IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 29 June 2023, accepted 4 August 2023, date of publication 31 August 2023, date of current version 15 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3310818

RESEARCH ARTICLE

A Comparative Study on R Packages for Text Mining

CARLOS J. HELLÍN, ADRIÁN VALLEDOR, JUAN J. CUADRADO-GALLEGO, ABDELHAMID TAYEBI AND JOSEFA GÓMEZ

Computer Science Department, University of Alcalá, 28805 Alcalá de Henares, Spain

Corresponding author: Josefa Gómez (josefa.gomezp@uah.es)

This work was supported in part by the Program "Programa de Estímulo a la Excelencia para Profesorado Universitario Permanente" of Vice Rectorate for Research and Knowledge Transfer of the University of Alcalá, and in part by the Comunidad de Madrid (Spain) under Project EPU-INV/2020/004.

ABSTRACT The term *Text Mining*, which is given to the set of techniques used for the extraction, cleaning and processing of the information in texts, has become useful to provide valuable information to other algorithms and widely used with statistical and machine learning methods. By enabling the extraction of useful insights from textual data, Text Mining has become a potent tool in decision-making and knowledge discovery across many areas, including health care, government, education and industry. R is a mature open-source programming environment that has overstepped its initial scope of application for statistical computing and graphics to be used in pretty all the Data Science knowledge Area Groups. The objective of this paper is to present review and benchmarking analysis of packages for text mining techniques with **R** in computational systems. The paper reviews thirteen different packages comparing them on their execution time and memory used, for which new tests have been specifically designed. The results of this approach have been intended to be used over the most common tasks carried out when analyzing texts, and comparisons included allow R users to know which packages are best for each task and to improve their performance. Text mining package (**tm**) stands out particularly in Tokenization and Stemming techniques, while **fastTextR** is the best choice for Topic Modeling and Normalization. Also in the case of the Term Frequency-Inverse Document Frequency (TF-IDF) technique, the **textir** package is a clear choice. The other packages will depend on whether the technique is applied to a document-term matrix (DTM) or to plain text. In addition, there are packages that perform better in runtime than in memory usage and vice versa, making the choice more difficult. Packages such as **udpipe** can achieve better results working in parallel. Future works will include the same analysis for parallel computing, hybrid approaches, and novel algorithms.

INDEX TERMS Text mining, natural language processing, information retrieval, benchmark, R.

I. INTRODUCTION

This paper focuses on the Text Mining process, which involves techniques for extracting, cleaning, and processing information for analysis. Following the Data Science Body of Knowledge developed in the EDISON project [11], Text Mining is in the Text Data Mining Knowledge Area, KA01.04 (DSDA.04/TDM). This is the fourth knowledge area of the six areas of the Data Analytics Knowledge Area Group, KAG1-DSDA. Following the definition of the body of knowledge, text analytics applies statistical, linguistic and structural techniques to extract and classify information from textual sources, a species of unstructured data. It is composed of the following eight knowledge units, KUs: 1. KU1.04.00. General overview and main concepts in text data mining; 2.KU1.04.01. Text analytics including statistical, linguistic and structural techniques to analyse structured and unstructured data; 3. KU1.04.02. Data mining and text analytics; 4. KU1.04.03. Natural language processing; 5. KU1.04.04. Predictive models for text; 6. KU1.04.05. Retrieval and clustering of documents; 7. KU1.04.06. Information extraction; and 8. KU1.04.07. Sentiments analysis. This research has been done over those KUs.

The associate editor coordinating the review of this manuscript and approving it for publication was Arianna Dulizia.

In several disciplines, including computer science, business, finance, artificial intelligence, and others, Text Mining has a substantial impact on decision-making, knowledge discovery and information extraction. By providing significant insights from enormous volumes of textual data that were previously challenging to examine, Text Mining techniques have changed the process of knowledge discovery. Information retrieval, document classification, document clustering, and automatic summarization are a few examples of practical text mining applications. These applications aid in effective data retrieval by classifying documents according to their content, assembling related documents and producing succinct summaries.

This study's comparative analysis of **R** Text Mining packages is what makes it novel. Although there have been other publications and books that include discusses Text Mining in R, this research offers a full evaluation of package performance, including execution time and memory usage. By directly contrasting various **R** packages and analyzing their capabilities and performance, the comparative analysis used in this study fills a research gap. Thus, the research offers understanding into the benefits and drawbacks of each package, assisting users in selecting the best package. As a result, academics and professionals can use the most potent tools at their disposal while also adding to the body of knowledge in the field of Text Mining.

The comparisons in this study are based on techniques in Natural Language Processing, Knowledge Representation, Information Extraction, and Sentiment Analysis. Other areas such as Information Retrieval, Document Classification, Document Clustering, and Automatic Summarization have not been considered in the comparisons.

During the last years, research has explored the use of text mining techniques with R. One of the first packages available was **kernlab**, published in 2004, by [27]. This package allows creating and computing with string kernels used for clustering or classifying data for text mining. Some time later, other main packages were released such as **lsa** [62], in 2005, which performs latent semantic analysis. The package **tm** [16], in 2007, which provides a comprehensive text mining framework for **R**. Some of the other packages that stand out and that will also be analyzed in this article are the following: **openNLP** [24], released in 2008, which serves as an interface to Apache OpenNLP, **tau** [8], in 2009, which offers utilities for text analysis and, **textir** [55], in 2011, which allows using multinomial inverse regression also for text analysis. Other packages have been published in recent years and will also be tested in this paper: **textstem** [47], which provides tools for stemming and lemmatizing text, **fastTextR** [48] for text classification and **udpipe** [61], which is a natural language processing toolkit. All of these last packages were released in 2007 and all packages described are available within the Comprehensive R Archive Network (CRAN) repository.

Some of the early papers describing the use of R to perform text analysis are [27], which introduces the **kernlab** package that provides to the **R** user with basic kernel functionality. In [17], which introduces the **tm** package and explains how typical text mining tasks can be carried out with this framework. Feinere [15] gives an introduction of text mining with R, giving a brief overview of the package **tm** package. While Theu et.al [56] conducted a runtime study of typical text mining tasks with the use of a plug-in for **tm**. These results are then compared to the parallel computing approach implemented in the **tm** package. Particularly in the last few years, more on text mining in **R** is found in the literature. Written by [52], this book, Text Mining with R: A Tidy Approach, serves as an introduction to text mining using the **tidytext** package and other tidy tools. Recently, one of the articles published by [19] presents a collection of character string/text/natural language processing tools for, among other techniques, pattern searching, string collation and sorting, normalization, transliteration, and formatting that are found in text mining.

This study contributes to the existing knowledge repository by providing a comprehensive analysis and benchmarking of Text Mining packages in R, focusing on their performance, as execution time and memory usage. The paper's contributions are:

- The techniques of Text Mining, which include information extraction, cleaning and processing from texts, are the main topic of this study.
- Techniques for Natural Language Processing, Knowledge Representation, Information Extraction and Sentiment Analysis are specifically examined in the study.
- The comparisons and analyses are based on the pertinent knowledge units found in the Text Data Mining Knowledge Area.
- The study examines and evaluates thirteen distinct Text Mining **R** packages, including **tm**, **fastTextR** and **udpipe**, contributing to the existing knowledge in the field.

This paper is organized as follows. Section II describes the selected Text Mining techniques whose **R** packages will be analyzed and compared. Section III presents the **R** packages with a description of each and the techniques they use. Section IV evaluates the methodology used to obtain the results. Section V presents all the results obtained and highlights the best and worst packages. Finally, Section VI presents the conclusions obtained from this comparative study.

II. TEXT MINING

Taking into account the spread of their use, nine very well-known and used Text Mining techniques have been selected, for which the set of R packages has been analyzed and compared. Those ones are: Tokenization, Stopwords, Lemmatization, Stemming, Part of Speech Tagging, Term Frequency-Inverse Document Frequency, Normalization, Topic Modelling, and Sentiment Analysis. In the following all of them are introduced.

A. TOKENIZATION

Tokenization, or the identification of fundamental units that do not need to be decomposed in further processing, is the initial step in text mining [60].

This technique incorporates a text into tokens, which are words, sentences, or other elements. Tokenization produces a list of tokens that may be scanned to find the words in a sentence. This list of tokens is then utilized as input for additional processing, such as parsing or text mining. Tokenization is useful not only in linguistics, but also in computer science, where it is used in lexical analysis.

