithms that are trained or fine-tuned on domain-specific data. By utilizing diverse approaches tailored to specific text characteristics, keyword extraction algorithms can better adapt to the unique requirements of different texts.

Keyword extraction serves various purposes across different applications, and the desired outcomes can vary significantly. For instance, in search engine optimization, algorithms that prioritize frequently occurring keywords or phrases may be more effective in improving the visibility and relevance of web content. In document summarization, algorithms that identify the most salient and representative keywords can help in creating concise summaries. Therefore, the choice of algorithm depends on the specific application context and the objectives of the keyword extraction task.

The field of NLP is rapidly evolving, with new techniques and algorithms continually being developed and refined. It was recently proven that methods that incorporate linguistic context will have better performance than those that do not [Nom22]. Advancements in machine learning, deep learning, and statistical approaches have significantly enhanced the accuracy and efficiency of keyword extraction. Researchers explore novel methodologies, such as graph-based algorithms, topic modeling, and neural networks, to tackle the complexities of keyword extraction. By leveraging the latest advancements, researchers can devise innovative approaches that address specific challenges and improve the overall performance of keyword extraction algorithms.

In summary, the need for different approaches in keyword extraction arises due to the linguistic variations, text characteristics, application contexts, and the ongoing advancements in NLP research. By incorporating diverse algorithms, tailored to specific language-specific nuances, text characteristics, and application objectives, more accurate and contextually relevant keywords can be extracted. This allows for improved performance in different scenarios, ensuring that the extracted keywords effectively serve the intended purpose within a given context.

2.1.2 Classifying the algorithms

We can group keyword extraction techniques in three general categories, namely statistical approaches, graph-based approaches and machine learning approaches. We have analyzed each group and identified the benefits, limitations and described in general how they work.

Statistical Approaches

Statistical approaches involve analyzing the statistical properties of words within a given text to identify important keywords. These methods rely on the assumption that important keywords tend to exhibit certain statistical characteristics. It is the most lightweight approach to analyze a piece of text, making it a viable option if you need a quick and straightforward implementation. The biggest limitation though is the lack of context awareness, making these approaches jump completely over the semantic meaning of words. In this category we can highlight algorithms such as TF-IDF, Renyi Entropy [SS21], RAKE and YAKE.

Graph-Based Approaches

Graph-Based approaches involve representing the text as a graph structure, where nodes represent words or phrases, and edges represent relationships or connections between them. These approaches leverage the graph structure to capture semantic and syntactic information. Compared to the previous approach, the semantic meaning is captured and used for computing the final result, but this comes at the cost of a more complex and challenging algorithm. Generally, the most difficult part is the building of an accurate and representative graph. It is not a trivial task to determine a useful relation between nodes and edges, and more often that not these algorithms make use of external language processing databases and tools, which makes them dependent on the availability of such projects. TextRank is a great example of algorithm from this category.

Machine Learning Approaches

Out of all the categories, Machine Learning approaches show the most promising performance and accuracy [Hul03] [KJ17]. It's main advantage is the human-like "understanding" of the text, making the upper-bound of it's efficiency, theoretically, the same as human performance, plus the added benefit of computer-like speed. The development of such algorithms is hard and cumbersome, and even when one is finished we still need a lot of data to train it on. This category includes methods such as Naïve Bayes and SVM.

2.2 The Romanian language

As any other language, the Romanian language has some particularities [RU12] that may or may not alter the effectiveness of keyword extraction algorithms.

Rich Morphology

Romanian is an inflectional language with a rich system of noun declensions, verb conjugations, and adjective agreement. This complexity introduces variations in word forms, making it essential for keyword extraction algorithms to handle morphological variations accurately. Stemming and lemmatization techniques specific to the Romanian language should be employed to ensure the extraction of meaningful and representative keywords.

Word Order

Romanian has a relatively flexible word order, allowing for different sentence structures. This flexibility poses a challenge for keyword extraction algorithms that rely heavily on word proximity or syntactic patterns. Algorithms must account for the varied word order in Romanian sentences to accurately identify keywords and their relationships within the text.

Diacritics

Romanian uses diacritical marks, such as accents and other diacritical signs, to indicate pronunciation and differentiate word meanings. Keyword extraction algorithms should handle these diacritics properly to avoid overlooking important keywords or treating words with diacritics as separate entities. Equivalence mapping techniques or pre-processing steps that handle diacritics normalization are often employed to address this issue.

Complex Noun Phrases

Romanian frequently employs complex noun phrases that involve prepositions, articles, and multiple modifiers. These noun phrases can be challenging for keyword extraction algorithms to interpret correctly. Algorithms need to account for the hierarchical relationships within these noun phrases to accurately extract keywords that represent the intended concepts.

Domain-Specific Terminology

Like any language, Romanian has domain-specific terminology that varies across different fields such as medicine, law, or technology. Keyword extraction algorithms should consider the specific vocabulary and terminology relevant to the domain in which the text is written. Incorporating domain-specific lexicons or training data can improve the accuracy and relevance of the extracted keywords.

It is also worth mentioning that Romanian is not a very popular language. The estimates as of the writing of this paper put the total number of Romanian speakers to about **26 million**, while the two most popular languages in the world - English and Mandarin Chinese, both rack up to **1.2 billion** speakers. Considering this, it is more than likely that any non language-specific algorithm for keyword extraction was more thoroughly tested for the more popular languages of the world. I am not saying that they were *developed* this way, just that statistically more English or Chinese input was fed into them than Romanian. *

Chapter 3

Technologies used

3.1 Web programming

Web programming, commonly known as “website building”, is the term used to group and describe all the processes and tasks needed to build, run and maintain software applications that are available to the user through a browser. Sometimes, wireframing and graphical design are also part of this process, alongside other tasks that are hidden from the user, like network security or database management.

Usually, the development process is split up in two major parts, which according to the most recent best practices should be as little coupled as possible.

Frontend development, known also as client-side programming, is the part of the application that the user interacts with - the GUI. As of right now, the most commonly used technologies for building this part is the markup language HTML, alongside the stylesheet language CSS and the scripting language JavaScript, or even it's superset Typescript. It's main goal is to provide easy and intuitive access to the application, such that a user doesn't need to know or understand how the *backend* works.

Backend development, on the other hand, is the part of the application where the business logic is implemented, the data is stored and the computations are done. It is abstracted from the user by the *frontend* and it can be built in numerous languages, frameworks or architectures.

In the following part I will explain the reasoning behind the tech stack used for building **RoMiner**.

3.2 Frontend technologies

3.2.1 Hypertext Markup Language

HTML, or Hypertext Markup Language, is the standard markup language used for creating web pages and structuring their content. It is the backbone of the World Wide Web and is understood by web browsers to render and display web pages.

