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

DIPLOMA THESIS

Automatic Summarization of Romanian Texts

Supervisor:
Lect. PhD Lupea Mihaiela

Author:
Anbruș Ioana Andreea

2020

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Sumarizarea automată a textelor în limba română

Conducător ştiinţific:

Lect. Dr. Lupea Mihaiela

Autor:

Anbruș Ioana Andreea

2020

Abstract

The purpose of this paper is to research automatic text summarization of Romanian texts and create a tool for computing automatic extracts by using a clustering technique.

Automatic text summarization is an important task of Natural Language Processing, which is highly used in other fields as news generation, Search Engine Optimization (SEO) or analytics. Although research in this area began over 50 years ago, new tools and techniques appear every day, but in the Romanian language area there is not very much development.

This paper will describe a summarization algorithm based on clustering, which will generate extracts of Romanian texts. Given a text in Romanian, a title, a compression rate and a method of clustering, the algorithm will determine the most important sentences included in the input text and decide which ones will be part of the extract.

As a practical tool for the algorithm, a web application which expects as input a Romanian text with more than 10 sentences and its title is created. The output is the summary of the text, computed on a web backend server, respecting the REST architectural style. The chosen technologies for developing the tool are Python with Flask framework for the backend server and React, the JavaScript library for the frontend server.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Contents

1. Introduction

The purpose of this chapter is to present the motivations of writing this paper and the problem statement. It also contains a brief description of the related work.

1.1 Motivations and Applications

Although automatic text summarization began over 50 years ago and it is still one of the most researched areas among the Natural Language Processing (NLP) community [8], there is very little research in this area for the Romanian language. At the same time, summarization is a very important task that supports developing other technologies in Natural Language Processing.

There are several fields where automatic text summarization is used, such as: news generation, analytics, Search Engine Optimization (SEO), report generation.

News writing can be time consuming, as the information about certain events can be rich and people tend not to read very long text. In order to save time, the news can be generated using automatic text summarization, having as input the initial information. Reports can also be rich in information, so automatic text summarization can help in making the information more concise and easier to read and understand.

Search Engine Optimization is a set of strategies, techniques and tactics used in the creation of a website, in order to assure its high-ranking placement in the search results page of a search engine (such as Google, Bing, Yahoo). Text summarization can be used by SEO specialists in order to assure the correct metadata describing the website and the right fit in the SEO set of rules. For example, right now, Google truncates the description text to 155-160 characters, so the good practice here is to have a description up to 160 characters, providing the most important details about the website. An automatic text summarization would help the SEO specialist by taking all the text in the website and creating a description of the exact needed length.

Another application of automatic text summarization would be in analytics, the technique can come in handy when analysing user comments (about a certain topic, product etc.) and selecting the relevant results, achieving an easier identification of the focus points.

1.2 Problem statement

The main goal of this paper is to investigate automatic text summarization in general, the already made development in this area and creating a tool for creating extracts for Romanian texts. The tool, as a web application, receives as input a single document in Romanian, performs a summarization algorithm and returns the summary (extract) of the text. The extract contains the most important sentences based on some criteria and according to a compression rate. The summarization will be based on clustering.

1.3 Related Work

The studies in text summarization is in continuous development since the late 1950s, although more tools for summarization started to develop in the late 1990s.

One of the first approaches in creating extracts was Luhn's system, created in 1959, which was a single document summarization system for technical articles based on term filtering, word frequency and weighting sentences [12]. Another early approach was Edmunson's single document extractive summarization system in 1969, which analyzed articles [7], or the ADAM system in 1975, which created indicative abstracts (original sentences are edited, no new sentences are created) of chemistry specific texts [17, 11].

From 1995 a lot of systems began summarization on **news articles**, such as ANES (1995) - based on term and sentence weighting and the first sentence was always added to the extract [5, 11]; Marcu (1997) - uses Rhetorical Structure Theory trees [11]; MultiGen (1999) - this system created abstracts, based on reformulation and could also receive multiple documents as input [2, 11]; LAKE (2004) - generates very short extracts of single document news using supervised learning and features like name entity recognition [6, 11]; NetSum (2007) - generates extracts based on machine learning techniques [21].

Another large area of reaserch in text summarization is **online news**. One of the first online news summarization systems is SUMMONS (1998) which is also one of the first multi-document summarization systems that could also return both extracts and abstracts [14, 11]. Other worth mentioning systems are: NewsInEssence (2001) which created personalized extracts on multiple documents [18, 11] or Newsblaster (2002) which created extracts from multiple documents and also displays thumbnails of images [13].

Although a lot of systems are domain-based, there are some that do not depend on any domain and perform summarization on any kind of texts. One notable system is GISTSumm (2003) which generates extracts of single-document texts based on selecting the most meaningful sentences by finding and analyzing the keywords of the text [15].

1.4 Outline

The next chapters offers a detailed overview of the summarization algorithm and the created tool for Romanian document summarization. Chapter 2 presents theoretical aspects regarding summarization and clustering. Chapter 3 describes the summarization algorithm implementation, while Chapter 4 focuses on the architecture, design and implementation details concerning the web application. Chapter 5 presents an example of summarization and an evaluation of the algorithm. The paper is ended with Chapter 6, which focuses on the overview, reflections and further development regarding the summarization tool.

2. Theoretical Aspects

This chapter introduces the definition of automatic text summarization and describes the stages of development in automatic text summarization.

2.1 Definitions

A **summary** is defined as a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s), and that is no longer than half of the original text(s) [10].

The most important part of the summarization task is to identify the most salient parts of the information provided in the original text, removing the rest of it.

Automatic Summarization is a process in which an algorithm creates the summary of a text by identifying the most important parts and keeping it short. Automatic Summarization can be **extractive** or **abstractive**

An **extract** is a selection of significant sentences of a text [11], while an **abstract** is a shortened form of speech/text that can serve as a substitute to the original text [7]. While they both represent a summary of a given text, the differences between the extractive based summarization and abstractive based summarization will be presented in the next paragraphs.

The length of the summary is based on the **compression rate**, which is a percentage of the initial text's length. The length can be expressed as the total number of words or the total number of sentences. Usually, the compression rate for a summary is between 30% and 50%.

2.2 Extraction Based Summarization

Text summarization based on extraction involves selecting the most meaningful sentences from the original text and presenting them as they are, avoiding the redundant information. The sentences are selected based on multiple criteria.

The extractive summarization process includes 2 stages, illustrated in Figure 2.1. The first stage is the **analysis** phase: having the input source text, the sentences are transformed into some more meaningful objects for the algorithm by calculating the frequency of the words and including other features for ranking the sentences. The other stage is the **synthesis** phase: select the most important sentences for creating the extract. After the selection is made, the other step for obtaining the extract is mapping the resulted sentences of the selection to the actual sentences from the source text.

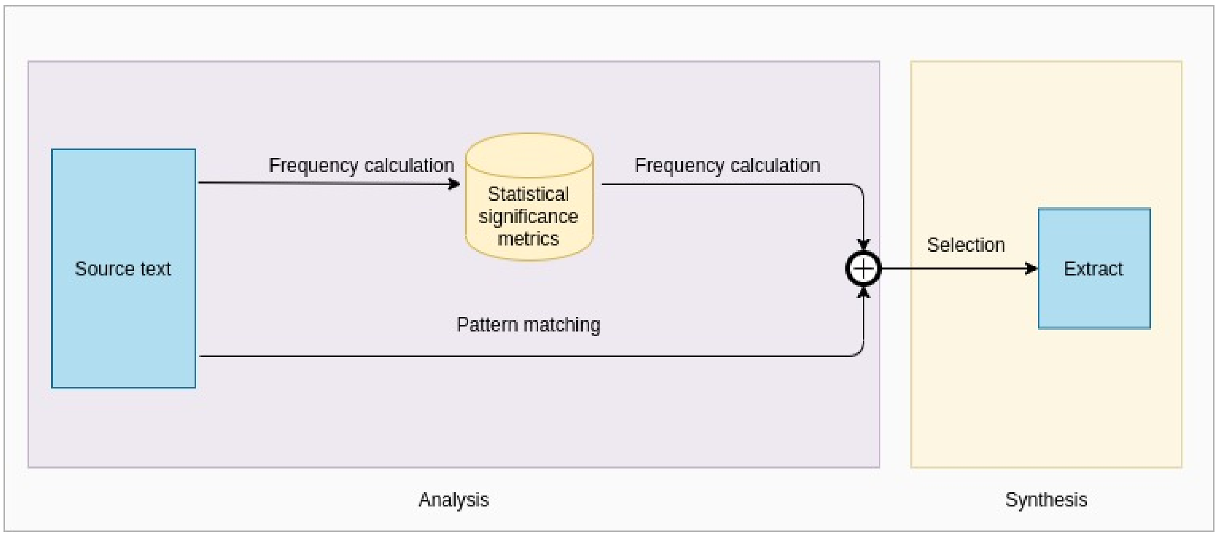


Figure 2.1: Extractive Summarization Schema [9]

The analysis phase consists of choosing some features in order to represent the sentences and rank them. The first and most common feature is the keyword frequency in a sentence; if a sentence contains frequent used words (usually nouns), it is more likely to appear in the summary. Also, sentences that contain the words from the title, proper nouns, acronyms or dates are likely to appear in the summary. The location of the sentences is also important, usually the first and last sentences have high chances of being included in the summary. Another important factor is the length of the sentences; the very short or very long sentences have a low probability of being included in the summary. An extractive algorithm can prioritize the domain specific keywords, meaning the words which belong to a certain field (medical, scientific, botanical, etc.) will weight more than regular keywords. Sentences containing pronouns should not be included in the summary, unless they are expanded into corresponding nouns, using anaphora resolution (resolving references to earlier words in the text), because the meaning and the consistency of the information can be lost. The enumerated features are used in ranking each sentence, the sentence with the highest rank in the cluster will be included in the summary. The techniques for extractive summarization will be further detailed in the next sections.

2.3 Abstraction Based Summarization

The summarization based on abstraction requires an algorithm that rephrases and shortens the original text. The algorithm understands the original text and creates a summary by rephrasing it (as a human being would). The abstraction uses linguistic methods and machine learning for text examination and interpretation. Because this method heavily employs machine learning from NLP such as grammars and lexicons, it can be considered a better approach for obtaining the summary of a document, but is also more difficult to implement.

The stages of the abstractive summarization are illustrated in Figure 2.2. As the extractive sumarization, the first stage of the abstractive summarization is the **analysis** phase: transforming the input source text into an internal representation by parsing and understanding it. Then, in the **transformation** phase, the internal representation of the source text is converted into an internal representation of the summary, by condensation and compaction. The last stage is **synthesis**, which involves generating the abstract from the internal representation of the summary.