The tokenization of documents is one of the needs of a parser or text mining, because retrieving information requires the words from the dataset. However, several issues remain, such as eliminating punctuation marks and dealing with other characters like parentheses and hyphens, as well as situations like abbreviations and acronyms.

Depending on the language, there may be more issues. The majority of words in English, French, and Spanish, for example, are separated by spaces. Other languages, such as Chinese, do not have clearly defined word boundaries. This has an impact on the tokenization process because it necessitates more information on the language vocabulary and morphology. The spelling and typographical structure of the words will also influence tokenization.

The structure of languages can be grouped into three categories [21]:

- **Isolating:** Words do not divide into smaller units. Example: Mandarin Chinese
- **Agglutinative:** Words divide into smaller units. Example: Japanese, Tamil
- **Inflectional:** Boundaries between morphemes are unclear and ambiguous in terms of grammatical meaning. Example: Latin.

B. STOPWORDS

Common terms like 'and', 'are', 'this', and so on are frequently employed as stopwords. They are useless when it comes to document classification. As a result, they must be deleted. The refinement of this list of stopwords, on the other hand, is difficult and inconsistent across literary sources. This method also increases system performance by reducing text data. These terms are dealt with in each text document and are not required for text mining applications [36]. Stopwords are removed from documents using a variety of methods:

1) THE CLASSIC METHOD

The classic method relies on the removal of stopwords from pre-compiled lists. Several lists can be found in the literature [18], [59].

2) METHODS BASED ON ZIPF'S LAW (Z-METHODS)

Apart from the classic method, there are methods based on Zipf's law, such as deleting the most frequent terms (TF-High), removing single words (TF1), and removing words with a low inverse document frequency (IDF). The terms in each data set are sorted according to their frequencies to determine the amount of words in the stop lists generated by the previous approaches (or the inverse frequencies of those in the IDF method) [36].

3) THE MUTUAL INFORMATION METHOD (MI)

The mutual information (MI) [10] is a supervised method for calculating the information between a given term and a document class in order to estimate how much information the term may provide about a particular class. When mutual information is low, the phrase is likely to have low discriminating power and should be deleted.

A term t and a class c are represented by two random variables, and mutual information is calculated between them [66]:

$$I(T;C) = \sum_{t\in T}\sum_{c\in C} P(t,c)\log_2\left(\frac{p(t,c)}{p(t)p(c)}\right) \tag{1}$$

where $I(T;C)$ is the mutual information between T and C, $T = \{0,1\}$ is the set of terms that appear ($T = 1$) or do not appear ($T = 0$) in a given document, and $C = \{0,1\}$ is the set of classes that the document belongs to ($C = 1$), or does not belong to ($C = 0$) [36]. The equation uses the probability of a term and a category occurring together $P(t,c)$, the probability of a term occurring alone $p(t)$, and the probability of a category occurring alone $p(c)$.

4) TERM BASED RANDOM SAMPLING (TBRS)

Reference [34] were the first to suggest this method for manually detecting stopwords in web texts. This method iterates across distinct bits of data that are randomly picked. The Kullback-Leibler divergence measure is then used to rank phrases from each chunk depending on their formatting values [10]:

$$d_x(t) = P_x(t)log_2\left(\frac{P_x(t)}{P(t)}\right) \tag{2}$$

$P_x(t)$ is the normalized frequency of a term t within a mass x, while $P(t)$ denotes the normalized frequency of t throughout the entire collection. The final list is made up of the least informative terms from all the chunks, with any possible duplications removed [36].

C. LEMMATIZATION

Many text mining applications require lemmatization as a preprocessing step. It is also employed in natural language processing and a variety of other linguistics-related applications. In addition, it is a good technique to come up with generic search engine keywords or labels for concept maps [41].

This approach is a morphological transformation that removes the inflectional ending of a word in the text and converts it to a base or dictionary form of the word, known as a lemma. In the case of a noun, the lemma corresponds to the singular form, in the case of a verb, the infinitive

form, and in the case of an adjective or adverb, the positive form. Lemmatization reduces the complexity of the studied text by reducing the total number of different terms, providing significant benefits to downstream text processing components [32].

For searching and indexing, lemmatization requires additional dictionary support, which improves its accuracy in feature extraction applications. Bio Lemmatizer is used to lemmatize biological material [32], whereas Word Net Lemmatizer with Word Net Database is utilized to lookup for lemmas [2]. In their research, [3] employed World Net to collect product features, whereas Concept Net [33] was used to extract detail on innovative technical items.

D. STEMMING

This technique is used to determine a word's root/stem. The words connect, connected, connecting, and connections, for example, can all be traced back to the word ''connect'' [44]. The goal of this strategy is to eliminate multiple suffixes, minimize the number of words, ensure that stems are precisely matched, and save time and memory space.

Stemming is the process of translating a word's morphological forms to its stem, presuming that they are semantically connected. When utilizing a stemmer, there are two things to keep in mind:

- Words with different meanings should be kept distinct.
- Morphological forms of a word should be mapped to the same stem because they are presumed to have the same basic meaning.

In text mining or language processing applications, these two principles are sufficient. Stemming is commonly thought of as a method for improving recall. The strength of stemming is lower in languages with a simple morphology than in languages with a more complicated morphology. The majority of stemming experiments to date have been conducted in English and other west European languages.

Truncating methods, statistical methods, and mixed methods are the three main types of stemming algorithms [50]. Each of these groups has its own method for locating the stems of word variations.

1) TRUNCATING METHODS

As the name implies, these approaches are concerned with deleting a word's suffixes or prefixes (often referred to as affixes) [50]. The Truncate (n) stemmer is the simplest basic stemmer. It truncates a word at the nth symbol, keeping n letters and removing the rest. Words shorter than n are preserved in this strategy. When the word length is short, the likelihood of over stemming increases. The S-stemmer - an algorithm that combines the singular and plural forms of English nouns - was another basic technique. Donna Harman came up with the idea for this algorithm [22]. The algorithm provides rules for removing plural suffixes and converting them to single versions.

2) STATISTICAL METHODS

These are the stemmers who use statistical tools and analysis. The majority of approaches remove the affixes, but once a statistical procedure is applied, the affixes are removed [50].

3) MIXED METHODS

a: INFLECTIONAL AND DERIVATIONAL METHODS

This is a different technique to stemming that takes into account both inflectional and derivational morphology. These sorts of stemmers require a huge corpus to evolve, hence they are included in corpus base stemmers as well. Word variants in inflectional are related to language-specific syntactic changes such as plural, gender, case, and so on, whereas word variants in derivational are related to the part-of-speech (POS) of a sentence where the word occurs [25].

b: CORPUS BASED STEMMER

Xu and Croft suggested this approach of stemming in their paper ''Corpus-based stemming utilizing co-occurrence of word variants'' [65]. They have an optional technique that seeks to address some of Porter stemmer's flaws.

Automatic alteration of conflation classes - words that have resulted in a common stem - to suit the characteristics of a specific text corpus using statistical approaches is referred to as corpus based stemming. This method has the advantage of potentially avoiding inappropriate conflations for a given corpus, and the outcome is an actual word rather than an unfinished stem. The downside is that statistical measures have to be constructed for each corpus independently, and processing time increases because two stemming techniques are employed first before employing this method in the first phase.

c: CONTEXT SENSITIVE STEMMER

This is an intriguing way of stemming because, unlike the traditional approach of stemming before indexing a document, for a Web Search, context-sensitive analysis is performed on the query side using statistical modeling. Reference [39] proposed this approach.

Before the query is submitted to the search engine, the morphological variations that might be relevant for the search are predicted for the terms of the input query. This drastically minimizes the frequency of incorrect expansions, lowering the cost of further computation while also improving precision. Following the extraction of the query's predicted word variations, these variants are subjected to context-sensitive document matching. This conservative approach protects against erroneous stemming, and it turns out to be crucial for enhancing precision.

E. PART OF SPEECH (POS) TAGGING

Parts of Speech Tagging is a technique of performing Semantic Analysis that involves allocating a word to one of the parts of speech. Nouns, verbs, adverbs, adjectives, pronouns, conjunctions, and their subcategories are all parts of speech.

Speech tagging is separated into two categories: supervised and unsupervised techniques, with each kind further classified.