HTML uses a markup syntax consisting of tags that define the structure and elements of a web page. These tags are enclosed within angle brackets ("`<`" and "`>`") and are typically represented in pairs, with an opening tag and a closing tag. The content of a web page is placed between these tags.

HTML tags describe the purpose and meaning of the content they enclose. For example, the `<h1>` tag is used to define a heading, while the `<p>` tag represents a paragraph. By using a combination of various HTML tags, you can create headings, paragraphs, lists, images, links, tables, forms, and more.

HTML is by far the most popular and recommended way to build a web application.

3.2.2 Cascading Style Sheets

CSS, or Cascading Style Sheets, is a style sheet language used to describe the presentation and visual layout of HTML documents. It provides a set of rules and properties that control the appearance of web pages, including colors, fonts, spacing, positioning, and more.

With CSS, you can define how HTML elements should be displayed on a web page. Instead of specifying the styling attributes directly within HTML tags, CSS allows you to separate the style information from the structure of the document. This separation of concerns enhances maintainability and provides more flexibility in designing and updating the look and feel of a website.

CSS works by selecting HTML elements and applying specific styles to them. This is achieved through CSS selectors. Selectors target specific elements or groups of elements to which the styles will be applied. For example, the selector `h1` targets all `<h1>` heading elements, while `.navbar` targets an element with the class of "navbar".

CSS is the de facto standard in styling web applications.

3.2.3 Javascript and Typescript

JavaScript is a high-level programming language primarily used for web development. It is often referred to as the “language of the web” as it is supported by all modern web browsers and allows developers to create dynamic and interactive elements on web pages.

TypeScript is a programming language that extends the capabilities of JavaScript by introducing type annotations to JavaScript. By definition, it is a superset of Javascript, and so any valid JavaScript code is automatically valid TypeScript code, but the reverse is not true. It is an open-source language developed by Microsoft and widely used in modern web development.

TypeScript can’t be natively run on the web, so it first needs to be compiled into JavaScript, and only after that shipped to the browser. This is possible because TypeScript adds mostly type annotations and syntactic sugar to JavaScript. This hassle is justified by the great tooling and IDE support for TypeScript, like autocompletion and code-jumping, as well for the “illusion” of writing stringly-typed code, which for large scale projects is definitely the superior approach.

3.2.4 React.js

React.js, often referred to as React, is a popular JavaScript library for building user interfaces (UIs). Developed by Facebook, React provides a component-based approach to UI development, enabling developers to create reusable UI elements and efficiently manage complex UI interactions.

React focuses on the concept of components, which are self-contained, reusable building blocks that encapsulate a specific part of the UI. Each component can have its own state, properties, and rendering logic. By composing these components together, developers can build complex UIs in a modular and maintainable manner.

One of the key features of React is its virtual DOM (Document Object Model) implementation. React maintains a virtual representation of the UI in memory, which is then efficiently updated and rendered to the actual DOM when necessary. This approach optimizes performance by minimizing direct manipulations to the DOM, resulting in faster rendering and improved application responsiveness.

Out of all the similar libraries (like Vue.js and Svelte) and frameworks (Angular), I personally prefer React because it has the smallest set of restrictions about how you should build your project [Pow23]. There is no file structure that you have to follow, no imposed naming conventions and no templates for the actual code. This

is neither good nor bad, it is just a personal preference of mine.

3.3 Backend technologies

3.3.1 Python

Python is a versatile and high-level programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, Python has gained significant popularity and has become one of the most widely used programming languages.

One of the key strengths of Python is its extensive standard library, which provides a wide range of modules and functionalities that can be readily utilized without the need for additional installations. This rich library ecosystem contributes to Python's versatility and enables developers to accomplish various tasks, from web development and data analysis to scientific computing and artificial intelligence.

3.3.2 spaCy

spaCy is an open-source Python library designed for natural language processing (NLP) tasks. It provides efficient and scalable tools for various NLP tasks, such as tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, and text classification.

One of the main strengths of spaCy is its focus on efficiency and performance [SKR⁺19]. It is built with the goal of being fast and scalable, allowing for efficient processing of large volumes of text. spaCy achieves this through optimized algorithms and the use of Cython, a programming language that combines the simplicity of Python with the speed of C.

Out of all the other open-source NLP approaches libraries, we use spaCy as it provides the greatest support for the Romanian language.

3.3.3 Django

Django is a high-level Python web framework that simplifies and streamlines the process of building web applications. Developed with the principle of "Don't Repeat Yourself" (DRY), Django promotes efficient and reusable code, enabling developers to quickly create robust and scalable web applications.

Traditionally, Django follows the Model-View-Controller (MVC) architectural pattern, which it refers to as Model-View-Template (MVT). The model represents the data structure and handles database operations, while the view manages the logic for processing requests and generating responses. The template defines the presentation layer and determines how data is displayed to users.

For this project, the **V(iew)** part will be a REST API that will be consumed by the React frontend. This complete separation of the frontend and the backend is called Client-Server architecture and is the standard for building modern applications. As such, we won't use the **T(emplate)** feature of Django, because the UI will be handled exclusively by the React application.

Chapter 4

RoMiner - different approaches for keywords extraction in Romanian texts

4.1 The project

4.1.1 What we try to achieve

RoMiner is a *primarily* web-based keyword extractor that employs a variety of techniques and advanced features. We present the user with the option to choose any combination of the three available extraction algorithms - **RAKE** (**R**apid **A**utomatic **K**eyword **E**xtraction), **TF-IDF** (**T**erm **F**requency - **I**nverse **D**ocument **F**requency) and **TextRank**. After a piece of text is analyzed, the user also has the option to compare the efficiency of each selected algorithm with a set of manually selected keywords.

This project aims to provide significant value for understanding the most effective method for keyword extraction in the Romanian language, a subsection of the larger keyword extraction field that could use some more research.

By employing three distinct algorithms, RoMiner offers a comprehensive approach to keyword extraction. Each algorithm has its strengths and weaknesses, and by comparing their outputs, users can gain a more complete understanding of which algorithm works best for the Romanian language. This level of comprehensiveness allows users to make more informed decisions when it comes to keyword analysis and optimization.

The main selling point of this application is that it is language-specific. The Romanian language has its unique characteristics and nuances, which can impact the effectiveness of different keyword extraction algorithms. RoMiner ensures that the algorithms are optimized for its linguistic features. This approach increases the ac-

curacy and relevance of the extracted keywords, providing users with a tailored solution for their Romanian text analysis needs.

RoMiner enables users to evaluate the performance of each algorithm by comparing their results side by side. By analyzing the extracted keywords from RAKE, TextRank, and TF-IDF, users can identify which algorithm consistently produces the most relevant and valuable keywords for their specific use case. This performance evaluation allows users to fine-tune their keyword extraction strategy and improve the quality of their analysis.