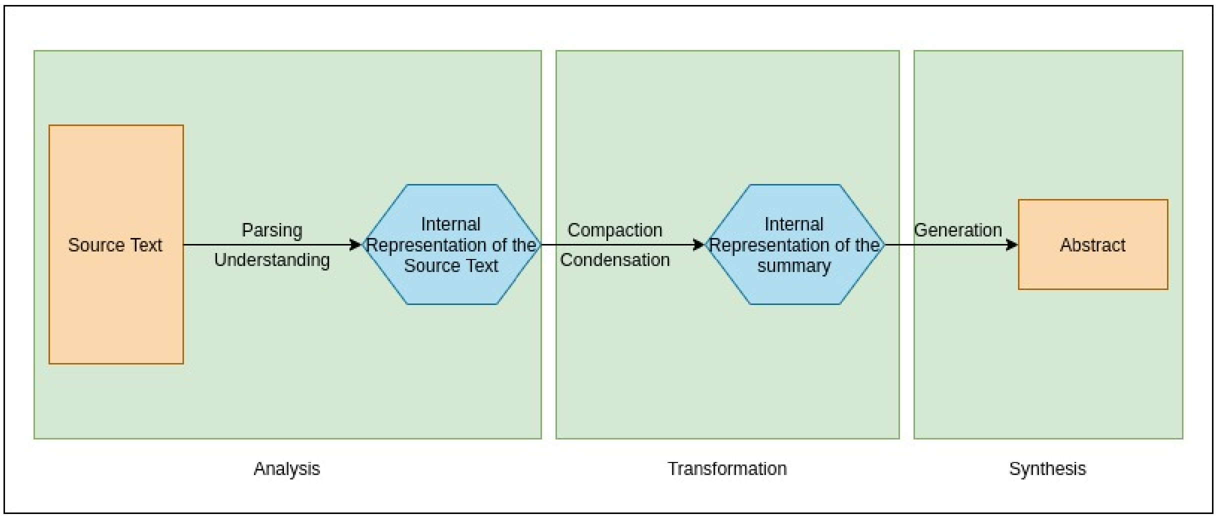


Figure 2.2: Abstractive Summarization Schema

2.4 Clustering

A **cluster** is a homogeneous group of objects (or cases, observations) [20]. **Clustering** is the task of grouping objects into clusters, based on similarities. Usually, the input data used in clustering are vectors representing high dimensional objects, having simple types such as numerical or boolean. Clustering is a method of **unsupervised learning** and it is used for statistical data analysis, for example in marketing, for determining the focus group points in customers' needs and interests, in biology, for classifications among different species and other fields. It will be used in the automatic text summarization, for determining which sentences are similar, by grouping them into different clusters and then, from each cluster choosing the most representative one to be part of the summary.

2.4.1 Types of Clustering

A clustering algorithm has a very clear task: having the raw data as input, by doing some computation, retrieves the clusters (Figure 2.3). The computations made for generating the clusters can be different from situation to situation. One way of categorizing the clustering algorithms is: **hard clustering** and **soft clustering**. While in hard clustering each point of the input data belongs to a single cluster,in soft clustering each point receives a probability of belonging to every defined cluster.

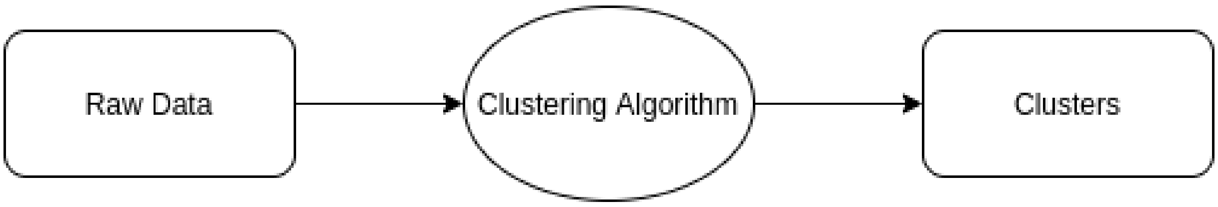


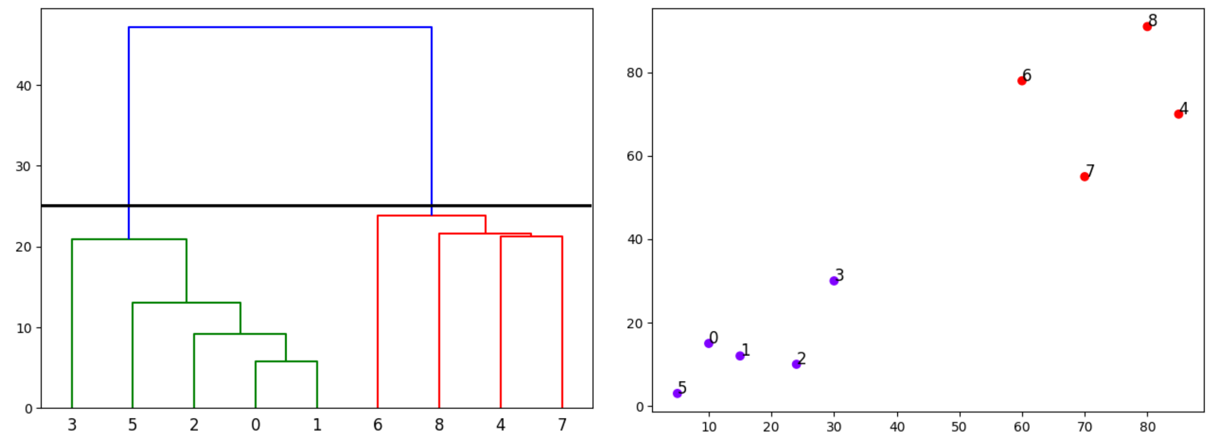
Figure 2.3: Clustering Schema

The clustering algorithms can be also categorized by technique: hierarchical clustering, detailed in section 2.4.2, density-based clustering, grid-based clustering, partition-based clustering, detailed in section 2.4.4 on K Means algorithm. The density based algorithms are able to determine clusters by finding the regions which grow with high density. A representative algorithm for this type of clustering is DBSCAN (Density Based Spatial Clustering of Applications with Noise), but this technique will not be used in the summarization process. The grid based algorithms create clusters by considering the space surrounding the data points, not with the data points [1].

2.4.2 Hierarchical Clustering

Hierarchical clustering consists in dividing the similar data sets and constructing a hierarchy of clusters [1]. Hierarchical clustering can have 2 approaches: **bottom-up** approach and **top-down** approach.

The bottom-up technique (also known as hierarchical agglomerative clustering or HAC) considers each data point as a single cluster, and merges the points successively to form other clusters, until all clusters are merged into one cluster having all data points. By this technique, a dendrogram of the clusters can be constructed and by following it the history of the cluster merging can be tracked. Figure 2.4 is a visual representation of a bottom-up clustering on a data set of 9 points, being split into 2 clusters. The clustering is based on the euclidean distance, each cluster being represented using a different color. The dendrogram (a) has on the Ox axis the labels of the points to be merged, and on the Oy axis the euclidean distances. The figure shows the steps made in each iteration, the first merged points being 0 and 1, then the cluster composed by the points 0 and 1 is merged with point 2 and so on. After all the merges are preformed, the two clusters defined can be visualized in Figure (b). The clustering process stopped when the 2 desired clusters were defined. The possibility of stopping the process when the number of desired clusters are defined represents an advantage of the Hierarchical Clustering algorithms.



(a) Dendrogram **(b)** Clusters

Figure 2.4: Hierarchical Clustering Schema

The top-down approach is the opposite of the bottom-up: starting from a single root cluster which contains all the data points, the algorithm will split the clusters step by step until the leafs represent a cluster with a single data point.

In this text summarization algorithm, the bottom-up approach will be implemented.

2.4.3 Hierarchical Agglomerative Clustering Algorithms

The first step in implementing the Hierarchical Agglomerative Clustering Algorithm, as it represents a bottom-up approach, is to consider each data point as a single cluster. After the initialization step, the algorithm will merge two clusters at once according to the similarity measure and a type of linkage between the clusters that we define.

For determining the distance between two clusters, a **linkage algorithm** needs to be defined. Some of the most popular linkage algorithms are: single-link, complete-link, average-link or centroid-link. The **single-link** considers the distance between two clusters as the distance between the closest members in the two clusters. The **complete-link** is the opposite of the single-link, meaning the distance between two clusters is considered to be the distance between the two furthest apart points belonging to the separate clusters. The **centroid-link** defines the distance between two clusters as the distance between the clusters' centroids, while **average-link** defines the distance between two clusters as the average distance between all the pairs of the clusters elements [19].

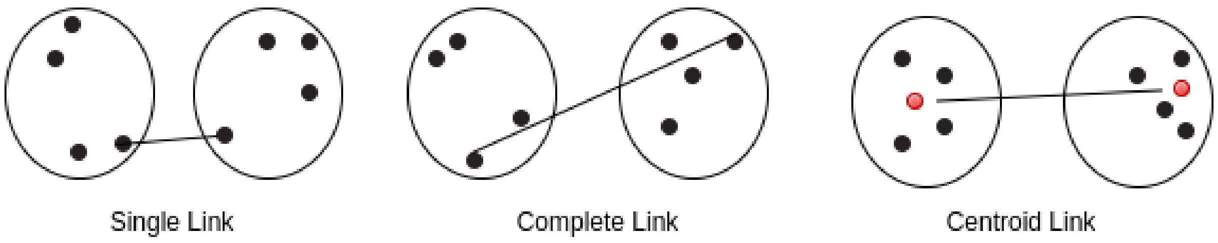


Figure 2.5: Cluster Linkage Algorithms Representation

Besides computing the distance between clusters, it is important to also have a method for computing the distance between two points. This distance is defined as similarity measure and it will be detailed in section 2.5.

2.4.4 K Means Clustering

K Means clustering is an approach to **partitioning clustering**. The partitioning clustering involves splitting the data into a given number of clusters. The K Means algorithm will choose randomly k centroids from the given data points and then will assign each point to the most similar centroid, forming the initial clusters. After having the initial clusters, the centroids are recomputed in order to find the better solution. K Means algorithm does not always give perfect results, as the centroids are randomly chosen at first and then recomputed. One very important aspect to this algorithm is the mean computed after the data points are assigned to the centroids, because this will compute the next centroids.

Figure 2.6 is a visual representation of a K Means clustering on a data set of 100 randomly selected points (in a two dimensional space) and K = 4, the given number clusters. Each color represents a cluster and the stars represent the resulted centroids.

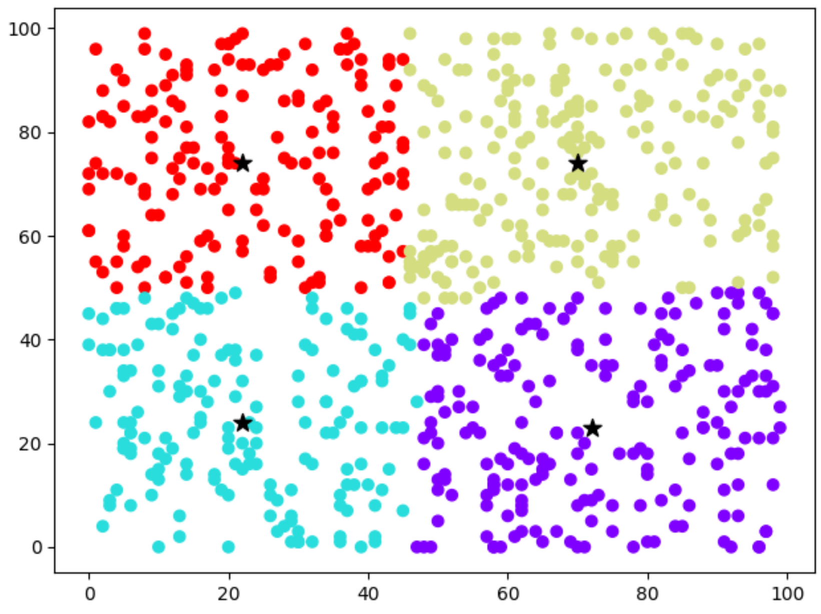


Figure 2.6: K Means Clustering Representation

2.4.5 Hierarchical Agglomerative Clustering vs K Means

One main difference between the two techniques is that K Means might not retrieve the same results if running the algorithm twice on the same data set while Hierarchical Agglomerative Clustering will always result the same data, and will even provide the history (dendrogram) of the constructed clusters.