Tagging is a procedure in natural language processing (NLP) that provides labels to linguistic components. It refers to the process of assigning part-of-speech tags to texts. A tagger is a computer application that is used for this purpose. The technique of assigning one of the parts of speech to a given word is known as part of speech tagging. The English term rust, for example, can be used as a verb or a noun. The following are some different types of speech tagging [38]:

1) SUPERVISED AND UNSUPERVISED TAGGINGS

A pre-tagged corpus (organized collection of text) is used for training to learn information about the tagset, word-tag frequencies, rule sets, and other things. Training and prediction are the two basic phases of supervised classification. Feature extractors generate different class labels during training by converting each input value to a feature set or class label. In order to produce a model, a pair of feature sets and class labels are input into a machine learning algorithm during training. The same feature extractor is utilized for producing predicted labels for unseen input or test set during the prediction phase. Pre-tagged corpora are not required for unsupervised Part of Speech (POS) tagging models. It works by taking a POS lexicon as input, which is a collection of possible POS tags for each word [23].

2) RULE BASED AND STOCHASTIC TECHNIQUES

Stochastic tagging is a phenomenon that involves frequency or probability. To assign tags to unknown or ambiguous words, rule-based approaches leverage contextual and morphological information. For example, if the ambiguous/unknown word X is preceded by a determiner and followed by a noun, identify it as an adjective [7]. The tagger can either infer these rules automatically or the designer can encode them. Eric Brill created the most well-known rule-based component of a voice tagger, which was the first to achieve an accuracy level comparable to stochastic taggers, i.e. 95%-97% [63].

F. TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY (TF-IDF)

The Term Frequency-Inverse Document Frequency (TF-IDF) statistic illustrates how essentially a word is to a document in a collection. In information retrieval and text mining, the TF-IDF is frequently employed as a weighting factor. The value of TF-IDF rises in proportion to the number of times a word appears in a document, but is offset by the word's frequency in the corpus. This can aid in controlling the fact that some terms are more commonly used than others. Stop-words filtering with TF-IDF can be done successfully in a variety of disciplines, including text summarization and categorization. The frequency and inverse document frequency statistics are combined to form TF-IDF. The number of times each term appears in each text is recorded and added together to further identify them. The number of times a phrase appears in a document is known as Term Frequency (TF) [35].

$$tf(t,d) = 0.5 + \frac{0.5 * f(t,d)}{maximum\ occurences\ of\ \ words} \tag{3}$$

Inverse Document Frequency (IDF) is a statistical weight for determining the significance of a term in a text document collection. The IDF function is used to reduce the weight of terms that appear frequently in the document set, while increasing the weight of terms that appear infrequently.

$$idf(t,d) = log\frac{|D|}{(no.of\ \ documents\ term\ t\ \ appears)} \tag{4}$$

This formula is then used to calculate Term Frequency - Inverse Document Frequency (TF-IDF) for each word.

$$tfidf(t,d,D) = tf(t,d) * idf(t,d) \tag{5}$$

The frequency of occurrence of phrase t in document d is denoted by d in equations 3 and 4. Equation 5 uses Term Frequency (Tft, d) and Inverse Document Frequency (idft, d) to calculate TF-IDF for each term in the document.

G. NORMALIZATION

The process of cleaning or removing extraneous data from a large collection of collected data is referred to as normalization in sentiment analysis. Data preprocessing is used to eliminate noise from retrieved text, making it more consistent and understandable. Data must be preprocessed before going to the analysis step in any text mining method, therefore the extracted data is preprocessed for subsequent processing.

H. TOPIC MODELING (TM)

Topic Models are generative models in Natural Language Processing that give a probabilistic framework. Topic modeling approaches are commonly used to organize, interpret, search, and summarize huge electronic archives automatically. The term ''topics'' refers to the unknown variable relationships that exist between words in a lexicon and their appearance in texts. A document is viewed as a collection of several themes. Topic models look for hidden themes across the collection and annotate texts with those topics. Each term is thought to be derived from one of those themes. Finally, a document coverage distribution of topics is constructed, which gives a new approach to look at data from a subject viewpoint [57].

1) LATENT SEMANTIC ANALYSIS (LSA)

The theory of knowledge acquisition, induction, and representation is known as latent semantic analysis (LSA) [30]. Reference [12] were the first to introduce it as an information retrieval (IR) technique. It is a machine-learning technique for assessing the relationships and similarity structures between texts and phrases that do not rely on human experience, prior theoretic models, semantic dictionaries, or knowledge bases [29].

The primary goal of LSA is to reduce the size of a dataset, which is accomplished through a matrix operation called singular value decomposition (SVD). SVD is a technique for dividing a matrix into three smaller matrices. The resulting reduced-order SVD gives the best k-dimensional approximation to the original matrix in the least square error sense [13] by keeping the k of the biggest singular values. Two sets of factor loadings are generated as a result of SVD, one for words and one for documents. In the same latent semantic space created by the SVD, each term and document is represented as a k-dimensional vector. As a result, each latent semantic factor is now linked to a set of high-loading terms and documents [51]. To understand and label the corresponding factor, high-loading phrases and documents are used. Before SVD computation, the number of factors is an input parameter that must be specified. LSA organizes key terms or documents into several levels of aggregation as the number of components changes. When employing a collection of representative articles to identify major subjects in a discipline, a higher level of aggregation indicates key study areas, while a lower level of aggregation represents generic research themes [51].

The LSA analysis can be broken down into three phases. The first step is to create a term-document matrix, with each row representing a key word or term and each column representing a document or context where the key word appears. The frequency of a key word in the corresponding document is represented by an entry in the matrix. The second step is to use various weighting strategies to turn the term frequencies in a term-document matrix. The third step is to use SVD to lower the dimensionality of the matrix, which is a critical element of the LSA approach. Only the k of the greatest solitary values are kept in this step. The best k-dimensional approximation to the original matrix is the reduced-order SVD [13].

Extensive testing has shown that LSA's classification performance is reliable [5] and that it can infer relationships from text [29]. Information retrieval (IR), search optimization, classification, clustering, filtering, and other IR-related applications can all benefit from it [13].

I. SENTIMENT ANALYSIS (SA)

People's thoughts, sentiments, estimations, and attitudes regarding entities and their properties in textual data are analyzed using sentiment analysis [31]. Unstructured text is handled in sentiment analysis, which causes several computer processing issues. Various strategies and procedures are employed in order to solve difficulties. Image recordings [64] or a mix of image and text [58] can be used for sentiment analysis. Sentiment analysis may be thought of as a process that begins with the establishment of a goal and progresses through the following steps: data retrieval, preprocessing, feature extraction, text categorization according to sentiment, interpretation of the results, and, finally, presentation. The most common method of classification is based on polarity or orientation. It is a two-step method (two binary classifications) in which text is first classified according to objectivity (objective text does not contain sentiment and is thus excluded from further analysis), then it is classified into a positive or negative category. Another option is to split text into one of three categories (positive, neutral, or negative) all at once. Sentiment analysis, rather than identifying a subjective text just as positive or negative, may also be done according to particular emotions such as surprise, fear, disgust, delight, and trust.

The text unit across which the analysis is carried out might be:

- **Document**: analysis determines if a positive or negative attitude about the subject of the document predominates (e.g. film review).
- **Sentence**: because a sentence may be considered a tiny document, there is no significant difference between sentiment analysis at the document and sentence levels.
- **Aspect**: the analysis is focused on some of the entity's most essential qualities or portions. It is common in restaurant or product reviews.

Words are the most common means of conveying emotion, however phrases can also be used. It is possible to assign numeric values to each of these classifications in addition to the fundamental categorization of the word in positive, negative, or neutral, as in the SentiWordNet dictionary, where each word is given three decimal values, the entire sum of which is one [14].