RoMiner's ability to utilize multiple algorithms makes it adaptable to various contexts and requirements. Different domains or industries may benefit from different keyword extraction techniques. With RoMiner, users have the flexibility to experiment with different algorithms and choose the one that aligns best with their specific goals. This versatility ensures that users can extract keywords that are most relevant and impactful for their particular field.

Manual selection of keywords allows for human judgment and expertise to be incorporated into the evaluation process. This ensures that the extracted keywords align with the specific context, domain, or industry being analyzed, which may not always be captured accurately by the algorithms alone. By comparing the algorithmically generated keywords with manually selected ones, it becomes possible to benchmark for the algorithms' performance, highlighting their precision and overall effectiveness.

4.1.2 Added value

The RoMiner project offers significant added value in the field of keyword extraction for the Romanian language. We will now explore the various contributions and benefits that RoMiner brings to the table.

Different types of text data have distinct characteristics and objectives. For instance, articles often aim to engage readers and provide valuable information, while scientific papers focus on rigorous research and technical terminology. RoMiner acknowledges that algorithm suitability plays a crucial role in capturing the contextual relevance and semantic nuances of diverse text types. By employing three algorithms - RAKE, TextRank, and TF-IDF - RoMiner offers a flexible framework that can adapt to varying text genres and optimize keyword extraction accordingly.

The inclusion of multiple algorithms in RoMiner facilitates tailored solutions for keyword extraction in different text types and domains.

4.2 Software requirements

Software requirements refer to the specific functionalities, features, and qualities that a software system must possess to satisfy the needs and expectations of its users and stakeholders. These requirements define what the software should do, how it should behave, and any constraints or limitations that need to be considered during development. They serve as a foundation for the software development process, guiding the design, implementation, and testing phases. They are usually implementation-agnostic, aiming to also describe the application to non-technical users.

Use case scenarios refer to specific instances or examples that describe the interactions between users (actors) and a system to achieve a particular goal or outcome. They are a valuable tool used when writing the software requirements, specifically to document the functional requirements of a system. Use case scenarios capture the various steps, actions, and conditions that occur during a user's interaction with the system, helping to identify system behavior, inputs, outputs, and dependencies. As such, the tables **tabelul 1** to **ultimul tabel** will be used to describe the functional requirements of the application.

ID and Description	UC-1: Text processing via the GUI
Actor	Web-based user
Description	A user extracts the keywords of a piece of text
Preconditions	The web application is on the home page, /
Postconditions	The web application navigates to the results page, / results
Happy flow	1. The user opens the application 2. The user writes or pastes a text in the input box. 3. The user selects from the sidebar the processing options. 4. The user clicks the "Process" button.
Alternative flow	A1. a) No text processing algorithm was chosen. b) An error message is displayed after the user clicks the "Process" button. A2. a) A blank or empty text was introduced in the text area. b) An error message is displayed after the user clicks the "Process" button.
Exceptions	-

Table 4.1: Process text on the web application

ID and Description	UC-2: Comparing the results
Actor	Web-based user
Description	A user compares the results obtained by the application with a set of manually selected keywords
Preconditions	1. The user was redirected by the app to the /results page 2. The user has selected the "Compare results" before requesting the process of the text. /
Postconditions	The web application highlights the common words between the manual set and computed set. It also assigns a score to each algorithm.
Happy flow	1. The user inputs a set of keywords, separated by newline 2. The application highlights the matching words 3. The application ranks each algorithm's performance compared to the set of manually selected keywords
Alternative flow	-
Exceptions	-

Table 4.2: Comparing the results with a set of manual data

ID and Description	UC-3: Text processing via the REST API
Actor	User with a valid API key
Description	A user extracts the keywords of a piece of text
Preconditions	The user has a valid API key
Postconditions	-
Happy flow	1. The user makes a request to the REST API, adding the piece of text to be analyzed in the <i>request body</i> and specifying the algorithms as <i>query parameters</i> . 2. The application processes the text and responds with the results, grouped by the selected algorithms.
Alternative flow	A1. a) No text processing algorithm was chosen. b) A 400 response is sent back. A2. a) A blank or empty text was sent in the request. b) A 400 response is sent back.
Exceptions	-

Table 4.3: Use of the application via REST

4.3 Keywords extraction

4.3.1 Preprocessing

The preprocessing step is the first step in any NLP-related problem (Natural Language Processing). It involves determining the lemma and part of speech for each element in a sentence. It also generates a set of sentences by splitting the input text. In our approaches, we preprocess the text in three steps, outlined below. It's purpose is to bring the text to a more precise and easy-to-understand state for the machine processing it. as seen in 4.2

Text segmentation

Text segmentation refers to the process of dividing a continuous sequence of text into smaller, meaningful units such as sentences, paragraphs, or tokens. It is usually done with the help of regular expressions. Beforehand, some algorithms require us to replace abbreviations and acronyms with their extended form.

Lemmatization

Lemmatization is the process of determining the base or dictionary form of a word, known as its **lemma**, by considering its inflected variants. For example, the lemma of the Romanian word **cățelul** (translated as **the dog**) is **cățel**, while the lemma for the verb **au mâncat** (**they ate**) is **a mânca**. It helps to reduce words to their canonical form, enabling improved analysis, classification, and information retrieval.

Text filtering

Text filtering refers to the process of selectively removing or retaining specific types of content from a text based on predefined criteria. It involves applying rules or algorithms to identify and eliminate unwanted elements such as stopwords, punctuation, special characters, or noise. Stopwords are defined as a subset of words in any particular language, that pose no semantic value. In Romanian, some examples are **acest** (translated as **this**), **are** (**has**) and **dar** (**but**). Noise may be defined as those words or sequence of words that may pose no significant interest to the user. This is often decided by the POS (Part Of Speech) of the certain word, or by the other words around it. Most often, when talking about keyword extraction, we will only care for the words that are either **nouns**, **verbs** or **adjectives**, as they are the most likely to

contain semantic value, more than adverbs or pronouns or other parts of speech. Text filtering helps to clean and refine textual data, enhancing the quality and relevance for downstream tasks such as information retrieval, sentiment analysis, or text classification.