Regarding the number of steps each algorithm will execute, the complexity of K Means is $O(n)$, while Hierarchical Clustering's is $O(n^2)$. While the complexity of K Means is lower, a disadvantage is that the certain number of iterations is not known, the algorithm stopping when there are no more changes in the centroids. The hierarchical clustering has a known number of iterations, $n - k - 1$, n being the number of elements, k being the number of clusters.

Another difference is the similarity measures used. K Means is used with Euclidean distance, while Hierarchical Agglomerative Clustering can be used with any similarity. The Euclidean distance is used in K Means because of the centroid calculations, based computing the mean of all the vectors in the cluster. On the other hand, Hierarchical Agglomerative Clustering can determine clusters with any given similarity.

2.5 Similarity Measures

In order to compute both the K Means and the Hierarchical Clustering algorithm, a similarity measure should be defined. This method should have as input two objects from the set of objects and compute how much of a resemblance they have, in order to add them into a cluster or another.

A frequently met similarity measure, specially in K Means algorithm, is the euclidean similarity, representing the euclidean distance between two n-dimensional vectors.

$$similarity(X,Y) = euclidean(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

A similarity measure that can be used in text summarization is the cosine similarity

$$similarity(X,Y) = cos(\theta(X,Y)) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} {x_i}^2}\sqrt{\sum_{i=1}^{n} {y_i}^2}}$$

Mathematically speaking, the dot product measures the angle between two n-dimensional vectors. In the n-dimensional space, the vectors may not be one near another, if we are taking in consideration the euclidean similarity, but they may still be similar. The angle between the two vectors is measured, and based by the size of the dot product result the similar objects are decided; the smaller the angle, the closer are the vectors considered.

The similarities and their implementation will be further explained in section 3.10.

3. Application A Summarizer for Romanian Texts

The purpose of this chapter is to describe the summarization algorithm and give a detailed explanation about how this algorithm is implemented.

3.1 Extractive Summarization Algorithm

This algorithm is designed to create an extract of a Romanian text by selecting its most important sentences. The **input** of the tool will be a text in Romanian, its title and the compression rate. The **output** will also be a text, having the length between 30% and 50% of the initial text. The summary will contain the exact same sentences from the initial text, but the algorithm will choose based on clustering the most important sentences. In order for the algorithm to have some conclusive results, the input text should have at least 10 sentences because having few sentences results in not having enough information about the significant words.

The summarization algorithm is represented in pseudocode in Algorithm 1 and represented visually in Figure 3.1. The implementation will be detailed in the next sections.

These are the steps of the summarization algorithm:

1. **Preprocessing**

   One of the preprocessing steps is represented by **finding the lemma of each word** of each sentence. In Romanian, the words have many forms according to cases, singular or plural forms,tenses etc, and it is important to consider each form of the word as being the most general one. For example, the words ″are″ and ″am″ should be considered as the same one i.e. ″a avea″. If the original form of the word, and not the lemma, is considered, the results of the words frequency will be incorrect and the algorithm will not be able to identify the most important words, as what could be considered an important word will have 3 forms, so instead of having one word with 3 occurrences, we will have 3 words,each with only one occurrence. For the lemmatization, the NLP-Cube framework is used, detailed in section 3.3.1.

2. **Compute terms frequency**

   The frequency of the terms used in the input text is necessary to determine the most important sentences. By **terms** we define **nouns** or **verbs**, in all their forms and declensions. It is important to use the lemmatization determined in the previous step for determining a correct term frequency.

3. **Select the most frequent words**

   From the terms frequency computed at step 2, the most frequent are considered to be the ones with at least two occurrences in the text. We denote as m the number of most frequent terms.

4. **Represent the sentences as vectors**

   The data cannot be clustered while being represented as a sequence of uneven length of words, so a transformation of the initial text into a mathematical one is required. The sentences will be represented as m-vectors (m being the number of most frequently used words, explained above), each value of the vector being the number of occurrences of each frequent word in that sentence.

5. **Add weights to sentences**

   Each sentence will have a corresponding weight/score which will help the algorithm choose the most important sentence. The first sentence of the text is usually an important sentence, so its score will receive extra points. Other important sentences are the ones which contain words included in the title or the sentences which contain proper nouns. The sentences which contain pronouns will not be considered as important because the noun to which they refer is not known, so their weight will decrease if these are encountered. The longest and shortest sentences will also receive a negative score, because short sentences do not give to much information, while long sentences can be difficult to follow in a summary. Short sentences are considered to have less than 5 words, while long sentences are considered to be over 30. The sentences in a summary should have medium length and be easy to understand.

6. **Clustering**

   After the sentences are transformed to vectors, the clustering algorithms can be applied. The number of clusters resulted will be the number of desired sentences in the summary based on the compression rate. The algorithm will return the the list of labels for the sentence representation, describing to which cluster does it belong to.

7. **Choose sentences from clusters**

   Based on the computed clusters, one sentence from each cluster will be chosen to be a part of the extractive summary. The selection is based on the sentence ranking; the sentence with the maximum rank in each cluster will be selected.

8. **Construct the extract**

   At this point, the algorithm has decided which sentences will use in the summary, but it needs to map the indexes of the vector representation to the actual initial sentences. Also, an important observation here is to construct the summary using the sentences in the initial order, so the resulted extract makes sense.

```
Input: input_text, input_title, compression_rate, method
Output: Automatic Extract
no_clusters ← computeNumberOfClusters(no_clusters,
  compression_rate);
most_frequent_terms, word_to_lemma ← termsFrequency(input_text,
  title);
sentences ← split(input_text);
representation ← vectorRepresentation(sentences,
  most_frequent_terms, word_to_lemma);
addWeightsToSentences(representation);
if method = 'hierarchical' then
    labels ← HierarchicalClustering(no_clusters, representation)
end
if method = 'kmeans' then
    labels ← KMeansClustering(no_clusters, representation);
end
summary ← constructSummary(labels, sentences);
return summary
```

Algorithm 1: Summarization Algorithm Pseudocode

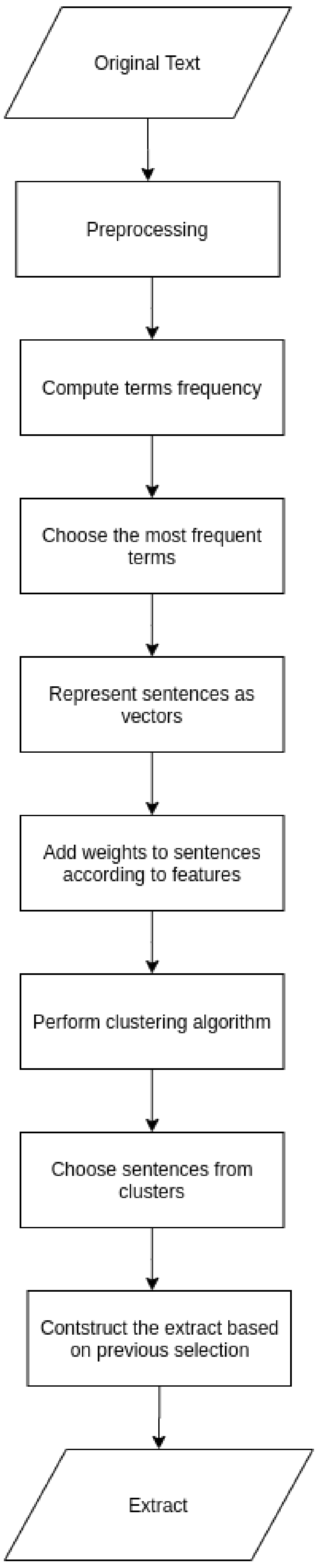

Original Text
Preprocessing
Compute terms frequency
Choose the most frequent terms
Represent sentences as vectors
Add weights to sentences according to features
Perform clustering algorithm
Choose sentences from clusters
Contstruct the extract based on previous selection
Extract

Figure 3.1: Extractive Summarization Algorithm Schema

3.2 Expected Input and Preparation

The tool will expect a text of minimum 10 sentences as input. In the preparation phase the number of clusters will be computed, based on a compression rate between 30% and 50% of the initial text. This step is performed here first of all, because the number of sentences in the summary needs to be equal to the number of clusters. Second of all, the number of sentences will possibly decrease because some sentences are removed before the clustering algorithm, so a later computation of the number of clusters will not be correct with respect to the initial text.

3.3 Preprocessing

The preprocessing step of the algorithm involves on one hand **determining the lemma and the part of speech** of each element in a sentence, on the other hand generating a **set of sentences** from the input text by splitting it. Lemmatization is the process of determining the dictionary form of a word, (e.g. have) given one of it's inflected variants (e.g. has, having, had) [3]. Lemmatization is important for categorizing correctly the words, for calculating later the term frequency. Determining the part of speech is also important, because not all parts of speech will be take into account in this algorithm. For example, sentences with a lot of adjectives and adverbs do not offer as much information as the ones with a lot of nouns and verbs in narrative texts, articles, etc, but they can be important in poetry or descriptive texts. Also, if a sentence contains many pronouns, it will not offer as much information as the sentence containing the nouns to which the pronouns refer.

3.3.1 NLP-Cube

NLP-Cube is an opensource Natural Language Processing Framework with support for languages which are included in the UD Treebanks [4]. NLP-Cube was used in this application for lemmatization. The tool is able to tokenize, lemmatize and tag the part of speech in an input text. The tool splits the text into sentences and then determine each world's lemma, part of speech and other attributes. The tool is customized for many languages, in this application the Romanian language is used. This is an example of a simple sentence being parsed by NLP-Cube:

```
from cube.api import Cube
cube = Cube(verbose=True)
cube.load("ro")

text = "Cand pisica nu-i acasa, soarecii joaca pe masa."
sentences = cube(text)
```

```python

    for entry in sentences:
        for word in entry:
            print(word)

```

Listing 3.1: NLP-Cube example

The results are:

Index	Word	Lemma	UPOS
1	Când	când	ADV
2	pisica	pisica	NOUN
3	nu	nu	PART
4	-i	fi	VERB
5	acasă	acasă	ADV
6	,	,	PUNCT
7	șoarecii	șoarece	NOUN
8	joacă	juca	VERB
9	pe	pe	ADP
10	masă	masă	NOUN
11	.	.	PUNCT

Table 3.1: NLP-Cube results caption

3.4 Terms Frequency

We define as **term** the lemma of a verb or a noun in the text. The pronouns are not taken into consideration here because they appear very often in texts, but it is difficult to understand sentences having pronouns without having a context for what the pronoun is referencing. For other types of texts, like poetry, literature or descriptive texts, adjective and adverbs could be considered terms, but it is not the case in this paper.

The most important aspect in calculating the term frequency is taking into consideration the form of the word that appears in the dictionary (lemma) and not the actual form that is present in the text. The Romanian Language has a lot of different forms for nouns according to case, genus, number and article, as well as for verbs for person, tense, mood. After retrieving the lemma of each word (using NLP-Cube tool detailed in Section 3.3.1) each word's frequency in each is computed. The function $f(t_k, S) = x$ is introduced to define the frequency of the terms, x being the number of occurrences in the sentence S of term t_k. Out of all the terms t_k, frequencies, the sum $\sum_{i=1}^{n} f(t_k, S_i)$ is computed, n being the number of sentences, and the result will be a dictionary having the key the term and the value the number of occurences.