III. PACKAGES FOR TEXT MINING WITH R

This section discusses the **R** packages which are available at CRAN and related to text processing for further analysis. For this purpose, a description of each of the frameworks or plugins will be made. The thirteen Text Mining R packages for which their performance over the selected techniques has been analyzed and compared are: udpipe, textstem, openNLP, text2vec, lsa, tm, textmineR, textir, tau, fastTextR, sentimentr, cleanNLP, and RSentiment. In the following, all of them are introduced. They have not been classified taking into account the techniques described in the previous section because pretty all of them allow applying most of the techniques. The packages were selected based on their popularity and functionality in the R community. They have established themselves as trustworthy and efficient tools for a variety of text processing tasks. For researchers, data scientists, and practitioners working with textual data in R, the combination of these package offers a wide range of functions and methodologies for text analysis. There are numerous other helpful NLP-related software in addition to the ones already selected. Some of them are mentioned below. The **topicmodels** [20] package offers an interface to Correlated Topics Models (CTM) and Latent Dirichlet Allocation (LDA) models for determining topics in texts. Another noteworthy package is **tidytext** [53], which offers word processing, sentiment analysis, and tools

for text mining. Also, the package **svs** [40], which provides straightforward implementations of several methods for semantic vector spaces, including correspondence analysis, latent class analysis, probabilistic latent semantic analysis, non-negative matrix factorization, and latent semantic analysis.

A. UDPIPE

The **udpipe** package [61] is a trainable pipeline performing sentence segmentation, tokenization, POS tagging, lemmatization and dependency parsing [54]. It uses the *Universal Dependencies project (UD)* as its main information source, whose goal is a multi-language relation in morphology and semantics. Currently, this project has 70 dependency trees on 50 different languages.

B. TEXTSTEM

The package **textstem** [47] provides a tool set that stems and lemmatizes text. For lemmatization, a dictionary is created from a text using the function `+make_lemma_dictionary()+`, which by default uses the **hunspell** [37] package as the engine. Other packages such as **lexicon** [45] and the TreeTagger tool can be used. Both R packages are faster than TreeTagger and do not require installation of external tools, but are less accurate. In contrast, using TreeTagger requires installing the program which is available for Linux, Windows, Mac-OS and ARM.

C. OPENNLP

The **openNLP** [24] package provides to **R** with an interface to the Apache OpenNLP tools. The Apache library is a Machine Learning-based toolkit for Natural Language Processing written in Java. This package supports tokenization, segmentation, segmentation, part-of-speech tagging, named entity extraction, chunking, parsing and coreference resolution.

D. TEXT2VEC

R has a feature that is copy-on-modify and, because of this feature, forming a large-scale DTM or TCM can be a serious bottleneck for analysts and researchers. By causing the entire collection of documents to be read into Random Access Memory (RAM), and processed as a single vector, it increases memory usage by a factor of 2 to 4.

The **text2vec** [49] package solves this issue by providing a better way to construct a DTM or TCM, using a list of iterators that traverse the text in multiple theards to form them. The user must provide the iterators list and division to be performed on the text.

E. LSA

The **lsa** [62] package provides the tools to perform Latent Semantic Analysis on texts with R. It allows the use of conceptual indexes that are a way of statistical derivation for value decomposition (texts of high conceptual order possess polysemy, among other factors, thus allowing for the reduction of synonyms, etc., in a matrix of terms associated with a document).

F. TM

The **tm** [16] package provides a framework for Text Mining in R, which allows applying a multitude of existing methods on text. It is flexible and allows integration with other packages and extensions, making it possible to use advanced methods in Text Mining such as whitespace removal, stemming, or stopword deletion.

G. TEXTMINER

The **textmineR** [26] package was designed to assist in the Text Mining process, with a syntax familiar to R users. It tries to maximize interoperability in the R ecosystem, to be scalable in space and computation time, and to have an R idiomatic language. It has extra topic model analysis and diagnostics features.

textmineR offers scalability thanks to the use of matrices such as dgcMatrix, which allow significant savings in terms of memory, the use of the **Rcpp** package to speed up R and the use of parallel processing when possible (through the `TmParallelApply()` function, which is syntactically common with respect to Windows and Linux).

This package offers simplicity in exchange for performance. It is designed to be run on a single node in a cluster, although, if available, it will use all the cores on that node by default. To improve performance, the underlying **text2vec** package is used.

H. TEXTIR

The **textir** [55] package allows the use of multinomial inverse regression by inference on texts and associated attributes. A minimalistic approach to the partial least squares routine is also included. This package allows the creation of a matrix based on term occurrence frequency (TF-IDF).

I. TAU

The **tau** [8] package has utilities for text analysis, such as document loading and preprocessing (tokenization and stopwords), as well as allowing the counting of terms or patterns in documents.

J. FASTTEXTR

The package **fastTextR** [48] provides an interface to the fastText library, available in Python. It allows word representation learning and supervised text classification.

K. SENTIMENTR

The **sentimentr** [46] package is designed to quickly calculate the polarity of sentiment that exists in a text at the sentence level, as well as allowing aggregation by rows or by grouping variables. The goal of **sentimentr** is to try to strike a balance between efficiency and effectiveness.

TABLE 1. Runtime (milliseconds) for tokenization.

R Package	Documents		
	10	100	1000
udpipe	950	3,270	27,940
text2vec	5	10	20
tm (Boost)	10	10	210
tm (MC)	10	60	600
tm (Scan)	10	10	180
tau	100	880	8,930
cleanNLP	10	190	1,875

TABLE 2. Memory usage (MB) for tokenization.

R Package	Documents		
	10	100	1000
udpipe	3.97	4.07	5.06
text2vec	3.89	3.91	4.36
tm (Boost)	3.92	3.98	5.99
tm (MC)	3.92	4.11	6.64
tm (Scan)	3.92	3.98	5.74
tau	3.92	4.26	6.96
cleanNLP	3.96	4.45	13.30

L. CLEANNLP

cleanNLP [1] is an R package that provides a set of tools to convert a corpus into a set of normalized tables for further analysis. To do so, **cleanNLP** makes use of the **udpipe** package, the Python spaCy library or the Java CoreNLP library as a backend.

M. RSENTIMENT

The **RSentiment** [6] package allows extracting sentiment information from English texts. It uses underlying NLP, text analysis and stemming. When computing the value of each sentence, it takes into account sarcasm, negations, various degrees of adjectives and emotions.

IV. EVALUATION METHODOLOGY

Execution time and memory use were the two primary performance indicators that were the subject of the relevant studies' evaluation metrics [9], [28]. These measurements are essential for determining how effectively the packages under comparison use their resources.

1) Execution Time: The amount of time it takes each package to complete particular tasks is measured by execution time. It offers information about the effectiveness and speed of the packages in handling different procedures. The examination looked at how quickly several functionalities executed. Researchers were able to determine which packages completed these tasks more quickly by contrasting the execution times.
2) Memory Usage: The amount of memory used up by each package while it is being executed is referred to as memory usage. It helps evaluate the packages' effectiveness at managing system resources by giving information on the memory footprint of the packages. Researchers could evaluate memory usage to find potential memory leaks or excessive memory consumption, as well as to quantify the effect of each package on system memory.

TABLE 3. Runtime (milliseconds) for stopwords.

R Package	Documents		
	10	100	1000
text2vec	5	10	90
tm	10	10	40
tau	10	10	10
lsa	790	7,560	117,170
textmineR	2,125	2,325	2,995

TABLE 4. Comparative memory usage (MB) for stopwords.

R Package	Documents		
	10	100	1000
text2vec	3.95	4.13	5.32
tm	3.92	3.92	4.11
tau	3.88	3.88	4.01
lsa	3.91	4.20	15.71
textmineR	4.09	4.43	7.71

The evaluation methods used to contrast the packages were centred on performance indicators like execution time and memory usage. To measure these metrics accurately, specific strategies including benchmarking and profiling were used. The evaluation methods are described below:

- Benchmarking: To compare the effectiveness of various packages, benchmarking was employed as a method. The identical task or set of tasks must be executed using each package, and the appropriate metrics must be measured. Their relative performance can be evaluated by contrasting the execution times and memory usage of the packages.
- Profiling: Another method used during the study was profiling. It entails examining how **R** expressions are executed or keeping track of memory utilization as operations are carried out. Data on the execution time and memory usage of each package were gathered using profiling tools like the `Rprof()` function. The effectiveness of the packages was then analysed and compared using this data.
- Iterations: For each package, several iterations were carried out to assure accurate findings. In order to account for any performance changes that could arise owing to factors like system load or caching effects, the evaluation required repeating the same task repeatedly, often 100 times. The representative value for comparison was the median execution time or memory use over these iterations.
- Results Analysis: To compare the packages, the data gathered through benchmarking and profiling was examined. Because they offer a more consistent description of the performance, the median execution time and memory use values were frequently employed for comparison. By looking at these measures, it is feasible to spot performance variations between the package and determine which one is best for a given task.