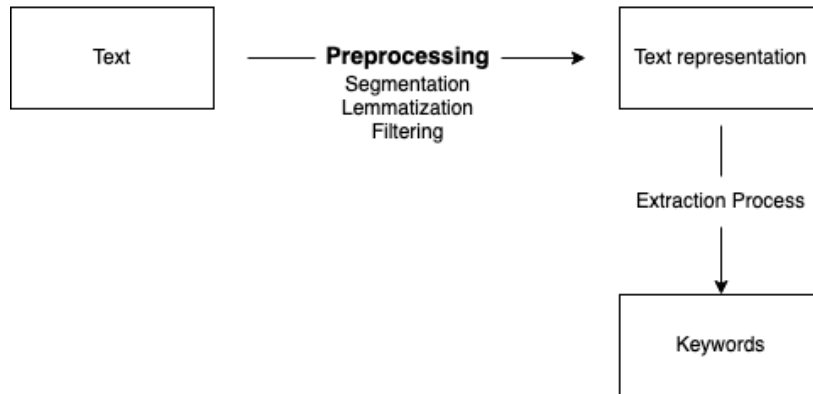


Figure 4.1: Extraction process flowchart

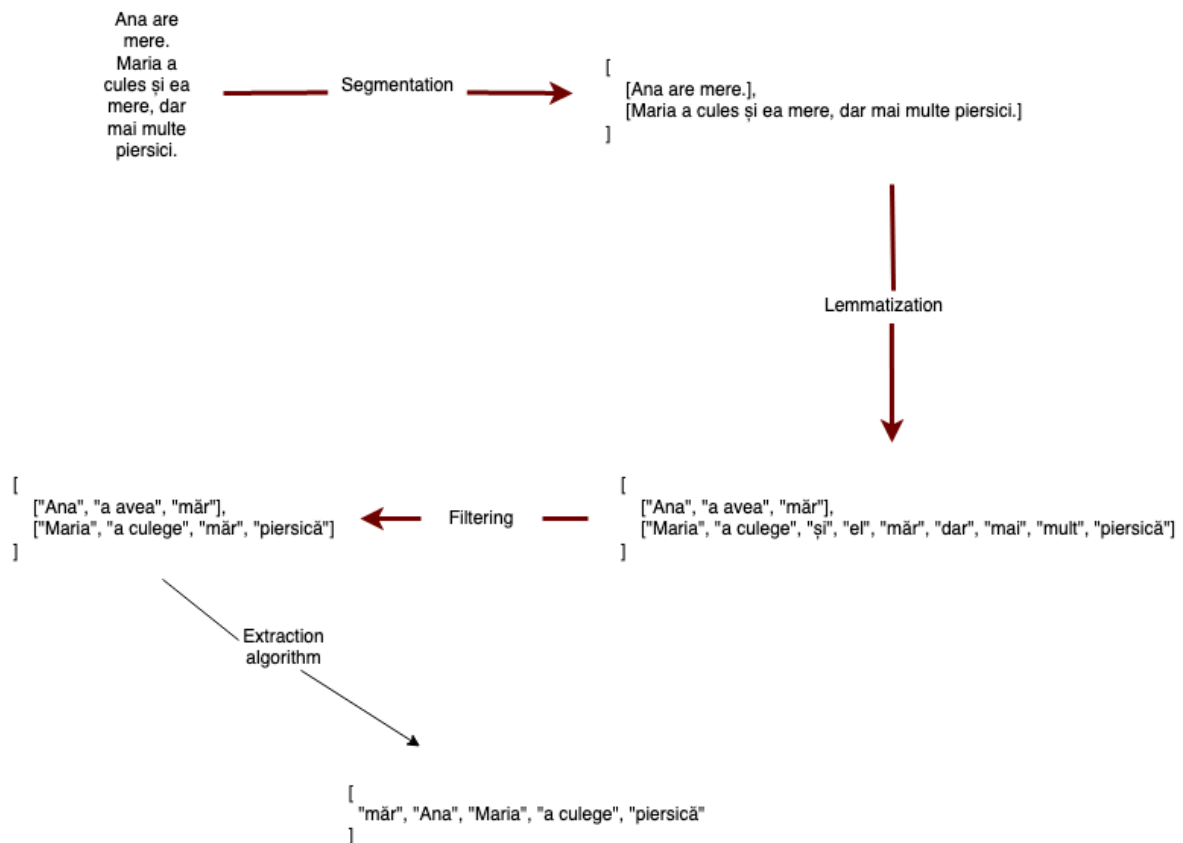


Figure 4.2: Extraction process example

4.3.2 The spaCy project

spaCy is an open-source Python library designed for NLP tasks. It provides efficient and high-performance tools for tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, and other common NLP tasks. It's known for its speed, accuracy, and ease of use, making it popular among researchers and developers for building applications that require advanced language processing capabilities. spaCy also offers pre-trained models for multiple languages, allowing users to leverage the power of NLP without the need for extensive training data or complex algorithms.

4.3.3 Available algorithms

RAKE

The RAKE (Rapid Automatic Keyword Extraction) algorithm is a popular and efficient method for automatically extracting keywords from a given text document. RAKE aims to identify important words that represent the main topics or themes within the text. It is particularly useful for text mining, information retrieval, and natural language processing applications.

The RAKE algorithm operates by combining two main steps: candidate keyword extraction and keyword score calculation. In the candidate keyword extraction step, the algorithm identifies potential keywords by splitting the text into individual words based on punctuation and stop word patterns. Stop words, which are commonly occurring words like "the," "and," or "is," are typically ignored during this process.

Next, the algorithm assigns a score to each candidate keyword based on its degree of importance within the text. The score is calculated by considering the frequency of the candidate keyword in the document and the degree to which it appears in association with other important words. RAKE employs a simple co-occurrence measure to determine the keyword's significance. Words that frequently co-occur with other important words tend to receive higher scores, indicating their potential importance as keywords.

After calculating the scores for all candidate keywords, RAKE sorts them in descending order based on their scores. The top-scoring keywords are then selected as the final extracted keywords representing the main topics or themes of the text.

The RAKE algorithm offers several advantages. It is relatively fast and efficient, making it suitable for processing large volumes of text. RAKE allows for the ex-

traction of multi-word phrases as keywords, which can capture more context and meaning compared to single words.

One of it's biggest disadvantages is that it may produce noisy or redundant keywords, especially when applied to texts with ambiguous or repetitive language. The algorithm also does not consider word order or syntactic structure, which may lead to the extraction of phrases that do not form grammatically correct sentences. Furthermore, RAKE may require fine-tuning and parameter adjustments to optimize keyword extraction results for specific domains or text types.