From this dictionary, the m most common terms are chosen for further computation. A frequent term is a term which appears more than twice in the input text. The resulted dictionary has the form:

```
{
    'term_1': number of occurrences,
    'term_2': number of occurrences,
    ...
    'term_i': number of occurrences,
    'term_m': number of occurrences,
}
```

3.5 Vector Representation of Sentences

3.5.1 The Sentence Class

Because in this algorithm, a sentence has more attributes which must be kept in memory, a `Sentence` class is created. The `Sentence` class will have the following attributes: `id` - is the position of the sentence in the original text, starting form 1, `text` - the initial input text of the sentence, unchanged, `score` - the sentence's score, representation - the list representing the vector representation of the sentence and `label` - the label the sentence receives after the clustering algorithm.

3.5.2 Vector Representation

The next step is creating a numerical representation of the words in each sentence in an **m-dimensional array with integer values**. The length of each vector is the number of most frequent words, computed in 3.4. Each value of the vector will represent the frequency of a common word in the current sentence. A sentence S is represented as a vector $V = [f(t_1, S), f(t_2, S), ..., f(t_m, S)]$, where $f(t_k, S)$ represents the frequency of the term t_k in the sentence S. Every sentence is split into words, and the words will be compared to their lemma, computed with the help of the NLP-Cube tool.

Another step here is removing the zero-vectors. A zero-vector is a vector with only 0 values. In this context, a zero vector represents a sentence which does not have any common words in its composition. A sentence with no common words is not relevant to the summary and the similarities can not be computed with zero vectors. For example, the cosine similarity ($similarity(X, Y) = cos(\theta(X, Y)) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|} = \frac{\sum_{i=1}^{m} x_i y_i}{\sqrt{\sum_{i=1}^{m} {x_i}^2} \sqrt{\sum_{i=1}^{m} {y_i}^2}}$) does not accept a zero vector as input because a division by 0 will result.

3.6 Sentence Ranking

Each sentence which has a possibility of participating in the extractive summary will have an assigned weight. In the beginning, all weights start at 0. The weights are increased or decreased according to some features.

One of the features is the appearance of terms that are also **present in the title**. The title's words' lemmas are determined before any other computations, and for each word in the term frequency computation, its lemma is compared to the title's words lemmas. If a word in the title is also present in a sentence, the weight of the sentence will be increased by 5.

Another feature is the **presence of the proper nouns**. Usually, the proper nouns contain important information for an informative text, so the weight of the sentence will also be increased here, but only by 1.

The third feature taken into account in this algorithm is the **presence of the pronouns** in a sentence. A sentence having pronouns does not offer as much information as the sentence containing the noun the pronoun refers to, so the weight will be decreased by 0.5. The value is so small, with respect to the other two because usually sentences contain many pronouns and the presence of a proper noun is more important that the presence of a pronoun.

Usually, the **first sentence** in a text is an important one, so the sentence with id 1 will also receive a higher score, its weight being increased by 10.

The last feature taken into account is the **length of each sentence**. Usually ,the shortest sentences do not offer much information, so they will receive a negative score value. A short sentence is defined here as a sentence with less than 5 words. The longest sentences give much information, but they are usually hard to be followed by human readers. A summary should be short and concise, so their weight will be also decreased. A long sentence is defined here as a sentence with more than 30 words.

No.	Feature	Feature Score
1	Word's lemma in the title	+5
2	A Proper noun in sentence	+ 1
3	A Pronoun in sentence	-0.5
4	First sentence in the text	+ 10
5	Sentence with less than 5 words	- 2
6	Sentence with more than 30 words	- 2

Table 3.2: Features' Scores

Once the scores were calculated, a ranking is obtained by decrease ordering the sentences by their weights. This ranking is used in the selection of the representative sentences.

3.7 Clustering

After the sentences are transformed into vectors, the clustering can be made. The tool will perform 2 clustering algorithms, Hierarchical Agglomerative Clustering (detailed in section 3.8) and KMeans Clustering (detailed in section 3.9). Both clustering methods are represented by python objects which require as `__init__` parameters the number of clusters, the similarity and some input data, here being the sentence vector representation list. Both classes have a predict method which compute the labels for each sentence, each label representing the corresponding cluster to which the sentence belongs. The objects of the clustering can be used as in listing 3.2.

```
 # Hierarchical Clustering
 cluster = MyHierarchicalClustering(noClusters=noClusters, similarity=
cosine, input=vectorRepresentation)
 labels = cluster.predict()
 # KMeans Clustering
 cluster = MyKMeans(noClusters=noClusters, similarity=euclidean, input=
vectorRepresentation)
 (labels, centroids) = cluster.predict()

```

Listing 3.2: Usage of Clustering Objects

3.8 Hierarchical Agglomerative Clustering Implementation

This application uses the hierarchical agglomerative clustering with customizable similarity and single linkage. The clustering algorithm is all embedded in a python class. The main focus point of this class is the `predict` method which performs the clustering algorithm (Listing 3.3).

The first step is to compute the distances between the data points (the vectors representing the sentences). A distance is defined here as the similarity between two data points. A new list (`points`) is computed, having the vectors (V_i, V_j) positions and the distance between them. After the similarities are computed, they are sorted in ascending order for preparing the single linkage of the clusters. The single linkage imposes the distance between the clusters to be the distance between the closest points. For creating the clusters, a fixed number of `number of points - number of clusters - 1` operations are executed. Each of these operations will merge two clusters. In the beginning, each point is considered to be a separate cluster, and at each operation the algorithm will perform a check to verify if the points belong to different clusters, because if two points are already in a cluster they should not be merged again. After the merging process of the points, the result will be the tree roots of each cluster, as the clusters can be represented as a tree, so a normalization of the resulted label list should be performed. The normalization will transform the list's values into integer values between 0 and `noClusters`. This algorithm is an adapted form of Kruskal's Minimum Spanning Tree algorithm, which by having a weighted graph will compute a tree with all vertices and minimum cost. The difference here is just stoping after a number of operations and having multiple minimum spanning trees.

```python
    def predict(self):
        self.roots = [i for i in range(0, len(self.input))]
        self.treeDepth = [1 for _ in range(0, len(self.input))]
        points = []
        for i in range(len(self.input)):
            for j in range(i + 1, len(self.input)):
                distance = self.similarity(self.input[i], self.input[j
])
                points.append((distance, i, j))
        points.sort(key=lambda object: object[0])
        operations = len(self.input) - self.noClusters
        for distance, x, y in points:
            if self.getClusterRoot(x) != self.getClusterRoot(y):
                self.mergeClustersSingle(self.getClusterRoot(x), self.
getClusterRoot(y))
                operations -= 1
            if operations == 0:
                break
        notNormalizedResult = []
        for i in range(len(self.input)):
            notNormalizedResult.append(self.getClusterRoot(self.roots[i
]))
        normalizedResult = self.normalize(notNormalizedResult)
        return normalizedResult

```

Listing 3.3: Predict Method of Hierarchical Clustering

3.8.1 Testing

This algorithm always gives the same results and does not include random data, so it can be tested. For automated testing, the python **`unittest` framework** was used. This is a python version of JUnit, by Kent Beck and Erich Gamma and represents the standard unit testing framework for the language [25]. To check the correctness of the algorithm a python machine learning library was used. The **scikit-learn** [16] library is an open-source library with tools and algorithms for predictive data analysis, including Agglomerative Hierarchical Clustering algorithm. The source of truth in this testing is considered to be the results given by the scikit-learn AgglomerativeClustering object, which takes as parameters the number of clusters, the linkage type and affinity(similarity) and uses the `fit_predict` method to obtain the labels with parameter the data points as lists. The labels are compared with the labels obtained by the crated clustering. The testing has 2 test cases, performing clustering with 2 similarities, euclidean and cosine. The input data is represented by random list of lists, with a maximum number of points of 1000, a maximum number of elements in an array of 20 and numbers in the lists from 0 to 100.

3.9 KMeans Implementation

The KMeans clustering is also embedded in a python class having the focus point the predict method. As presented in the Hierarchical Agglomerative Algorithm, this method performs the clustering algorithm (Listing 3.4) and returns the normalized labels, but it will also return the centroids of the computed clusters.

The first step in performing the clustering algorithm is the initialization of the centroids. This is performed randomly, by choosing a point form the data set to represent a cluster. The `initializeCentroids` method will choose m random centroids, m being the number of clusters given as `__init__` parameter to the class. After the initialization step, the points are assigned to the closest cluster (using the euclidean similarity measure). After all the points are grouped in a cluster, the centroids are recalculated using the method `meanOfClusters`. This will take each newly computed cluster and calculate the mean between all the points assigned to the cluster. The computed means will become the new centroids and the points are reassigned to the new centroids. This step is performed until the old centorids and the new centroids are not changed anymore, or until a certain number of iterations (the chosen number here is 200).

```
def predict(self):
    self.centroids = self.initializeCentroids()
        for i in range(200):
         old_centroids = self.centroids
         self.assignPointsToClusters()
         new_centroids = self.meanOfClusters(old_centroids)
         if np.array_equal(old_centroids, new_centroids):
             break
         self.centroids = new_centroids
    normalized_labels = self.normalizeLabels(self.labels)
    return normalized_labels, self.centroids
```

Listing 3.4: Predict Method of KMeans Clustering

3.9.1 Testing

The KMeans algorithm cannot be tested using `unittest` testcases because the algorithm can have different outputs each time it is used. The differences in results are determined by the random assignment of the centroids in the first step of the algorithm.

3.10 Similarities Implementation

This algorithm uses three similarity measures: euclidean similarity, cosine similarity and a custom similarity. The euclidean similarity is used in KMeans, while the cosine and custom similarities are used in Hierarchical Agglomerative Clustering. All similarities methods take as input parameters `x` and `y` and will return a number representing the distance between them. `x` and `y` are lists representing an m-dimensional vector. As a structural detail, the similarities methods are organized in a separate file, and the methods are passed as `__init__` parameters to both KMeans class and Hierarchical Clustering class, so they can easily be changed. The similarities are not a constructing part of the clustering algorithm by itself, but they are an important tool to define the distances between points.

The euclidean and cosine similarities are implemented acording to their formula, $euclidean(X,Y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$, and $cosine(X,Y) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|} = \frac{\sum_{i=1}^{m} x_i y_i}{\sqrt{\sum_{i=1}^{m} {x_i}^2}\sqrt{\sum_{i=1}^{m} {y_i}^2}}$

The custom similarity measure is defined by:

$$similarity(X,Y) = \frac{\sum_{i=1}^{m} min(x_i, y_i)}{\sum_{i=1}^{m} max(x_i, y_i).}$$

This similarity computes the ratio of the number of frequent terms and the number of frequent terms that appear in the sentences represented by X and Y.

In all three similarities, $x_i = f(t_i, X)$, represents the number of occurrences of the term t_i in the sentence represented by the vector X.

3.11 Constructing the summary

After the clustering step, the labels for each sentence are retrieved. The labels are kept in the `Sentence` object as a number between 0 and m - 1, m being the number of clusters. One sentence from each cluster is chosen to represent a sentence from the extractive summary. This sentences are chosen based on the weights computed earlier, the sentence with the maximum weight in the cluster is chosen. After selecting the m sentences, they are sorted by `id` for assuring the chronological order of the sentences. It is important to keep the order of the sentences as they appeared in the initial text.

4. Implementation Details

This chapter focuses on presenting the used architecture, design and present the implementation details.

4.1 Architecture

The tool is a web application respecting the REST software architectural design. The **RESTful Web Services** allow communication with the client by receiving HTTP methods (as GET, POST, PUT, DELETE) along with some additional data and responding with the exact data the client needs for displaying the information to the user. The response is usually in a JSON (JavaScript Object Notation) format, but it can also be XML. A visual representation of REST architecture and data communication between the client and the server can be observed in Figure 4.1. This web application can be split in two main servers: the frontend and backend.