Evaluation analysis was made on those packages grouped in the techniques that are used for the same aim. The methodology used consisted in the 6 following activities.

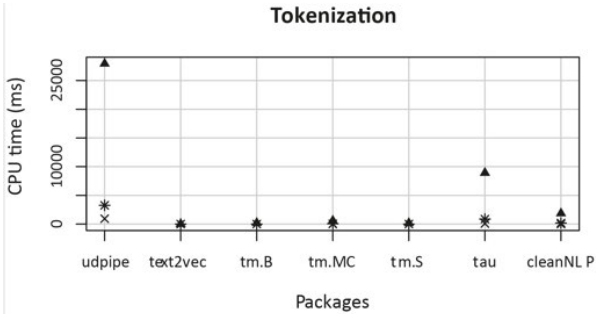

FIGURE 1. Runtime in milliseconds for tokenization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

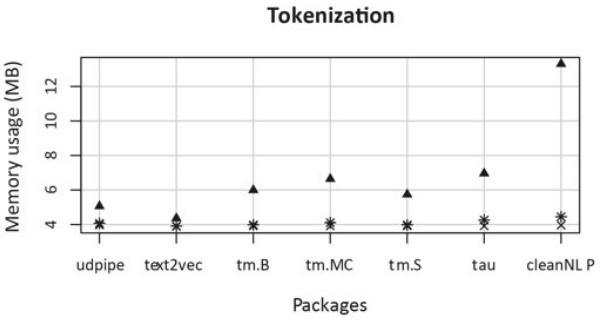

FIGURE 2. Memory usage in megabytes for tokenization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

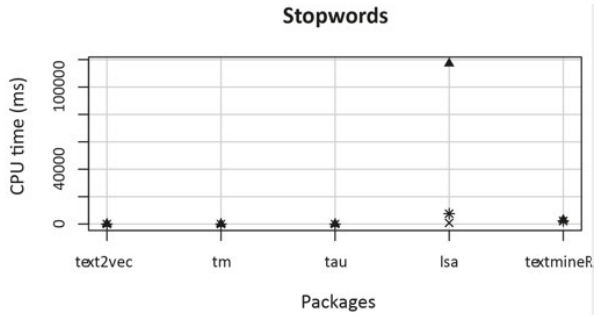

FIGURE 3. Runtime in milliseconds for stopwords on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

1) In the first step a series of random samples of documents were extracted from a public dataset, ten, one hundred and one thousand documents, with the idea of testing these samples. It was selected the "movie_review" dataset included in the R package **text2vec**. This sample is composed of three variables, "id", which is the id of the document, "sentiment", which is the sentiment of the document (one means positive and zero means negative) and finally "review" which is the raw plain text containing the movie review. The code is:

```
R> library(text2vec)
R> library(data.table)
R> data("movie_review")
```

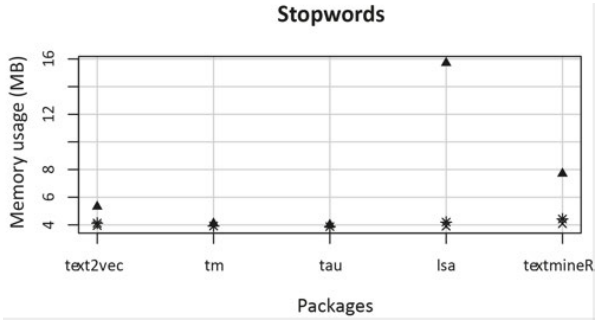

FIGURE 4. Memory usage in megabytes for stopwords on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

```
R> txt <- setDT(movie_review)
R> setkey(txt, id)
R> set.seed(1234)
R> txt10 <- txt[J(sample(movie_review$id, 10))]
R> txt100 <- txt[J(sample(movie_review$id, 100))]
R> txt1000 <- txt[J(sample(movie_review$id, 1000))]
```

2) Once the data were uploaded, iterations were performed. A total of 100 iterations were chosen for each package using the constant *ITERATIONS*. The code is:

```
R> for (i in 1:ITERATIONS) {
```

3) For each iteration, a call is made to the `gc()` function so that the garbage collector is in charge of recovering unused memory, which takes an appreciable amount of time and thus avoid misleading results during profiling. The code is:

```
R> gc()
```

4) Tests were performed with the `Rprof()` function, which enables profiling of the execution of R expressions if *MEMORY_PROFILING* constant is *FALSE* or the memory usage if it is *TRUE*. The results are written to a file with the file name following the format of the package name and the iteration number. The code is:

```
R> filename <- paste(PACKAGE, i, ".out", sep="")
R> Rprof(filename, interval = 0.01,
memory.profiling = MEMORY_PROFILING)
```

5) For each package of a technique, the necessary functions of the package are used to run the tests. The code for this step is explained below for each technique and package.

6) Median for runtime or reserved memory is extracted from the result files using the `summaryRprof()` function. The code is:

```
R> summaryRprof(paste(PACKAGE, ITERATIONS, ".out", sep=""))
R> x <- c()
R> for (i in 1:ITERATIONS) {
R>  if (MEMORY_PROFILING) {
R>      summary <- readLines(paste(PACKAGE, i, ".out",
sep=""))
R>      lastLine <- summary[length(summary)]
R>      split <- strsplit(lastLine, ":")[[1]]
R>      if (length(split) >= 3) {
R>          smallVallocSize <-  as.numeric(split[2])
R>          largeVallocSize <-  as.numeric(split[3])
R>          total <{-} smallVallocSize + largeVallocSize
R>          x <- c(x, total)
R>      }
R>  } else {
R>      summary <- summaryRprof(paste(PACKAGE, i, ".out",
```

```
sep=""))
R>      total <- summary$by.total$total.time[1]
R>      if (is.na(total)) {
R>          x <- c(x, 0)
R>      } else {
R>          x <- c(x, total)
R>      }
R>  }
R> }
R> median(x)
```

TABLE 5. Runtime (milliseconds) for lemmatization.

R Package	Documents		
	10	100	1000
udpipe	1,450	7,725	62,860
textstem	10	40	910

Step 5

Due to the interoperability offered by the packages, a complete analysis on functionality combining packages is something that, without having a clear analysis objective, would mean a much more complete development, without being able to go deeper into the benefits offered by each package. For this reason, the decision taken at the time of this comparison is to focus on the functionality offered by each one independently, so that, when carrying out a future analysis, it will serve as a guide as to which package offers better performance according to a specific objective. Therefore, step 5 is described below with the code used being separated from the rest.

1) **udpipe**. With **udpipe** it is necessary to have the model of the language to be used, as well as the information to be processed (the text to be processed and the id of each one). The function has some extra parameters activated by default, which are tagger and parser (if they are with "default" value, they look for lemmatization/POS and lexical dependencies respectively). To speed up the function, the value of these variables will be "none", forcing **udpipe** to perform only tokenization. Because **udpipe** has been forced to perform only tokenization, there will be variables with null values.

```
R> library(udpipe)
R> library(profvis)
R> longstr <- paste0("../eng.udpipe")
R> udmodel <- udpipe_load_model(longstr)
R> udp10 <- udpipe_annotate(udmodel,
x = txt10$review, doc_id = txt10$id,
tagger = "none", parser = "none")
R> udp10 <- as.data.frame(udp10)
```

2) **text2vec**. With the **text2vec** package, the samples must be passed as a parameter to the `word_tokenizer()`.

```
R> library(text2vec)
R> t2vtkn10 <- word_tokenizer(txt10[[3]])
```

The result with **text2vec** is different. Unlike **udpipe**, the result object is a list of lists (with token inside) corresponding to each of the processed documents. Package **text2vec** has a good performance (more on this later in the results' comparison) but it would be even better if it could perform the tokenization function using iterators in several threads (as it uses when creating a DTM or TCM).

3) **tm**. With the **tm** package, the process differs. In order to apply tokenization, the input must be a Corpus. A Corpus is a set of documents stored in the same standard structure. Therefore, this is the first step to be followed.

```
R> library(tm)
R> c <- data.frame(doc_id = txt10$id,
text = txt10$review)
c <- DataframeSource(c)
R> corpus10 <- VCorpus(c)
```

Once the Corpus has been obtained, tokenization can be performed. There are three different functions with which tokenization can be done, which are Boost (uses `Boost_Tokenizer()` via **Rcpp**), `MC_tokenizer()` (which implements the MC tokenizer) and `scan_tokenizer()`, which simulates the scan function, forcing characters to be taken. These functions are analyzed separately to obtain the results of each one.

```
R> boost10 <- Boost_tokenizer(corpus10)
R> mc10 <- MC_tokenizer(corpus10)
R> scan10 <- scan_tokenizer(corpus10)
```

The result variable with the tokenization follows a similar structure to the one obtained with the **text2vec** package, with the difference that it appears that the result (the list of lists) has been converted to a character vector, containing all the tokens of the processed documents (in the first position, a String with value "list(list(content))" is stored).