Input: Preprocessed document D , Output size n

Output: List of extracted keywords K

$Freq \leftarrow$ empty frequency table;

$CoFreq \leftarrow$ empty co-occurrence frequency table;

$Scores \leftarrow$ empty score table;

foreach sentence S in D **do**

$phrases \leftarrow$ Extract candidate phrases from S ;

foreach phrase p in $phrases$ **do**

$words \leftarrow$ Split p into constituent words;

foreach word w in $words$ **do**

$Freq[w] \leftarrow Freq[w] + 1$;

foreach word c in $words$ **do**

if $c \neq w$ **then**

$CoFreq[w, c] \leftarrow CoFreq[w, c] + 1$;

end

end

end

end

end

foreach phrase p in $phrases$ **do**

$score[p] \leftarrow 0$;

$words \leftarrow$ Split p into constituent words;

foreach word w in $words$ **do**

$score[p] \leftarrow score[p] + \frac{Freq[w]}{\sum_{c \in words} CoFreq[w, c]}$;

end

end

$K \leftarrow$ Select top n phrases from $scores$;

return K ;

Algorithm 1: RAKE Algorithm

TF-IDF

The TF-IDF (Term Frequency-Inverse Document Frequency) algorithm is a widely used technique in natural language processing and information retrieval for assessing the importance of terms in a collection of documents. TF-IDF aims to capture the significance of a term in a document by considering both its frequency within the document and its rarity across the entire document collection.

The TF-IDF algorithm operates based on two main components: term frequency (TF) and inverse document frequency (IDF). The term frequency measures the number of times a term appears in a particular document. It assumes that the more frequently a term occurs within a document, the more important it is to that document.

The inverse document frequency component evaluates the rarity of a term across the entire document collection. It calculates the logarithm of the ratio between the total number of documents and the number of documents containing the term. Terms that occur in a small number of documents receive a higher IDF value, indicating their relative uniqueness and importance.

To calculate the TF-IDF score for a term in a document, the algorithm multiplies the term frequency by the inverse document frequency. This results in a score that reflects the term's importance within the document while considering its prevalence across the entire collection. Higher TF-IDF scores indicate terms that are more significant and characteristic of a specific document.

Among its advantages, it is a relatively simple and interpretable algorithm that can effectively capture the importance of terms. It can handle large document collections efficiently and can be applied to different languages and domains. TF-IDF is also flexible, as it allows for customization through techniques like term weighting adjustments or incorporating additional factors.

However, TF-IDF has certain limitations. It does not consider the semantic relationships between terms or the context in which they appear. Thus, it may not capture the full meaning or nuances of terms within the documents. Additionally, TF-IDF treats each term independently, disregarding the potential relationships between terms or the structure of the document.

Input: Preprocessed document D , Output size n

Output: List of extracted keywords K

$Freq \leftarrow$ empty frequency table;

$TFIDF \leftarrow$ empty TF-IDF table;

$d \leftarrow D$ split into sentences;

foreach word w in D **do**

$$\left| \begin{array}{l} tf[w] \leftarrow \frac{Freq[w]}{\sum_{w' \in D} Freq[w']}; \\ idf[w] \leftarrow \log \left(\frac{|D|}{|\{d:w \in d\}|} \right); \\ TFIDF[w] \leftarrow tf[w] \times idf[w]; \end{array} \right.$$

end

$K \leftarrow$ Select top n words from $TFIDF$;

return K ;

Algorithm 2: TF-IDF Algorithm

TextRank

The TextRank algorithm is a graph-based ranking algorithm used for automatic keyword extraction and text summarization. Inspired by the PageRank algorithm used by search engines to rank web pages, TextRank applies similar principles to identify important words or phrases within a given text document.

The TextRank algorithm operates by constructing a graph representation of the text, where nodes represent words, and edges represent the relationships between them. The nodes are weighted based on their importance, which is determined through an iterative process.

Initially, the text is preprocessed by removing stop words and applying part-of-speech tagging to identify nouns, adjectives, and verbs. The resulting words or phrases serve as the nodes in the graph. Co-occurrence statistics are then used to connect the nodes, with edges indicating the strength of the relationship between them. The co-occurrence can be measured by proximity in the text or other statistical measures.

Once the graph is constructed, the TextRank algorithm applies an iterative approach to rank the nodes based on their importance. It starts by assigning an initial score to each node. In each iteration, the algorithm updates the scores based on the scores of connected nodes and the strength of their connections. The process continues until the scores converge or a predefined threshold is reached. This score is calculated as:

$$score[i] = (1 - d) + d * \left(\sum_{j \in \text{neighbours of } i} \frac{\text{weight of edge}[i][j]}{InOut(j)} \right) * score[j] \quad (4.1)$$

where d is a factor set by the user, between 0 and 1. i and j are individual words and are considered neighbours if the edge connecting them has a non-zero value. $InOut(i)$ denotes the sum of all the edges connected to vertex i . The initial weight of each edge is 1.

The final scores obtained from the TextRank algorithm represent the importance of each node (word) within the text. Higher scores indicate greater importance. These scores can be used to extract keywords or key phrases that represent the most significant concepts or themes within the text.

Input: Preprocessed document D , Output size n

Output: Graph G

$G \leftarrow$ empty graph;

foreach sentence S in D **do**

$words \leftarrow$ Split S into constituent words;

foreach word w in $words$ **do**

 Add vertex v with label w to G if it does not exist;

foreach word c following w and $c \in S$ **do**

 Add vertex u with label c to G if it does not exist;

 Add an edge from vertex v to vertex u with weight 1;

end

end

end

Algorithm 3: Graph building in TextRank

4.4 Implementation details

4.4.1 System Architecture

The implementation of the RoMiner application involves a well-structured system architecture and a carefully selected technology stack. This subchapter provides an overview of the key components and technologies used in the development of RoMiner.

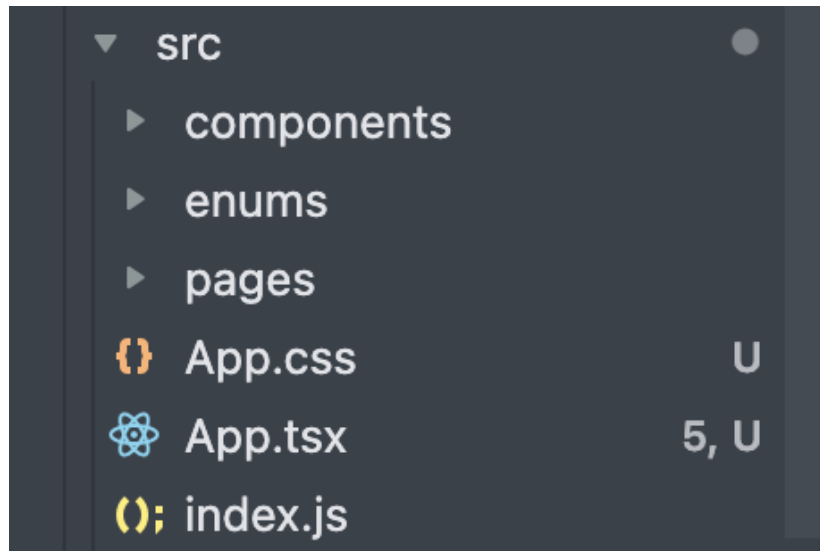


Figure 4.3: Frontend project structure

Frontend: React

The frontend of RoMiner is built using the React framework, a popular and efficient JavaScript library for building user interfaces. React allows for the development of a responsive and interactive user interface, enabling users to interact seamlessly with the application. The React components in RoMiner facilitate a smooth user experience, providing an intuitive interface for inputting text, configuring extraction algorithms, and visualizing the keyword results.