4.1.1 Backend

The **backend** represents the data access layer of an application. It usually it has access to the database to retrieve data and performs different kinds of computations with the data. In this application, the database layer does not exist because there is no need of storing any data. The backend here is represented by a Python webserver, created using Flask. **Flask** is a lightweight micro web application framework for Python [28]. This framework was chosen because of its simple setup and and its lack of constraints when

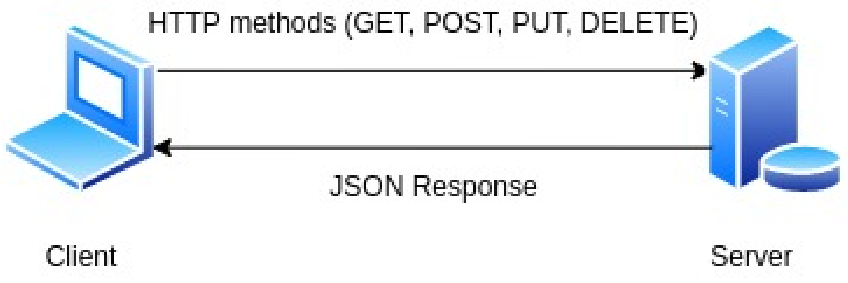


Figure 4.1: REST API Schema

it comes to architectural decisions such as having a database. Flask does not offer a database abstraction layer, form validation or other libraries [28], like other well-known Python web frameworks such as Django. The decision of Flask besides other framework was made based on the simplicity of the server.

The only purpose of the backend of the application is to retrieve the HTTP requests, and perform the summarization algorithm for sending back a response - the computed summary.

4.1.2 Frontend

The **frontend** represents the layer that transforms the data to a graphical interface that the user can understand. The frontend server receives JSON objects from the HTTP responses and displays them in a web page using HTML, CSS and mostly, JavaScript. For this web application, the React JavaScript library is used.

React is an open source JavaScript library for building user interfaces developed by Facebook [24]. It is highly used and appreciated in today's frontend development, with a 71.7% of developers who used it that would use it again according to [23]. The wonderfulness of React is JSX, a syntax extension to JavaScript that makes possible writing HTML-like text inside JavaScript code.

The purpose of this frontend is to send HTTP request to the backend, retrieving the data (the computed summary) and displaying it. For the user interface design, Material Design (design created by Google) was used with the usage of a popular React UI framework, **Material-UI** [27].

4.2 Design

4.2.1 Use Case Diagram

The application is only a tool to reflect the summarization algorithm, so it is a simple one, with two use cases. The focus point of the application is inputting a text with its title and retrieving its extractive summary. The other use case is just for demonstration purposes. The application stores a number of input text examples, and by selecting one the user can see the initial text and its summary. The use case diagram can be observed in Figure 4.2.

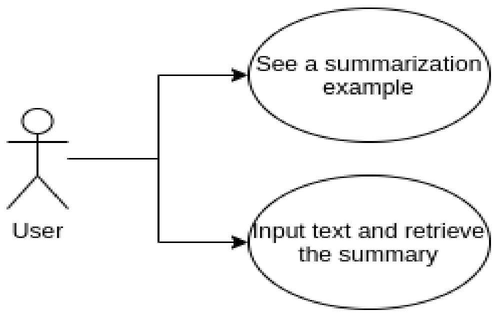

See a summarization example
Input text and retrieve the summary
User

Figure 4.2: Use Case Diagram

4.2.2 Class Diagram

The application has a simple architecture, the main class being the Algorithm. An algorithm object is used in the server. The Algorithm class uses a list of Sentence objects which contain every needed information about a sentence. According to the method requested by the user, the algorithm class will initialize in the `do` method a HierarchicalAgglomerativeClustering object or a KMeans object. The class diagram is visually represented in Figure 4.3.

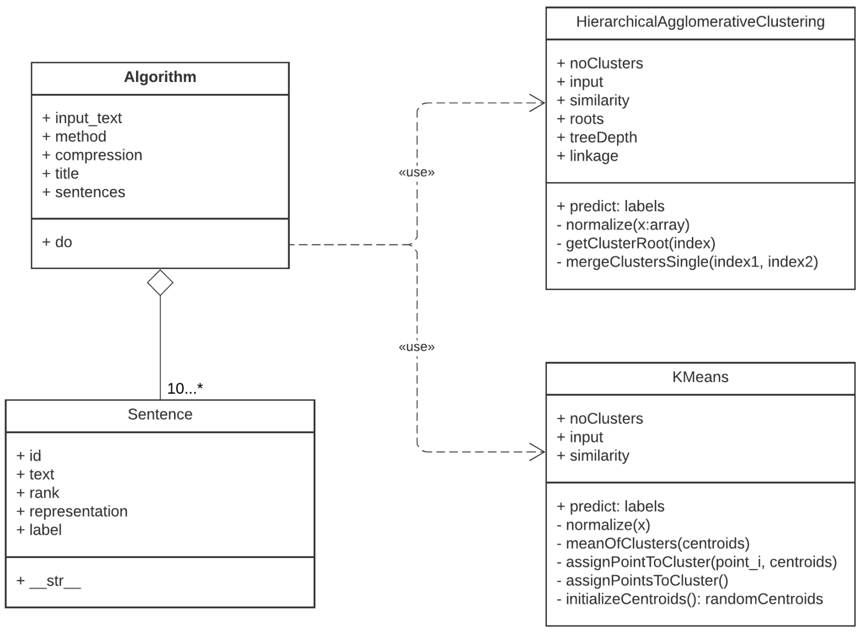


Figure 4.3: Class Diagram

4.2.3 Sequence Diagram

Figure 4.4 illustrates the application's sequence diagram, the frontend-backend communication and the classes included in the summarization algorithm. The user will input the text and send a request. The backend will decode the HTTP request and initialize an Algorithm object. The algorithm will perform the preprocessing, data representation, ranking and will send the data for clustering to a an object (Hierarchical Clustering or KMeans), depending on the requested method. The clustering object will return the clusters' labels and the algorithm object will decide which sentence belongs in the summary and which not according to the computed weights. At last, the backend server receives the summary, which is sent to the frontend in a HTTP response.

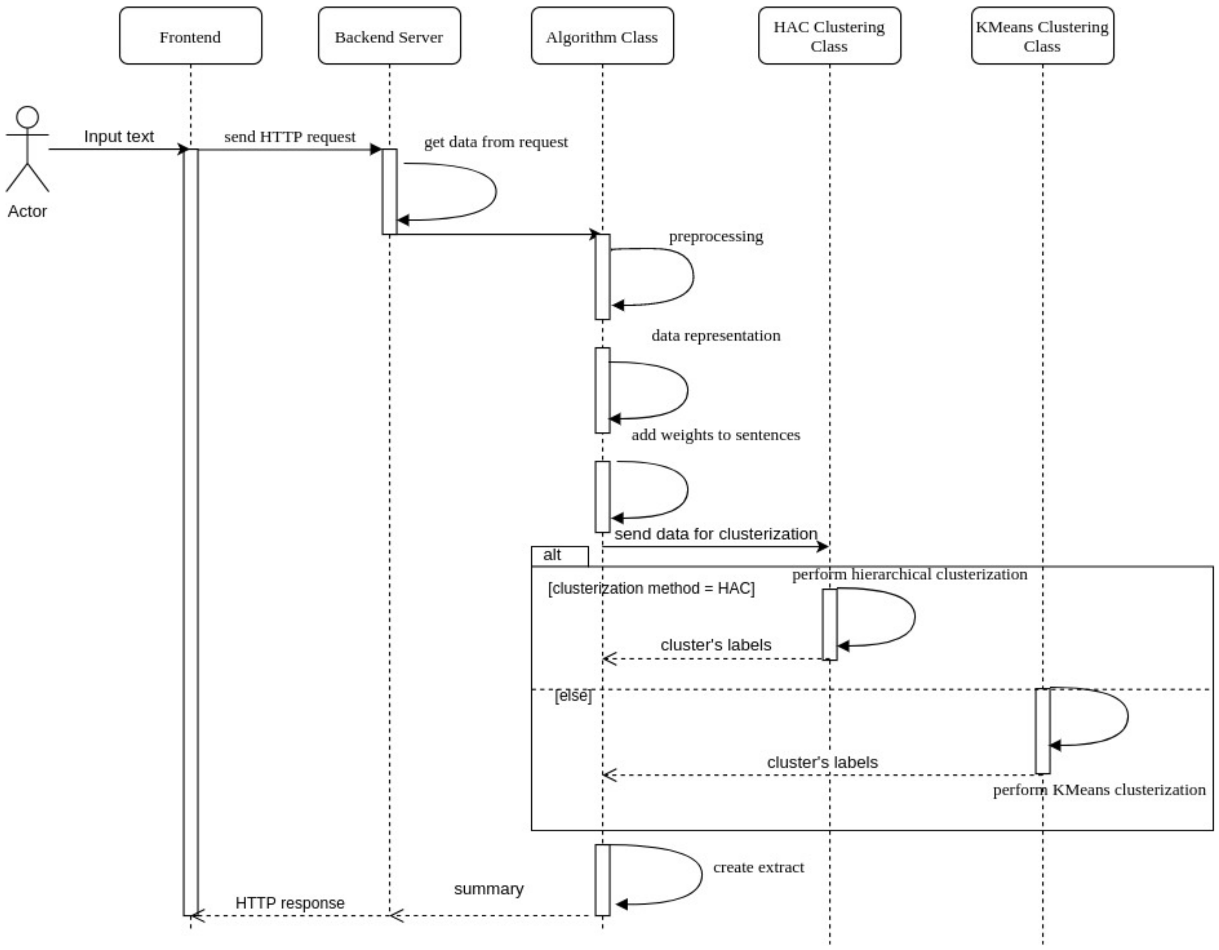


Figure 4.4: Sequence Diagram

4.3 Backend Implementation

The purpose of the backend is a very trivial one: to receive the HTTP request and send responses. The server contains only two accessible routes: `/summarize/{clusteringTechnique}/{compressionRate}` and `/summarize-example/inputId/{clusteringTechnique}/{compressionRate}`.

The first and most important one is a POST HTTP request, expecting as body an object containing the title and the input text. The method will create an Algorithm object which will perform the summarization, with respect to the request parameters `clusteringTechinque` and `compressionRate`, parameters which the user chooses in the frontend interface (detailed in Section 4.4). After the summarization algorithm, the extract is sent as the body of a HTTP response having a 200 status if everything is working as expected. The structure of the HTTP response is:

```
{
    data: {"summary": "The summary of the input text"},
    status: 200,
    Content-Type: application/json,
    Content-Length: 1234
}
```

The second route is a GET HTTP request for demonstration purposes. The user could choose an example number from the web interface and one of the example texts will be summarized and displayed along the input text and title. The `inputId` represents the number of the example input text. The other parameters have the same purposes. The structure of the HTTP response in this case is:

```
{
    data: {
        "input_text": "The input text",
        "title": "The title of the text",
        "summary": "The summary of the input text"
        },
    status: 200,
    Content-Type: application/json,
    Content-Length: 4321
}
```

4.4 Frontend Implementation

The frontend part of the application has two main tasks: ask for data and display data. In this case, the data is the summary of the input text.

For retrieving data from the server, only two of the eight HTTP methods are used: GET and POST. For the HTTP calls, the npm `axios` package is used. This is a promise based HTTP client fro the browser and node.js [22], which is useful for automatically transforming the data into JSON data and for working with promises in JavaScript which is a very important requirement and concept in this language.