4) **tau**. With the **tau** package, it has a very similar function to the previous ones, and whose result is a list of characters with the tokens obtained.

```
R> library(tau)
R> tautkn10 <- tokenize(txt10$review)
R> tautkn10[1:12]
```

5) **cleanNLP**. Finally, the **cleanNLP** package. This package is an interface that uses other packages to provide functionality. Specifically, the **coreNLP** package (based on Java, and with which it has a connection through rJava), the **tokenizers** package (**R**'s own), **udpipe** (already analyzed, and therefore obviated) and finally the Python **spaCy** package can be used in an underlying way. Due to a bottleneck problem with memory and the rJava interface, the analysis will be performed with **tokenizers**, which is **R**'s own. To get the tokens that are part of the documents, first the annotation is performed and subsequently the tokens found are extracted.

```
R> library(cleanNLP)
R> cnlp_init_tokenizers()
R> cnlp10t <- cnlp_annotate(txt10$review,
text_var = txt10$id)
```

The result is an object with the same structure obtained with **udpipe** (although without some variables, such as the division by sentences or paragraphs).

The other techniques and packages follow the same methodology proposed here.

V. COMPARISON AND RESULTS

This section provides a discussion and critical comparison of packages for text mining with **R**.

TABLE 6. Memory usage (MB) for lemmatization.

R Package	Documents		
	10	100	1000
udpipe	4.31	7.34	19.64
textstem	4.73	5.43	10.02

TABLE 7. Runtime (milliseconds) for stemming.

R Package	Documents		
	10	100	1000
textstem	10	30	800
tm	10	10	150

TABLE 8. Memory usage (MB) for stemming.

R Package	Documents		
	10	100	1000
textstem	4.48	4.85	8.05
tm	3.92	4.02	5.38

TABLE 9. Runtime (milliseconds) for POS tagging.

R Package	Documents		
	10	100	1000
udpipe	1,500	7,215	65,605
openNLP	410	3,835	25,820

The main factor involved in this comparison is the performance, both in execution time and memory used, which will be compared by trying documents using the packages described in the previous sections. It should be noted that although these values are dependent on the architecture and characteristics of the system, this will allow giving an approximation about them in a particular environment, so that this information can be generalized, based on the samples.

It is important to mention that searches in literature databases regarding similar studies provide scarce results for comparison with the obtained results.

A. TOKENIZATION WITH R

As it can be seen in Table 1, the packages that offer the best performance in this sample are **text2vec** and **tm**, with the Boost and Scan functions. While **tau** and **cleanNLP** have a slightly higher time, along with, **udpipe** which has the highest execution time.

The **tm** package perceives an increase in time and memory that it suffers as the sample size increases, due to the creation of the Corpus in order to be able to perform a later tokenization. There is also a noticeable increase in memory used from **cleanNLP** in the 1,000 document sample (see Table 2).

As a conclusion, the best package in terms of performance when dealing with Tokenization is the **text2vec** package. Alternatively, there is the **tm** package (Boost and Scan), which has similar performance, especially in memory management.

It should be noted that **udpipe** has such poor performance because for Tokenization the language model to be used must be loaded in memory beforehand, otherwise the function does not provide results. If it were not for this fact, performing Tokenization with **udpipe** using parallelism would have better results than those obtained.

TABLE 10. Memory usage (MB) for POS tagging.

R Package	Documents		
	10	100	1000
udpipe	3.98	4.15	5.75
openNLP	7.81	22.163	3119.59

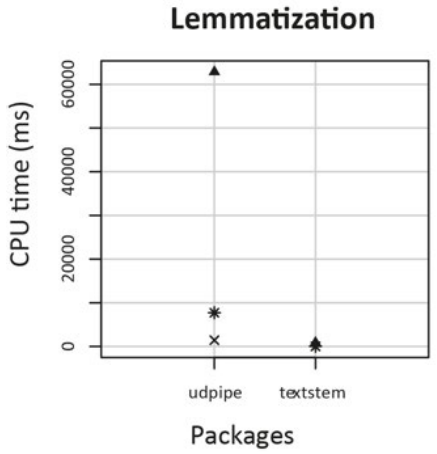

FIGURE 5. Runtime in milliseconds for lemmatization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

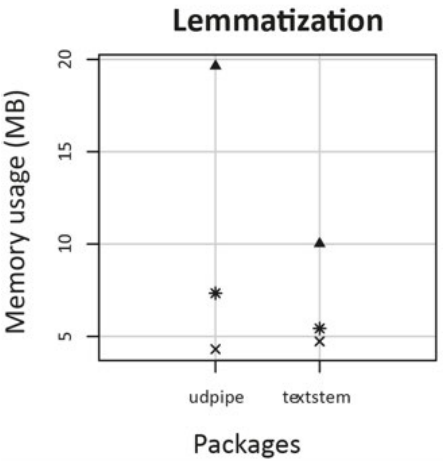

FIGURE 6. Memory usage in megabytes for lemmatization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

B. STOPWORDS WITH R

Regardless of the sample size, as it can be seen in Table 3, the package that offers the best performance is **tau**. However, it should be noted that this package does not work with DTM but with plain text, so if the input is taken into account as a factor, the package that offers the best performance (working with DTM) is **text2vec**.

When changing the sample size, the time taken by the **lsa** package increases considerably, especially in the case of the sample of 1,000 documents, where the performance offered by **lsa** is the worst among the other packages. The performance of the rest remains uniform, both in the **tau**

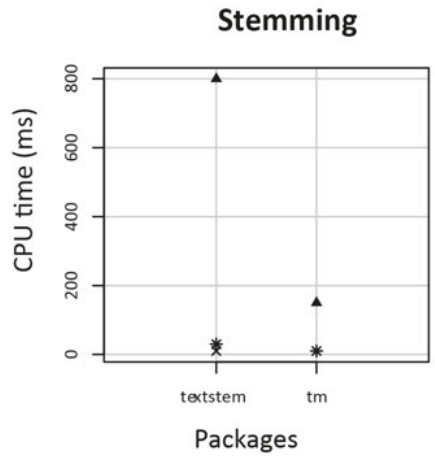

FIGURE 7. Runtime in milliseconds for stemming on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

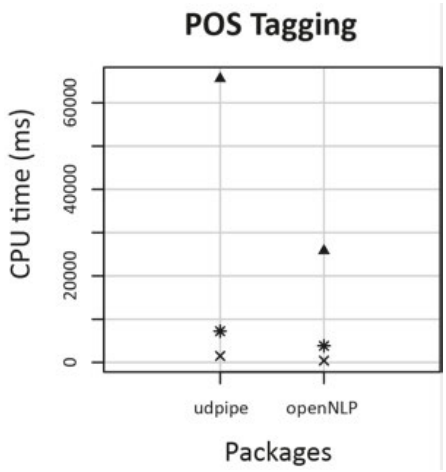

FIGURE 9. Runtime in milliseconds for POS tagging on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

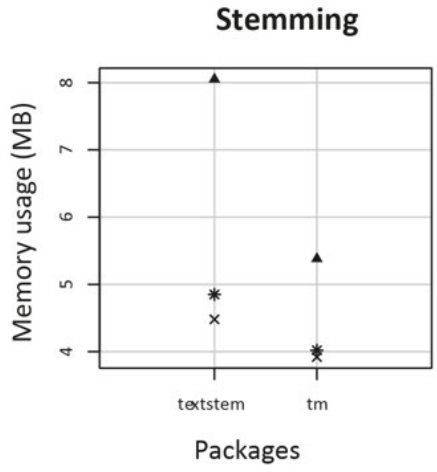

FIGURE 8. Memory usage in megabytes for stemming on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

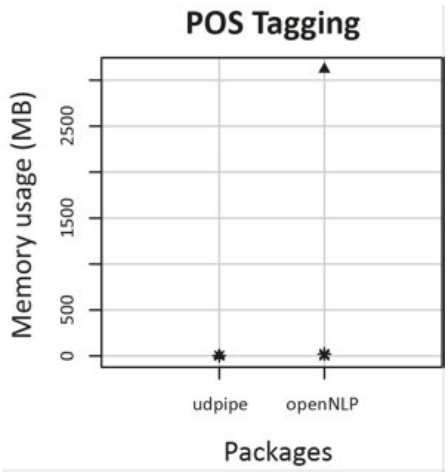

FIGURE 10. Memory usage in megabytes for POS tagging on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

package, the most efficient in plain text, and in the **text2vec** package, the most efficient among those working with DTM.