The source files are grouped in three main packages. The components package contains the individual components, like the text input box, alongside their CSS styling. We use **CSS Modules** to avoid classname collisions between components. The **enums** package contains all the different enumerated values, like the backend endpoints or the types of extraction algorithms we support. Finally, the **pages** package encapsulates the different pages of the application, like `/` or `/results`. One page has one or more components, usually not just one, that create the full UI.

Backend: Django

The backend of RoMiner is implemented using the Django web framework, a high-level Python framework that simplifies the development of robust and scalable web applications. Django provides essential functionalities such as routing, request handling, and database management, ensuring the smooth operation of RoMiner. It also enables seamless integration with the frontend, facilitating efficient communication and data exchange between the user interface and the server.

Microservice: Intensive Computations

To handle the intensive computations required for keyword extraction, RoMiner utilizes a microservice architecture. The microservice, specifically designed for performing complex calculations, operates independently from the main application. This architectural approach allows for scalability and improved performance, as the microservice can be scaled up or down based on the workload. It ensures that the main application remains responsive and provides users with a seamless experience, even when processing large amounts of text data. This also allows us to let users interact with our application without using the web-based GUI, which may be desirable in some cases.

The extraction algorithms are separated in different classes that all extend the same base class **TextExtractor**, as described in 4.4 below. We then use the **Factory Pattern** to analyze a given text with the required alorithm(s).

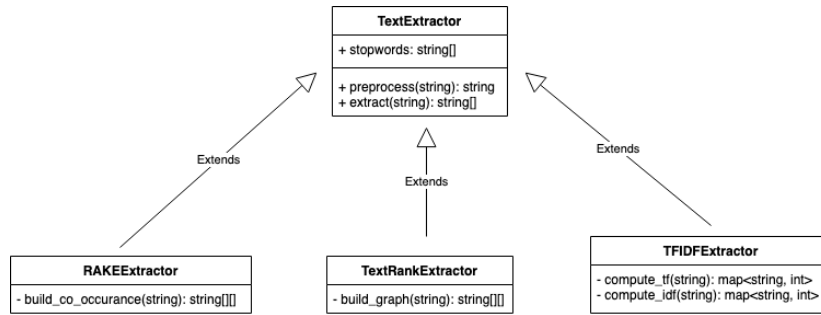


Figure 4.4: Class Inheritance

Communication and Integration

The communication between the frontend, backend, and the microservice is established through well-defined APIs (Application Programming Interfaces). These APIs enable seamless data exchange and ensure efficient integration between the different components of RoMiner. The frontend communicates with the backend API to send user inputs, configuration settings, and retrieve the results of keyword extraction. The backend, in turn, interacts with the microservice API to delegate the intensive computations and retrieve the processed data.

In summary, the system architecture of RoMiner consists of a React frontend, a Django backend, and a dedicated microservice for intensive computations. The combination of these technologies provides a robust and scalable foundation for the application, enabling users to interact with a responsive user interface while the microservice performs the complex keyword extraction algorithms efficiently.

The communication between the components is facilitated through APIs, ensuring seamless data exchange, and integration.

4.4.2 Accessing the application

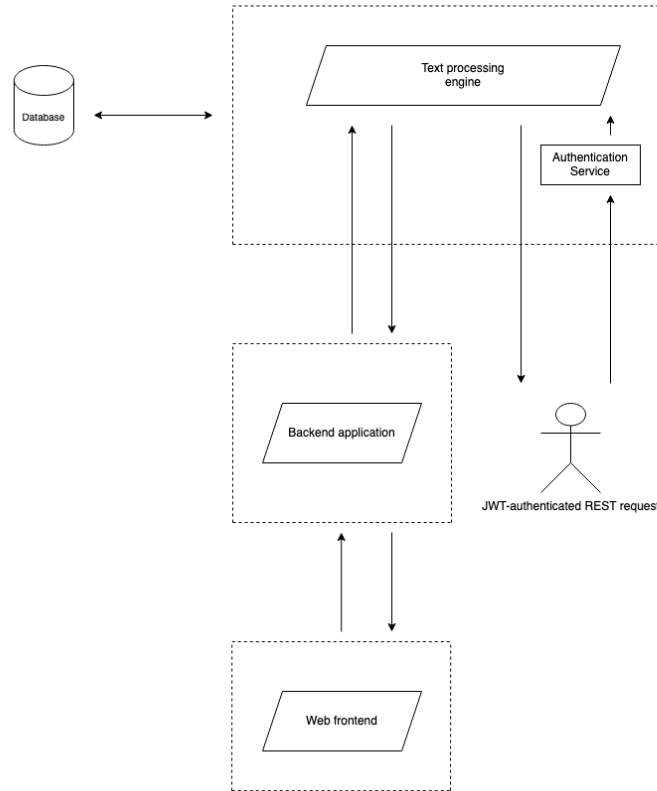


Figure 4.5: System design

This subchapter explores the different access options available for users of the RoMiner application. RoMiner offers a web application for free usage, allowing users to interact with the system through a user-friendly interface. However, for users requiring enhanced performance and task automation capabilities, RoMiner provides direct access to its microservice through a REST API. This section discusses the two access methods and the benefits they offer to users.

Web Application Access

The RoMiner web application serves as the primary access point for users. It offers a user-friendly interface that allows users to input text, configure keyword extraction algorithms, and view the results seamlessly. The web application is freely accessible

to all users and does not require any additional setup or authentication. This access method is suitable for users who require basic functionality and prefer a manual approach to interact with the application.

REST API Access to the Microservice

For users with more demanding requirements, RoMiner provides direct access to its microservice through a REST API. This access option allows users to integrate RoMiner's functionality into their own systems, automate tasks, or achieve higher performance levels. The REST API enables programmatic access to the microservice, allowing users to submit requests, retrieve keyword extraction results, and perform advanced operations.

To ensure secure access to the microservice API, RoMiner implements a token-based authentication mechanism using JSON Web Tokens (JWT). Users can obtain a valid JWT by authenticating with the RoMiner backend. This authentication process grants users access to the microservice API and verifies their identity for subsequent requests.

REST API access to the RoMiner microservice offers several advantages to users. Users can leverage the direct access to the microservice to achieve faster and more efficient keyword extraction. By bypassing the web application interface, users can directly interact with the underlying computational engine, reducing processing overhead and latency.

The REST API enables users to automate repetitive or large-scale keyword extraction tasks. By integrating the RoMiner functionality into their own applications or scripts, users can streamline their workflows and programmatically perform keyword extraction operations.