In the main use case, meaning submitting an input text along its title, a POST call to the backend server is made. The request has two important components: the URL to which the request, `/summarize/{clusteringTechnique}/{compressionRate}` and the payload of the request, meaning the data transmitted throughout the network. The two parameters `clusteringTechnique` and `compressionRate` represent the user's preferences for the summary. `clusteringTechnique` can be Hierarchical Agglomerative Clustering or KMeans and `compressionRate` can be 30% or 50%. The payload is represented by a JSON object of form :

```
{
    "title": "The title of the input text",
    "input_text": "The input text as string"
}
```

.

5. Results

The purpose of this chapter is to present a detailed example of summarization and an analysis over automatic summarization vs manual summarization.

5.1 Example

This section contains a detailed example of a Romanian text being summarized using the summarization algorithm, with all the steps presented in the last chapters.

5.1.1 Input data

The input data is represented by the input text, its title and a compression rate of **30%**. The example will consider an extractive summarization using **Hierarchical Agglomerative Clustering**.

The example text in 5.1 is selected from [26].

Iulius Cezar

S1: Gaius Iulius Caesar a fost un lider politic și militar roman și una dintre cele mai influente și mai controversate personalități din istorie.

S2: Rolul său a fost esențial în instaurarea dictaturii la Roma, lichidarea democrației Republicii și instaurarea Imperiului Roman.

S3: A provocat războaie de cucerire fără acceptul senatului roman.

S4: Cucerirea Galiei, plănuită de Cezar, a inclus sub dominația romană teritorii până la Oceanul Atlantic.

S5: În anul 55 îHr Cezar a lansat prima invazie romană în Marea Britanie.

S6: Cezar a ieșit învingător într-un război civil, devenind dictator al lumii romane, și a inițiat o vastă acțiune de reformare a societății romane și a guvernării acesteia.

S7: El s-a proclamat dictator pe viață și a centralizat puternic guvernarea statului slăbit din cauza războiului civil pornit tot de Cezar.

S8: Prietenul lui Cezar, Marcus Brutus, complotează pentru a îl asasina, în speranța de a salva republica.

S9: Dramatica asasinare din Idele lui Marte a fost catalizatorul unui al doilea război civil, între cezari (Octavian, Marc Antoniu, Lepidus) și republicani (între alții, Brutus, Cassius și Cicero).

S10: Conflictul s-a încheiat cu victoria cezarilor în Bătălia de la Philippi și stabilirea formală a unui al Doilea Triumvirat, în care Octavian, Antoniu și Lepidus au preluat împreună controlul asupra Romei.

S11: Tensiunile iscate între Octavian și Antoniu au condus la un nou război civil, culminând cu înfrângerea lui Antoniu în Bătălia de la Actium.

S12: Octavian a ajuns liderul absolut al lumii romane.

S13: Perioada de războaie civile a transformat Republica Romană în Imperiul Roman, cu nepotul de bunic, în același timp și fiu adoptiv al lui Cezar, Octavian, cunoscut mai târziu ca Cezar August, instalându-se ca primul împărat.

S14: Campaniile militare ale lui Cezar sunt cunoscute în detaliu prin prisma propriilor sale consemnări: Comentarii de Bello Gallico.

S15: Multe detalii ale vieții sale au fost relatate mai târziu de istorici, precum Suetonius, Plutarh și Cassius Dio.

Input Text 5.1

5.1.2 Most Frequent Terms

The most frequent words are selected based on their lemma, determined using the NLP-Cube tool. The tools considers frequent, the terms that appear more than twice in the input text. Of course, the NLP-Cube is not perfect for the Romanian language, but its results are somewhat accurate. Here, the number of most frequent terms is 20. The list of most frequent terms is:

1. Cezar	6. lider	11. lume	16. cezar
2. Octavian	7. imperiu	12. roman	17. Lepidus
3. război	8. războaie	13. guvernare	18. Crassius
4. Antoniu	9. cucerire	14. viață	19. bătălie
5. republică	10. dictator	15. Brutus	20. detaliu

5.1.3 Representation and Ranking

The vectors are represented as m vectors, m being the number of most frequent terms. Each value from the vector is the frequency of one of the most frequent terms in the current sentence. For example, sentence one contains 'Cezar' once, 'lider' once and 'roman' once. The vectors corresponding to the input text 5.1 are:

S1: [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S2: [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S3: [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S4: [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S5: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S6: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0]
S7: [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
S8: [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
S9: [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0]
S10: [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0]
S11: [0, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
S12: [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
S13: [2, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
S14: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
S15: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1]

While constructing the vector representation, the ranking of each sentence is being computed, based on the given scores. Initially, all scores for features are 0 and values are added or subtracted, taking in consideration the criteria stated in 3.6. The corresponding scores are:

S1: 22.5	S4: 7	S7: 5.5	S10: 4.5	S13: 13
S2: 1	S5: 7	S8: 7.5	S11: 4	S14: 8
S3: 0	S6: 5.5	S9: 7.5	S12: 1	S15: 4

5.1.4 Clustering

After the sentences are represented as vectors, the clustering algorithm can be applied. In this example, the algorithm will be Hierarchical Agglomerative Clustering with cosine similarity, creating 5 clusters (30% compression rate) with single link. The resulted clusters are:

cluster C0: S1, S2, S3, S4, S7, S8, S9, S10, S11, S14, S15

cluster C1: S5

cluster C2: S6

cluster C3: S12

cluster C4: S13

In Figure 5.1 the history of the clusters creation can be observed in a dendrogram. The numbers represent the sentences' indexes. The clustering process stops when the 5 wanted clusters are formed.

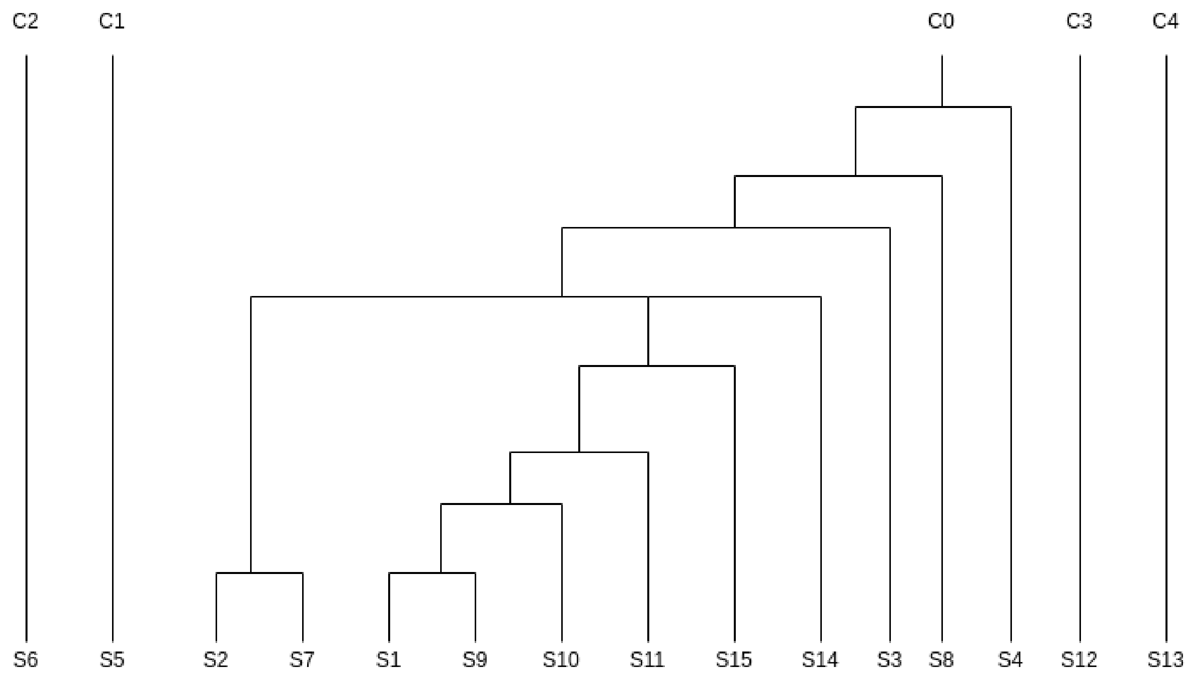


Figure 5.1: Clustering dendrogram

5.1.5 Summary

One sentence from each cluster is chosen to be a part of the summary. These sentences are chosen based on the scores; from each cluster the sentence with highest score is chosen to be the representative of the cluster and to be a part of the summary. The resulted sentences will be concatenated in the order of the initial text to form the extractive summary.

S1: Gaius Iulius Caesar a fost un lider politic și militar roman și una dintre cele mai influente și mai controversate personalități din istorie.
S5: În anul 55 îHr Cezar a lansat prima invazie romană în Marea Britanie.

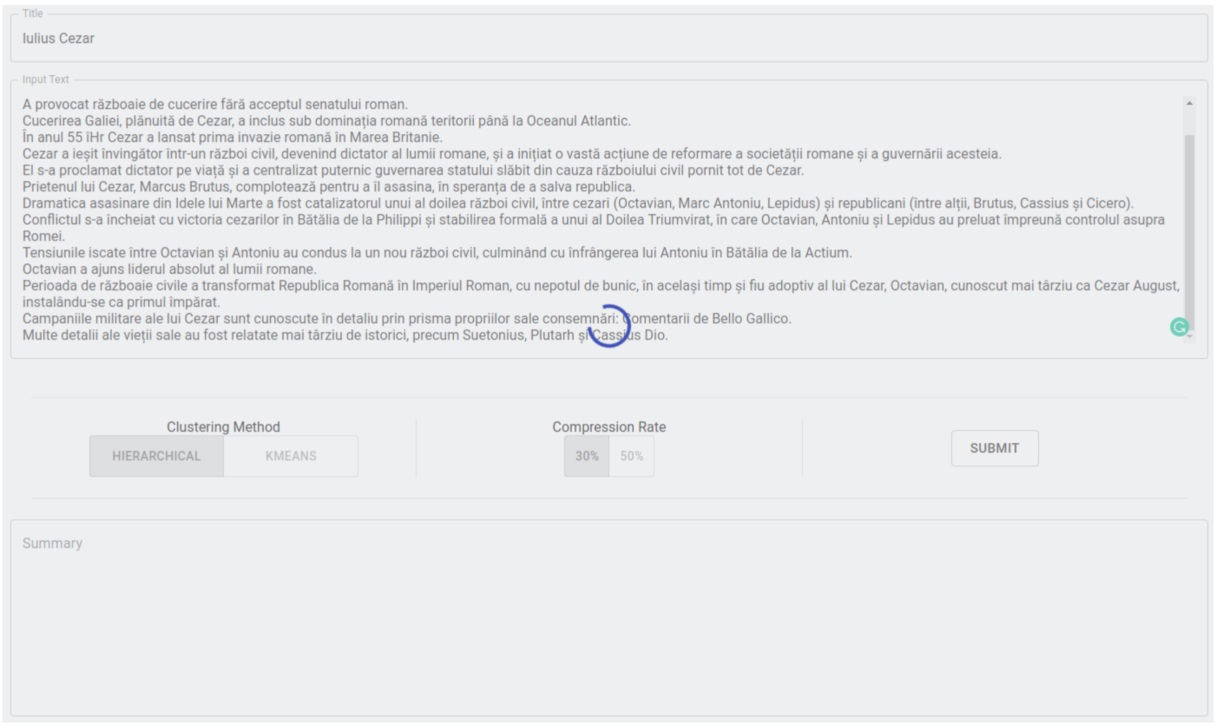
S6: Cezar a ieșit învingător într-un război civil, devenind dictator al lumii romane, și a inițiat o vastă acțiune de reformare a societății romane și a guvernării acesteia.
S12: Octavian a ajuns liderul absolut al lumii romane.
S13: Perioada de războaie civile a transformat Republica Romană în Imperiul Roman, cu nepotul de bunic, în același timp și fiu adoptiv al lui Cezar, Octavian, cunoscut mai târziu ca Cezar August, instalându-se ca primul împărat.