Regarding memory management shown in the Table 4, the most resource-intensive package in all samples is **lsa**, followed by **textmineR**, while **text2vec**, **tm** and **tau** make better use of memory.

In conclusion, there are five packages, two that apply to pure text, such as **tm** and **tau** (**tm** to a Corpus, which composes the set of documents of the sample space, while **tau** is to a plain text) and three others that apply stopwords elimination to DTM. In terms of memory management, **tau** is more efficient than **tm** in plain text, since generating the Corpus from samples of many documents implies a higher use of resources. Among those applied to DTM, **text2vec** is more efficient than **textmineR** in both memory and time.

C. LEMMATIZATION WITH R

Tables 5 and 6 show, by comparing all samples, that the best package is **textstem**, although **udpipe** with parallelism would have better results.

As for memory, there are few differences here. **Udpipe** uses a lot of resources in its base function, which scales according to the sample size. However, **textstem**, like **udpipe**'s base function, increases memory usage with sample size, but to a lesser extent.

In conclusion, in terms of execution time, the best package is **textstem** on samples of ten, one hundred and one thousand documents, while **udpipe** without parallelism is not as fast. Both packages use more or less the same memory, so the choice is left to the user's needs.

TABLE 11. Comparative runtime (milliseconds) for TF-IDF.

R Package	Documents		
	10	100	1000
text2vec	10	20	120
textir	10	20	90
textmineR	1,630	1,760	2,240
cleanNLP	20	170	1,810

TABLE 12. Comparative memory usage (MB) for TF-IDF.

R Package	Documents		
	10	100	1000
text2vec	4.00	4.22	6.81
textir	3.97	4.20	7.52
textmineR	4.26	4.69	10.36
cleanNLP	4.16	5.30	11.07

TABLE 13. Runtime (milliseconds) for normalization.

R Package	Documents		
	10	100	1000
fastTextR	10	20	130
text2vec	10	10	100

TABLE 14. Memory usage (MB) for normalization.

R Package	Documents		
	10	100	1000
fastTextR	3.88	3.88	3.88
text2vec	3.96	4.18	6.12

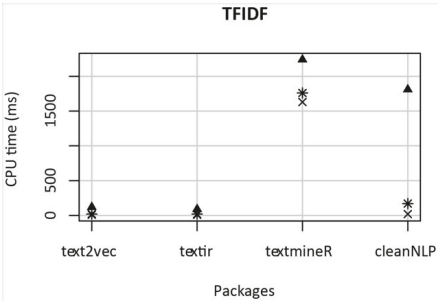

FIGURE 11. Runtime in milliseconds for TF-IDF on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

D. STEMMING WITH R

Table 7 shows that, when the sample contains a low volume of documents to which to apply this reduction technique, the time is negligible in both packages. However, when the sample increases, large differences are present. While the **tm** package maintains a good performance (even taking into account that the Corpus had to be created beforehand to perform this technique), the execution time of **textstem** increases when the sample reaches a thousand documents.

In memory usage (see Table 8), the result is similar. While the **tm** package does a better memory management even if it has to store in memory the Corpus it is going to work with, **textstem** requires more resources to produce the same result. In conclusion, for Stemming, the package that offers the best performance is the **tm** package.

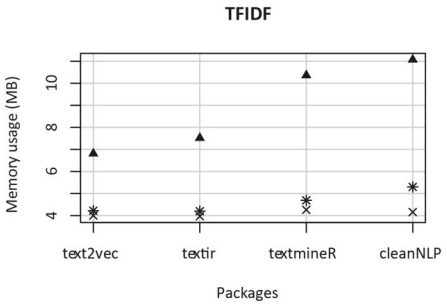

FIGURE 12. Memory usage in megabytes for TF-IDF on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

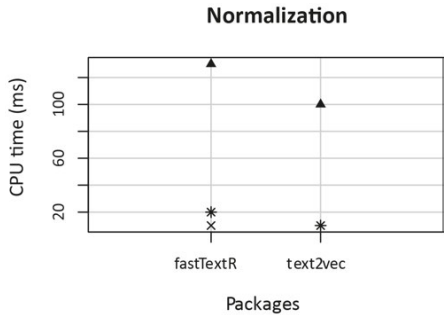

FIGURE 13. Runtime in milliseconds for normalization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

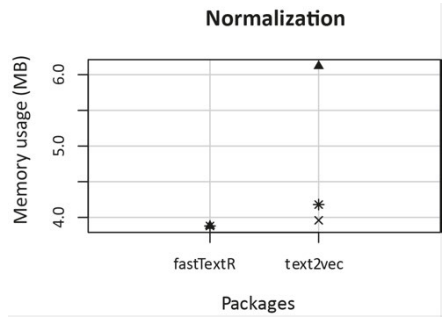

FIGURE 14. Memory usage in megabytes for normalization on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

E. POS TAGGING WITH R

As it can be seen in the runtime Table 9, **openNLP** could be a clear winner. The disadvantage is that it consumes too much memory, as shown in Table 10 (even sometimes **R** cannot allocate that much memory with a thousand documents). This is because it uses rJava as an interface to connect to Java and use it in an underlying way, which causes unnecessary memory and time expenditure.

So if better memory consumption is needed the **udpipe** package is a good choice, but **openNLP** has less execution time considering that it is compared only with **udpipe** and it also needs the Java Virtual Machine, which consumes more resources.

TABLE 15. Runtime (milliseconds) for topic modeling.

R Package	Documents		
	10	100	1000
textmineR	3,770	6,710	70,600
fastTextR	320	330	480
lsa	670	7,180	109,735

TABLE 16. Memory usage (MB) for topic modeling.

R Package	Documents		
	10	100	1000
textmineR	6.17	7.63	23.09
fastTextR	3.90	3.91	3.91
lsa	3.92	5.39	85.23

TABLE 17. Runtime (ms) for sentiment analysis.

R Package	Documents		
	10	100	1000
Rsentiment (Score)	1,130	15,025	155,065
sentimentr (Score)	70	490	4,180
Rsentiment (Classification)	1,255	17,190	154,720
sentimentr (Classification)	50	340	3,380

TABLE 18. Memory usage (MB) for sentiment analysis.

R Package	Documents		
	10	100	1000
Rsentiment (Score)	4.14	4.38	5.46
sentimentr (Score)	4.89	6.16	10.76
Rsentiment (Classification)	4.16	4.39	4.90
sentimentr (Classification)	4.85	7.84	14.51

F. TF-IDF WITH R

Conclusions on this technique based on Tables 11 and 12 are that, **textir** and **text2vec** offer a negligible execution time in the ten and one hundred document samples. This is, however, different for the larger samples, such as the one thousand document sample, where the execution time increases more with **text2vec** than with **textir**. This was expected since for the creation of this type of matrix, **text2vec** has to first create a vocabulary of the documents, the DTM on this vocabulary and finally, create the TF-IDF. Regarding memory management, both packages follow the same line as in the runtime table. This improvement, compared to **textmineR**, is due to the use of iterators that work in parallel traversing the documents to form the frequency matrix.

As for the **textmineR** package, its performance is quite poor compared to the other two packages, requiring more memory needed for the execution in the sample of one thousand documents, and also longer execution time in this same sample compared to the other packages. The last package, **openNLP**, does not offer a better result. While the ten-document sample shows a decent result, with larger sample sizes it proves to be inefficient compared to the other packages. Therefore, the best result is offered by the **textir** package.

G. NORMALIZATION WITH R

Tables 13 and 14 show that **text2vec** uses more resources, both time and memory, to perform this function. Except in the sample of one thousand documents, where the execution time of **fastTextR** is higher. However, its usefulness is much greater, since applying normalization to a DTM is more convenient than applying it to a text and then transforming it, because it increases the execution time and the memory reserved for this task. In this case, **fastTextR** uses practically no memory, since the result is saved as plain text.