REST API access allows users to integrate RoMiner seamlessly into their existing systems or workflows. This flexibility empowers users to tailor the application to their specific requirements, incorporating keyword extraction functionality directly into their data pipelines or software solutions.

4.4.3 How we store your data

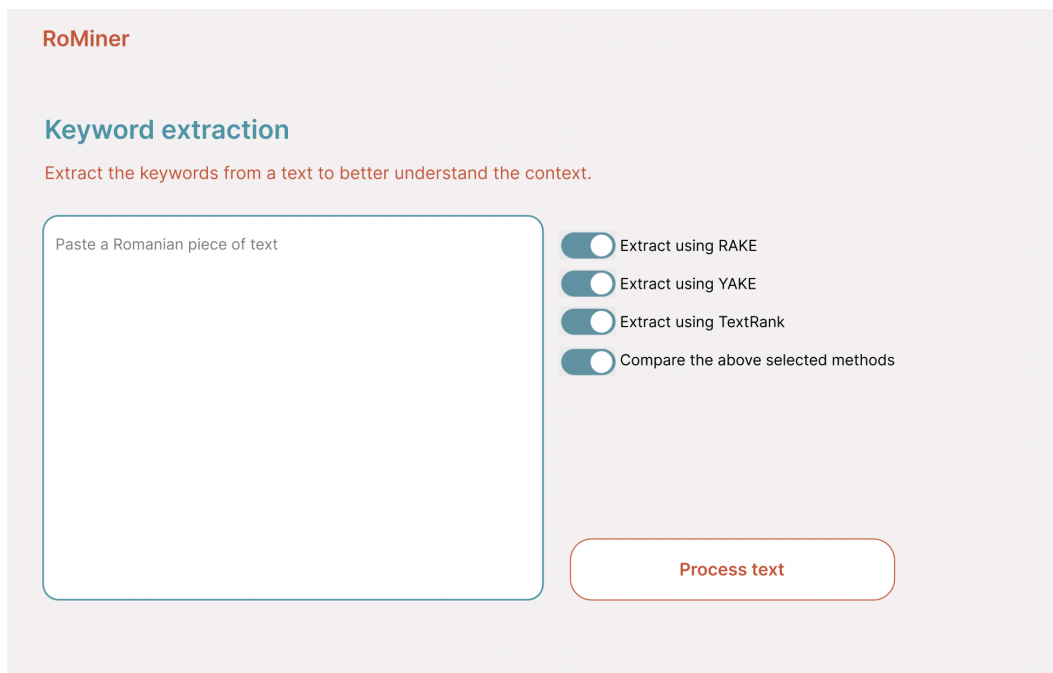
RoMiner follows a strict policy of not storing any personally identifiable information (PII) of its users. PII includes data such as names, email addresses, or any other personal details that could be used to identify individuals. By not collecting or storing PII, RoMiner ensures that user privacy is safeguarded.

While RoMiner refrains from storing PII, it does maintain a record of the input texts, their types, and the corresponding keyword extraction results. This data is valuable for generating insights and improving algorithm selection for different kinds of texts. The recorded information allows RoMiner to analyze patterns and evaluate the performance of various keyword extraction algorithms in specific text contexts.

The data recorded by RoMiner plays a vital role in the development of algorithm selection recommendations. By analyzing the aggregated data, RoMiner can identify trends and patterns related to algorithm performance in different text types. This analysis enables RoMiner to suggest specific algorithms that are more likely to yield accurate and relevant keyword extraction results for a given text genre or domain. These algorithm suggestions help users make informed decisions and streamline their keyword extraction process.

4.4.4 User Manual

Following, we will present the user manual for the **RoMiner** application. The home page / prompts you to select one or more of the available algorithms to analyze a Romanian text. After the processing is done, the user is redirected to the **/results** page, where the keywords are shown in an intuitive way. Here, if the user has chosen the option before, he can compare the results with a set of manually extracted keywords.



The screenshot shows the 'RoMiner' application interface for keyword extraction. At the top left, the 'RoMiner' logo is displayed in red. Below it, the section 'Keyword extraction' is highlighted in blue. A red instruction line reads: 'Extract the keywords from a text to better understand the context.' On the left, there is a large text input area with the placeholder text 'Paste a Romanian piece of text'. To the right of the input area, there are four toggle switches, each with a blue handle and a label: 'Extract using RAKE', 'Extract using YAKE', 'Extract using TextRank', and 'Compare the above selected methods'. All four toggles are currently turned on. At the bottom right, there is a red-outlined button labeled 'Process text'.

Figure 4.6: Home page

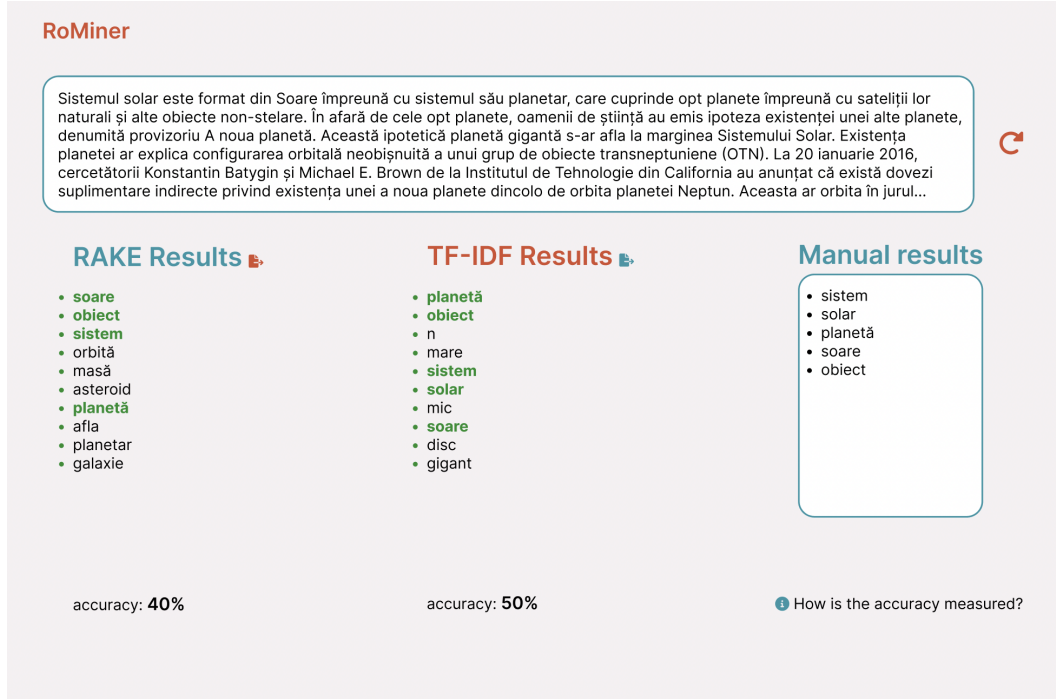


Figure 4.7: Results page

4.4.5 Measuring the performance

A metric is crucial to compare results in this paper as it provides a quantitative evaluation of the performance and accuracy of the extraction process. The proposed metric, denoted as *acc* (accuracy) is defined as the ratio of the number of extracted words found in the manual input, divided by the total number of words in the manual input. By using this metric, we can objectively assess the effectiveness of the extraction method and determine its success in capturing the relevant information.

$$acc = \frac{\text{number of extracted words found in the manual input}}{\text{number of words in the manual input}} \quad (4.2)$$

Manual Test Results

We will now take a look over a set of manual tests conducted on our application, using the above mentioned metric. For this tests we've used a variety of Romanian texts, varying in size, context and complexity. For these tests we've chosen the **top 10** keywords from each algorithm.