This extract was compared with two manual extracts with 80% similarity between each other. The automatic extract matched both manual extracts with a 60% similarity. The results interpretation is detailed in Section 5.3

5.2 User manual

The application is a trivial one, the user being able to insert the input text, its title and retrieve the automatic summary. For demonstration purposes, the user can select some existing examples and also select the technique and compression rate, in order to observe and compare the different results . The use cases of the application are illustrated in the Use Case diagram in Figure 4.2.

Figure 5.2 illustrates the text submission for summarization. The user will fill the title and input text and its preferences for summarization. Those are submitted and the user will wait for the summary response. For the example from Section 5.1 The summarization process will take approximately 45 seconds.



Title
Iulius Cezar

Input Text
A provocat războaie de cucerire fără acceptul senatului roman.
Cucerirea Galiei, plănuită de Cezar, a inclus sub dominația romană teritorii până la Oceanul Atlantic.
În anul 55 îHr Cezar a lansat prima invazie romană în Marea Britanie.
Cezar a ieșit învingător într-un război civil, devenind dictator al lumii romane, și a inițiat o vastă acțiune de reformare a societății romane și a guvernării acesteia.
El s-a proclamat dictator pe viață și a centralizat puternic guvernarea statului slăbit din cauza războiului civil pornit tot de Cezar.
Prietenul lui Cezar, Marcus Brutus, complotează pentru a îl asasina, în speranța de a salva republica.
Dramatica asasinare din Idele lui Marte a fost catalizatorul unui al doilea război civil, între cezari (Octavian, Marc Antoniu, Lepidus) și republicani (între alții, Brutus, Cassius și Cicero).
Conflictul s-a încheiat cu victoria cezarilor în Bătălia de la Philippi și stabilirea formală a unui al Doilea Triumvirat, în care Octavian, Antoniu și Lepidus au preluat împreună controlul asupra Romei.
Tensiunile iscate între Octavian și Antoniu au condus la un nou război civil, culminând cu înfrângerea lui Antoniu în Bătălia de la Actium.
Octavian a ajuns liderul absolut al lumii romane.
Perioada de războaie civile a transformat Republica Romană în Imperiul Roman, cu nepotul de bunic, în același timp și fiu adoptiv al lui Cezar, Octavian, cunoscut mai târziu ca Cezar August, instalându-se ca primul împărat.
Campaniile militare ale lui Cezar sunt cunoscute în detaliu prin prisma propriilor sale consemnări: Comentarii de Bello Gallico.
Multe detalii ale vieții sale au fost relatate mai târziu de istorici, precum Suetonius, Plutarh și Cassius Dio.

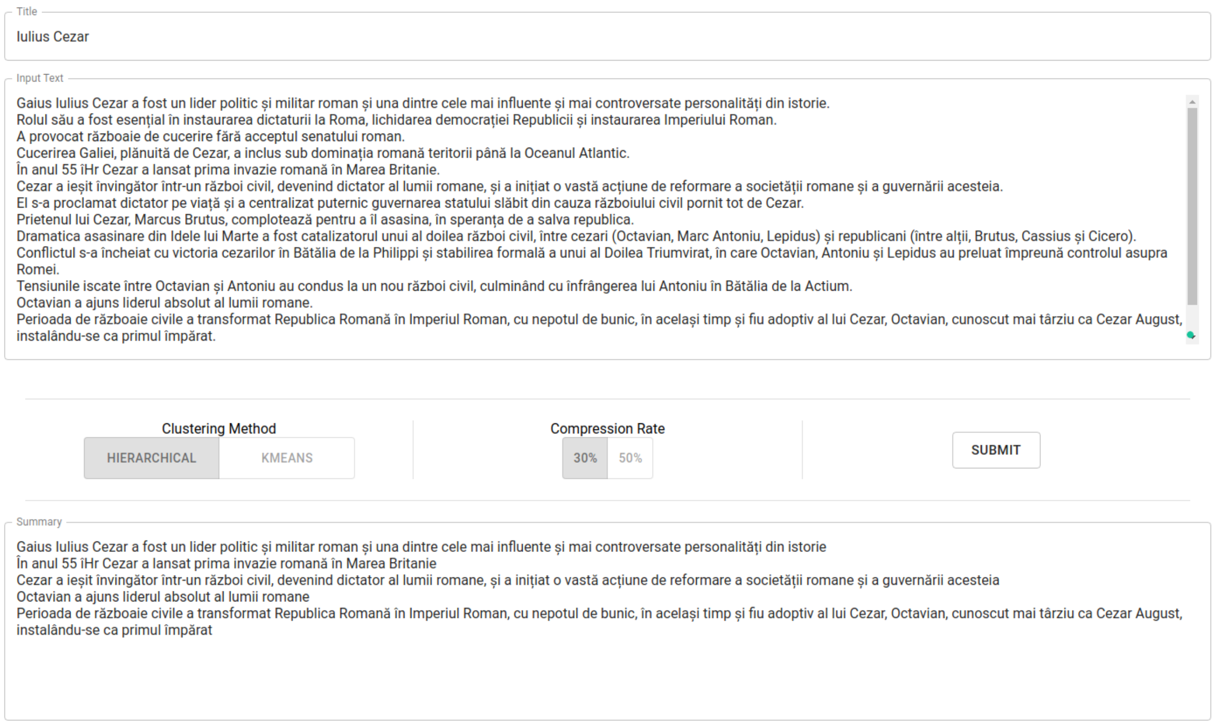
Clustering Method: [✓ HIERARCHICAL] KMEANS

Compression Rate: [✓ 30%] 50%

SUBMIT

Summary

Figure 5.2: Submitting a text for summarization



Title
Iulius Cezar

Input Text
Gaius Iulius Cezar a fost un lider politic și militar roman și una dintre cele mai influente și mai controversate personalități din istorie.
Rolul său a fost esențial în instaurarea dictaturii la Roma, lichidarea democrației Republicii și instaurarea Imperiului Roman.
A provocat războaie de cucerire fără acceptul senatului roman.
Cucerirea Galiei, plănuită de Cezar, a inclus sub dominația romană teritorii până la Oceanul Atlantic.
În anul 55 îHr Cezar a lansat prima invazie romană în Marea Britanie.
Cezar a ieșit învingător într-un război civil, devenind dictator al lumii romane, și a inițiat o vastă acțiune de reformare a societății romane și a guvernării acesteia.
El s-a proclamat dictator pe viață și a centralizat puternic guvernarea statului slăbit din cauza războiului civil pornit tot de Cezar.
Prietenul lui Cezar, Marcus Brutus, complotează pentru a îl asasina, în speranța de a salva republica.
Dramatica asasinare din Idele lui Marte a fost catalizatorul unui al doilea război civil, între cezari (Octavian, Marc Antoniu, Lepidus) și republicani (între alții, Brutus, Cassius și Cicero).
Conflictul s-a încheiat cu victoria cezarilor în Bătălia de la Philippi și stabilirea formală a unui al Doilea Triumvirat, în care Octavian, Antoniu și Lepidus au preluat împreună controlul asupra Romei.
Tensiunile iscate între Octavian și Antoniu au condus la un nou război civil, culminând cu înfrângerea lui Antoniu în Bătălia de la Actium.
Octavian a ajuns liderul absolut al lumii romane.
Perioada de războaie civile a transformat Republica Romană în Imperiul Roman, cu nepotul de bunic, în același timp și fiu adoptiv al lui Cezar, Octavian, cunoscut mai târziu ca Cezar August, instalându-se ca primul împărat.

Clustering Method: [✓ HIERARCHICAL] KMEANS

Compression Rate: [✓ 30%] 50%

SUBMIT

Summary
Gaius Iulius Cezar a fost un lider politic și militar roman și una dintre cele mai influente și mai controversate personalități din istorie
În anul 55 îHr Cezar a lansat prima invazie romană în Marea Britanie
Cezar a ieșit învingător într-un război civil, devenind dictator al lumii romane, și a inițiat o vastă acțiune de reformare a societății romane și a guvernării acesteia
Octavian a ajuns liderul absolut al lumii romane
Perioada de războaie civile a transformat Republica Romană în Imperiul Roman, cu nepotul de bunic, în același timp și fiu adoptiv al lui Cezar, Octavian, cunoscut mai târziu ca Cezar August, instalându-se ca primul împărat

Figure 5.3: Summarization Result

5.3 Results interpretation

For checking the tool, 2 persons have created **manual extracts** of 15 randomly selected Romanian texts. The chosen text has a title and between 10 and 50 sentences. Each person has created 2 extracts for each text: one with a 30% compression rate, one with a 50% compression rate.

5.3.1 Manual Extracts Similarities

The persons did not know how the algorithm worked and they were also not inspired one by the other, the results being completely unbiased. Clearly, a rigorous evaluation should be made with a lot more texts and a lot more persons.

The summaries were influenced by each person's personality and way of thinking, so they were not, of course, the same. As table 5.1 shows, the two manual extracts had an average of 42.59% similarity for the 30% extracts and 60.19% similarity for the 50% extracts.

Text no.	Number of sentences	Similarity for compression rate = 30%	Similarity for compression rate = 50%
1	15	80.00%	62.5%
2	19	16.66%	40.00%
3	49	22.66%	68.00%
4	14	50.00%	71.42%
5	45	35.71%	65.21%
6	13	75.00%	44.44%
7	31	44.44%	71.42%
8	47	28.57%	66.66%
9	18	40.00%	44.44%
10	13	25.00%	42.85%
11	35	36.35%	38.88%
12	29	33.33%	86.66%
13	26	50.00%	53.84%
14	17	40.00%	55.55%
15	14	57.14%	66.66%
Average		42.59%	60.19%

Table 5.1: Manual Extracts Similarities

5.3.2 Manual Extracts vs Automatic Extracts

5.3.2.1 Hierarchical Agglomerative Clustering, single link

For each input text, the automatic extracts were computed using the summarization algorithm and the results were compared with the manual extracts. Tables 5.2 and 5.3

present the percentages of similarity between the generated automatic extract and the manual extract for 30% and 50% compression rate.

Text No.	No. of sentences	No. of sentences summary	Similarity Percentage to Person 1	Similarity Percentage to Person 2
1	15	5	60.00%	60.00%
2	19	6	50.00%	33.33%
3	49	15	33.33%	33.33%
4	14	4	50.00%	50.00%
5	45	14	42.86%	35.71%
6	13	4	50.00%	75.00%
7	31	9	55.56%	44.44%
8	47	14	42.86%	35.71%
9	18	5	20.00%	40.00%
10	13	4	50.00%	50.00%
11	35	11	36.36%	45.45%
12	29	9	11.11%	22.22%
13	26	8	62.50%	25.00%
14	17	5	60.00%	20.00%
15	14	7	42.86%	57.14%
Average			44.49%	41.82%
Total Average: 43.15%				

Table 5.2: Similarities for compression rate = 30% Hierarchical Clustering

Text no.	No. of sentences	No. of sentences summary	Similarity Percentage to Person 1	Similarity Percentage to Person 2
1	15	8	50.00%	62.5%
2	19	10	60.00%	60.00%
3	49	25	44.00%	44.00%
4	14	7	42.85%	71.42%
5	45	23	56.52%	56.25%
6	13	7	57.14%	50.00%
7	31	16	62.50%	71.42%
8	47	24	50.00%	50.00%
9	18	9	55.55%	66.66%
10	13	7	71.42%	57.14%
11	35	18	61.11%	55.55%
12	29	15	60.00%	60.00%
13	26	13	69.23%	53.84%
14	17	9	66.66%	66.66%
15	14	12	50.00%	66.66%
Average			57.13%	59.91%
Total Average: 58.52%				

Table 5.3: Similarities for compression rate = 50% Hierarchical Clustering

5.3.2.2 Single Link vs Complete Link

The purpose of clustering is to categorize the similar sentences in the same cluster, so only the main ideas are selected in the summary and the redundancy is avoided.

The summarization algorithm only performs the Hierarchical Agglomerative Clustering with single link, but an evaluation with complete link (described in Section 2.4.3) was made using the Hierarchical Agglomerative Clustering from *Scikit-learn: Machine Learning in Python* library [16].

Table 5.4 show the percentages of the sentences appearing in different clusters. The reason for this analysis is to determine if the sentences picked by the two persons are in different clusters defined by the algorithm.

	HAC - single link				HAC - complete link			
	30%		50%		30%		50%	
Text No.	Person 1	Person 2	Person 1	Person 2	Person 1	Person 2	Person 1	Person 2
1	60.00	60.00	50.00	75.00	80.00	80.00	75.00	75.00
2	50.00	50.00	70.00	70.00	66.67	66.67	60.00	80.00
3	40.00	33.33	56.00	60.00	60.00	60.00	60.00	76.00
4	60.00	75.00	57.14	85.71	75.00	50.00	71.43	85.71
5	42.86	35.71	56.52	56.52	64.29	57.14	69.57	65.22
6	75.00	100.00	71.43	85.71	75.00	75.00	71.43	71.43
7	60.00	44.44	62.50	56.25	77.78	77.78	68.75	62.50
8	42.86	35.71	50.00	50.00	64.29	64.29	62.50	62.50
9	50.00	40.00	55.56	66.67	80.00	80.00	66.67	66.67
10	75.00	50.00	85.71	57.14	75.00	75.00	71.43	85.71
11	45.45	54.55	61.11	55.56	63.64	63.64	77.78	66.67
12	11.11	33.33	60.00	66.67	66.67	77.78	73.33	80.00
13	62.50	25.00	69.23	53.85	62.50	50.00	69.23	61.54
14	60.00	20.00	66.67	66.67	80.00	60.00	66.67	77.78
15	28.57	42.86	50.00	58.33	57.14	57.14	58.33	75.00
Average	50.89	46.66	61.46	64.27	69.86	66.30	68.14	72.78
	Total Average Single Link: 55.82%				Total Average Complete Link: 68.54%			

Table 5.4: Percentages of Single link vs Complete link

5.3.2.3 KMeans Clustering

Tables 5.5 and 5.6 presents the comparison between the manual extracts and the automatic extracts of one run of the algorithm using Kmeans clustering on the initial texts for a 30% and 50% compression rate.

Text No.	No. of sentences	No. of sentences summary	Similarity Percentage to Person 1	Similarity Percentage to Person 2
1	15	5	60.00%	60.00%
2	19	6	50.00%	33.33%
3	49	15	40.00%	13.33%
4	14	4	50.00%	25.00%
5	45	14	42.86%	35.71%
6	13	4	25.00%	50.00%
7	31	9	55.56%	33.33%
8	47	14	35.71%	28.57%
9	18	5	20.00%	40.00%
10	13	4	50.00%	75.00%
11	35	11	45.45%	27.27%
12	29	9	44.44%	22.22%
13	26	8	25.00%	50.00%
14	17	5	40.00%	20.00%
15	14	7	42.86%	57.14%
Average			41.79%	38.06 %
Total Average: 39.92%				

Table 5.5: Similarity Percentages of KMeans Clustering with compression rate = 30%

Text no.	Number of sentences	Number of sentences summary	Similarity Percentage to Person 1	Similarity Percentage to Person 2
1	15	8	62.50%	62.50%
2	19	10	50.00%	50.00%
3	49	25	56.00%	44.00%
4	14	7	28.57%	71.42%
5	45	23	60.86%	52.17%
6	13	7	42.85%	57.14%
7	31	16	50.00%	50.00%
8	47	24	58.33%	50.00%
9	18	9	66.66%	55.55%
10	13	7	28.57%	28.57%
11	35	18	50.00%	44.44%
12	29	15	60.00%	66.66%
13	26	13	46.15%	76.92%
14	17	9	77.77%	44.44%
15	14	12	50.00%	50.00%
Average			52.55%	53.59%
Total Average: 53.07%				

Table 5.6: Similarity Percentages of KMeans Clustering with compression rate = 50%

6. Conclusions

Although automatic text summarization had its beginning in the late 1950s and many approaches were provided since, most of them were for English texts, the Romanian Language being somehow neglected.

The purpose of this paper was to analyze automatic text summarization, create a tool for this task for Romanian texts and integrate it in a web application. This paper has researched an important Natural Language Processing task, compared different clustering algorithms and evaluated the resulted autommatic extractive summaries of different text with manual extracts created by two persons for different compression rates.

6.1 Reflections

This application has combined the most important subjects studied in the last three years of the bachelor programme, like data structures and algorithms, artificial intelligence - mainly clustering algorithms, web programming, working with git, choosing modern tools for implementation.

I believe that using new frameworks is important in a programmer's life, so I made every effort to implement and use some modern and suitable frameworks and programming languages, like React and Python.

As a domain that was not studied in the last three years, Natural Language Processing, and particularly the summarization task was researched in this paper.

Even if a summary can hardly be evaluated because of the type of text, the person's subjective perspective and personality, the results of the automatic summaries compared to manual summaries are encouraging.

6.2 Further Development

Although the summarization algorithm works, it is far from perfect and improvements can always be made.

One improvement can be done in the evaluation process. 2 persons and 15 texts are not enough to determine the correctness of the algorithm; more input texts and more persons to create manual extracts would generate a more concrete and closer to reality conclusion.

Despite the fact that every language has its own individuality and the summarization process should be, from a certain point, particular for the language, the proposed approach could be extended on different languages, by some minor changes in the preprocessing phase.

Also, more features can be added to the summarization algorithm for an improvement in the ranking. For example, searching domain specific words, or looking for specific phrases such as 'în concluzie', 'deci' etc which can define an important sentence.

Another improvement can be made on the clustering algorithms, more algorithms can be implemented and their results can be evaluated on manual extracts.

From a web application point a view, the next steps would deploying the application and register a domain name for it, in order for actual users being able to utilize it.

Bibliography

Books & Articles

[1] Navneet Kaur Amandeep Kaur Mann. "Review Paper on Clustering Techniques". In: *Global Journal of Computer Science and Technology* (2013). ISSN: 0975-4172. URL: https://computerresearch.org/index.php/computer/article/view/353.

[2] Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. "Information Fusion in the Context of Multi-Document Summarization". In: *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*. ACL '99. College Park, Maryland: Association for Computational Linguistics, 1999, pp. 550–557. ISBN: 1558606093.

[3] Toms Bergmanis and Sharon Goldwater. "Context Sensitive Neural Lemmatization with Lematus". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 1391–1400. URL: https://www.aclweb.org/anthology/N18-1126.

[4] Tiberiu Boros, Stefan Daniel Dumitrescu, and Ruxandra Burtica. "NLP-Cube: End-to-End Raw Text Processing With Neural Networks". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 171–179. URL: https://www.aclweb.org/anthology/K18-2017.

[5] Ronald Brandow, Karl Mitze, and Lisa F. Rau. "Automatic Condensation of Electronic Publications by Sentence Selection". In: *Inf. Process. Manage.* 31.5 (Sept. 1995), pp. 675–685. ISSN: 0306-4573.

[6] Ernesto D'Avanzo, Bernardo Magnini, and Alessandro Vallin. "Keyphrase extraction for summarization purposes: The LAKE system at DUC-2004". In: (Jan. 2004).

[7] H. P. Edmundson. "New Methods in Automatic Extracting". In: *J. ACM* 16.2 (Apr. 1969), pp. 264–285. ISSN: 0004-5411.

[8] Som Gupta and S.K Gupta. "Abstractive Summarization: An Overview of the State of the Art". In: *Expert Systems with Applications* 121 (Dec. 2018).

[9] Udo Hahn and Inderjeet Mani. "The Challenges of Automatic Summarization". In: *IEEE Computer* 33 (2000), pp. 29–36.

[10] E. H. Hovy. *Automated Text Summarization. In R. Mitkov (ed), The Oxford Handbook of Computational Linguistics*. 2005.

[11] Elena Lloret. "TEXT SUMMARIZATION : AN OVERVIEW". In: (2008). URL: https://www.dlsi.ua.es/~elloret/publications/TextSummarization.pdf.

[12] H. P. Luhn. "The Automatic Creation of Literature Abstracts". In: *IBM Journal of Research and Development* 2.2 (1958), pp. 159–165.

[13] Kathleen R. McKeown et al. "Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster". In: *Proceedings of the Second International Conference on Human Language Technology Research*. HLT '02. San Diego, California: Morgan Kaufmann Publishers Inc., 2002, pp. 280–285.

[14] Kathleen McKeown and Dragomir R. Radev. "Generating Summaries of Multiple News Articles". In: *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '95. Seattle, Washington, USA: Association for Computing Machinery, 1995, pp. 74–82. ISBN: 0897917146.

[15] Thiago Pardo, Lucia Rino, and Maria Nunes. "GistSumm: A Summarization Tool Based on a New Extractive Method". In: June 2003, pp. 210–218.

[16] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[17] J. J. Pollock and A. Zamora. "Automatic Abstracting Research at Chemical Abstracts Service". In: *Journal of Chemical Information and Computer Sciences* 15.4 (1975), pp. 226–232.

[18] Dragomir Radev et al. "NewsInEssence: A System For Domain-Independent, Real-Time News Clustering and Multi-Document Summarization". In: *Human Language Technology Conference. San Diego, USA* (Sept. 2001).

[19] Mooi E Sarstedt M. *Cluster Analysis. In: A Concise Guide to Market Research. Springer Texts in Business and Economics*. Springer, Berlin, Heidelberg, 2019, pp. 301–315.

[20] Marko Sarstedt and Erik Mooi. "Cluster Analysis". In: *A Concise Guide to Market Research: The Process, Data, and Methods Using IBM SPSS Statistics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 301–354. ISBN: 978-3-662-56707-4.

[21] Krysta Svore, Lucy Vanderwende, and Christopher Burges. "Enhancing Single-Document Summarization by Combining RankNet and Third-Party Sources". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 448–457. URL: `https://www.aclweb.org/anthology/D07-1047`.

Online Resources

[22] *Axios Documentation*. 2020. URL: `https://www.npmjs.com/package/axios` (visited on 05/22/2020).

[23] Raphaël Benitte, Sacha Greif, and Michael Rambeau. *State of JS*. 2019. URL: `https://2019.stateofjs.com/front-end-frameworks/overview/` (visited on 05/22/2020).

[24] Facebook. *React Documentation*. 2020. URL: `https://reactjs.org/docs/getting-started.html` (visited on 05/22/2020).

[25] Python Software Foundation. *Python Documentation*. 2020. URL: `https://docs.python.org/2/library/unittest.html#` (visited on 05/03/2020).

[26] *Iulius Cezar*. 2020. URL: `https://ro.wikipedia.org/wiki/Iulius_Cezar` (visited on 05/22/2020).

[27] Material-UI. *Material-UI Documentation*. 2002. URL: `https://material-ui.com/` (visited on 05/22/2020).

[28] Armin Ronache. *Flask Documentation*. 2015. URL: `https://palletsprojects.com/p/flask/` (visited on 05/22/2020).