While our keyword extraction tests provide some useful insights, their effectiveness is indeed limited and can be enhanced by incorporating the expertise of volunteers who can manually extract keywords from a larger corpus of Romanian texts, ensuring a more diversified data set.

Text No.	Type	Size	RAKE	TF-IDF	TextRank
1	Literature	Medium	80%	60%	40%
2	Literature	Medium	60%	60%	60%
3	Literature	Large	60%	50%	30%
4	Scientific	Medium	40%	60%	50%
5	Scientific	Large	50%	50%	60%
6	Scientific	Large	30%	70%	60%
7	Scientific	Large	60%	70%	70%
8	Scientific	Small	70%	70%	60%
9	Conversation	Small	50%	70%	30%
10	Conversation	Small	60%	60%	30%
11	Conversation	Medium	50%	70%	30%

Table 4.4: Obtained accuracy for each algorithm

Chapter 5

Conclusions

By providing a comprehensive set of tools, RoMiner empowers users to choose the most suitable algorithm based on their specific requirements. This tailored approach ensures that the extracted keywords accurately represent the content and objectives of the given text, enabling users to derive more meaningful insights, optimize their analysis, and make informed decisions based on the specific text type they are working with.

RoMiner goes a step further in providing added value by enabling users to manually compare the results of the keyword extraction algorithms and collecting feedback on the type of text used. When users engage in the manual comparison feature, RoMiner prompts them to specify the type of text they are working with, such as articles, scientific papers, or other domains. This user-driven evaluation allows RoMiner to gather valuable information on the performance and accuracy of each algorithm for different text types over time.

By storing the relevant scores of each algorithm for each type of text, RoMiner can build algorithm accuracy profiles specific to different text genres. Over time, as more users contribute their evaluations, RoMiner accumulates a wealth of data that enables the assessment of algorithm performance across various text types. These accuracy profiles offer added value by providing insights into the strengths and weaknesses of each algorithm for specific contexts. Users can then leverage these accuracy profiles to make informed decisions on which algorithm is most suitable for their particular text type, optimizing their keyword extraction process and analysis.

The collection of user feedback and the creation of algorithm accuracy profiles allow RoMiner to continuously improve and customize its keyword extraction capabilities. By analyzing the accumulated data, RoMiner can identify patterns, refine algorithms, and enhance their accuracy for specific text types. This continuous

improvement ensures that RoMiner remains up-to-date and relevant, consistently providing valuable and accurate keyword extraction results for different domains and text genres.

In summary, RoMiner's added value to the existing body of research is the focus on user-driven evaluation and the creation of algorithm accuracy profiles. By collecting user feedback on the type of text used and storing relevant scores, RoMiner builds accuracy profiles that provide insights into algorithm performance for different text genres. This data-driven approach empowers users to make informed decisions, optimize their keyword extraction process, and enhance the accuracy and relevance of their analysis.

Bibliography

- [CMP⁺20] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Mário Jorge, Celia Nunes, and Adam Jatowt. Yake! keyword extraction from single documents using multiple local features. *Inf. Sci.*, 509:257–289, 2020.
- [FNAD19] Nafise Firoozeh, Adeline Nazarenko, Frédéric Alizon, and Béatrice Daille. Keyword extraction: Issues and methods. *Natural Language Engineering*, 26(3):259–291, Nov. 2019.
- [GKMM18] Shweta Ganiger and Rajashekharaiyah K M M. Comparative study on keyword extraction algorithms for single extractive document. pages 1284–1287, 06 2018.
- [Hul03] Anette Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223, 2003.
- [KJ17] Bhavneet Kaur and Sushma Jain. Keyword extraction using machine learning approaches. In *2017 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA)(Fall)*, pages 1–6. IEEE, 2017.
- [Li21] Jinye Li. A comparative study of keyword extraction algorithms for english texts. *Journal of Intelligent Systems*, 30(1):808–815, 2021.
- [Man20] Nikita Mangwani. Key phrase extraction from document using rake and text rank algorithms. 12 2020.
- [MT04] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- [Nom22] Taro Nomoto. Keyword extraction: A modern perspective. *SN Computer Science*, 4(1), Dec. 2022.

- [Pan23] Rui Pan. Automatic keyword extraction algorithm for chinese text based on word clustering. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 04 2023.
- [Pow23] Zadhid Powell. Angular vs react: A detailed side-by-side comparison. 03 2023.
- [RECC10] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. *Automatic Keyword Extraction from Individual Documents*, chapter 1, pages 1–20. John Wiley Sons, Ltd, 2010.
- [RU12] Georg Rehm and Hans Uszkoreit. *The Romanian Language in the Digital Age*, volume 16. Springer Science & Business Media, 2012.
- [SKR⁺19] Xavier Schmitt, Sylvain Kubler, Jérémy Robert, Mike Papadakis, and Yves LeTraon. A replicable comparison study of ner software: Stanfordnlp, nltk, opennlp, spacy, gate. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 338–343, 2019.
- [SS21] Aakanksha Singhal and DK Sharma. Keyword extraction using renyi entropy: a statistical and domain independent method. In *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 1, pages 1970–1975. IEEE, 2021.
- [SW10] Claude Sammut and Geoffrey I. Webb, editors. *TF-IDF*, pages 986–987. Springer US, Boston, MA, 2010.
- [wik06] F-score — Wikipedia, the free encyclopedia, 2006. [Online; accessed 03-March-2023].
- [YD15] Eric J. Yates and Luke C. Dixon. Pagerank as a method to rank biomedical literature by importance. *Source Code for Biology and Medicine*, 10:16, 2015.
- [ZSFDB21] Aidin Zehtab-Salmasi, Mohammad-Reza Feizi-Derakhshi, and M. A. Balafar. Frake: Fusional real-time automatic keyword extraction. *ArXiv*, abs/2104.04830, 2021.