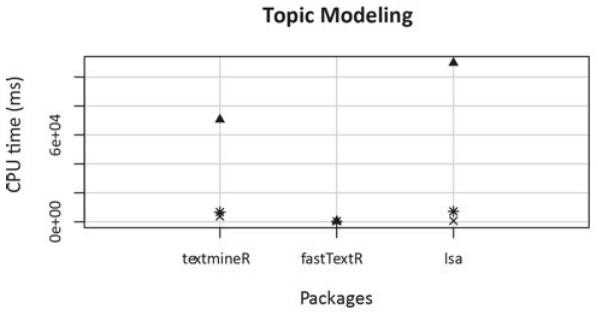

FIGURE 15. Runtime in milliseconds for topic modeling on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

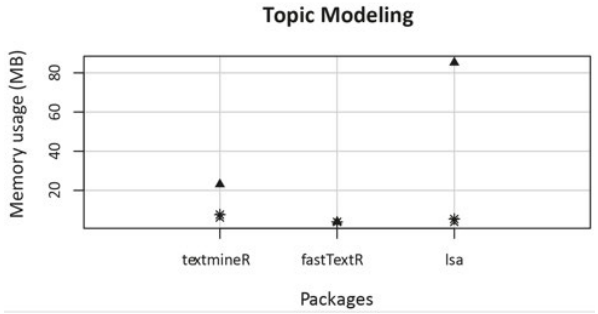

FIGURE 16. Memory usage in megabytes for topic modeling on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

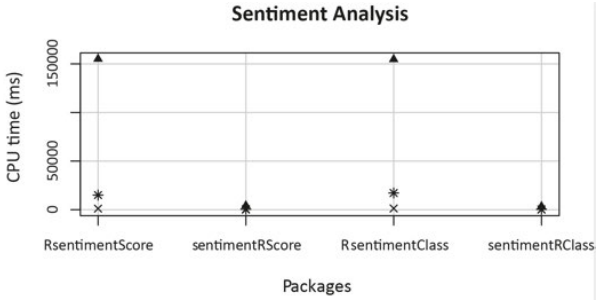

FIGURE 17. Runtime in milliseconds for sentiment analysis on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

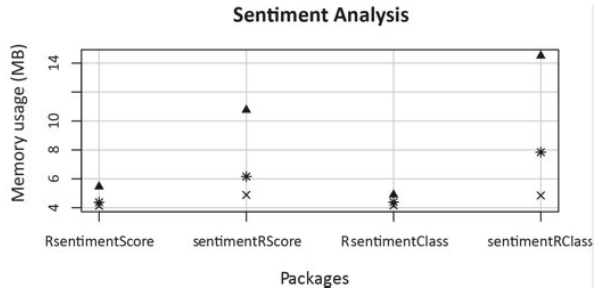

FIGURE 18. Memory usage in megabytes for sentiment analysis on the movie_review dataset utilizing 10 documents (×), 100 documents (∗) and 1000 documents (▲).

TABLE 19. Summary of benchmarking results considering runtime (miliseconds). (C stands for classification and S stands for score.)

Techniques	R Package	Documents		
		10	100	1000
Tokenization	text2vec	5	10	20
	tm (Scan)	10	10	180
	tm (Boost)	10	10	210
	tm (MC)	10	60	600
	cleanNLP	10	190	1,875
	tau	100	880	8,930
	udpipe	950	3,270	27,940
Stopwords	text2vec	5	10	90
	tau	10	10	10
	tm	10	10	40
	lsa	790	7,560	117,170
	textmineR	2,125	2,325	2,995
Lemmatization	textstem	10	40	910
	udpipe	1,450	7,725	62,860
Stemming	tm	10	10	150
	textstem	10	30	800
POS Tagging	openNLP	410	3,835	25,820
	udpipe	1,500	7,215	65,605
TF-IDF	textir	10	20	90
	text2vec	10	20	120
	cleanNLP	20	170	1,810
	textmineR	1,630	1,760	2,240
Normalization	text2vec	10	10	100
	fastTextR	10	20	130
Topic Modeling	fastTextR	320	330	480
	lsa	670	7,180	109,735
	textmineR	3,770	6,710	70,600
Sentiment Analysis	sentimentr (C)	50	340	3,380
	sentimentr (S)	70	490	4,180
	Rsentiment (S)	1,130	15,025	155,065
	Rsentiment (C)	1,255	17,190	154,720

H. TOPIC MODELING WITH R

With Tables 15 and 16 the following observations can be made: the Supervised Classification of **fastTextR** gives the base algorithm a remarkable performance. Regarding memory usage, **fastTextR** does not reserve too much memory for this process, which is an advantage over its competitors, since **textmineR** and **lsa** need much more megabytes (especially in samples of one hundred and one thousand documents) when reserving memory for the model. For instance, according to Table 16, **fastTextR** uses only 3.90 MB of memory when analyzing 10 documents, while **textmineR** and **lsa** use 6.17 MB and 3.92 MB, respectively. This pattern persists for bigger document sizes, as **fastTextR** consistently uses only 3.91 MB of memory, while **textmineR** and **lsa** require significantly more memory. As a conclusion, the package that offers the best performance in both aspects of execution time and memory used is **fastTextR** with its Supervised Classification algorithm.

I. SENTIMENT ANALYSIS WITH R

The results for sentiment analysis with **R** are shown in Tables 17 and 18. The packages offer similar functionality, separated by Score and Classification. While **sentimentr** is built using R's own resources, **RSentiment** uses an interface to Java in order to develop its functionality. This is a runtime disadvantage as shown in the tables, due to the resources that need to be provided to Java. The clear winner is **sentimentr** in terms of computational efficiency, but in terms of memory usage it consumes somewhat more especially with 100 and 1,000 documents, which unlike **Rsentiment** makes better use of memory. For example, Table 17 shows that **sentimentr** uses 6.16 MB and 10.76 MB of RAM with 100 and 1000 documents, respectively, while **Sentimentr** uses 4.38 MB and

TABLE 20. Summary of benchmarking results considering memory usage (MB). (C stands for classification and S stands for score.)

Techniques	R Package	Documents		
		10	100	1000
Tokenization	text2vec	3.89	3.91	4.36
	tm (Scan)	3.92	3.98	5.74
	tm (Boost)	3.92	3.98	5.99
	tm (MC)	3.92	4.11	6.64
	tau	3.92	4.26	6.96
	cleanNLP	3.96	4.45	13.30
	udpipe	3.97	4.07	5.06
Stopwords	tau	3.88	3.88	4.01
	lsa	3.91	4.20	15.71
	tm	3.92	3.92	4.11
	text2vec	3.95	4.13	5.32
	textmineR	4.09	4.43	7.71
Lemmatization	udpipe	4.31	7.34	19.64
	textstem	4.73	5.43	10.02
Stemming	tm	3.92	4.02	5.38
	textstem	4.48	4.85	8.05
POS Tagging	udpipe	3.98	4.15	5.75
	openNLP	7.81	22.163	3119.59
TF-IDF	textir	3.97	4.20	7.52
	text2vec	4.00	4.22	6.81
	cleanNLP	4.16	5.30	11.07
	textmineR	4.26	4.69	10.36
Normalization	fastTextR	3.88	3.88	3.88
	text2vec	3.96	4.18	6.12
Topic Modeling	fastTextR	3.90	3.91	3.91
	lsa	3.92	5.39	85.23
	textmineR	6.17	7.63	23.09
Sentiment Analysis	Rsentiment (S)	4.14	4.38	5.46
	Rsentiment (C)	4.16	4.39	4.90
	sentimentr (C)	4.85	7.84	14.51
	sentimentr (S)	4.89	6.16	10.76

TABLE 21. Summary of the best and worst packages for each technique depending on memory usage and time consumed.

Techniques	Best Package		Worst Package	
	time	memory	time	memory
Tokenization	text2vec	text2vec	udpipe	udpipe
Stopwords	text2vec	tau	textmineR	textmineR
Lemmatization	textstem	udpipe	udpipe	textstem
Stemming	tm	tm	textstem	textstem
POS Tagging	openNLP	udpipe	udpipe	openNLP
TF-IDF	textir	textir	textmineR	textmineR
Normalization	text2vec	fastTextR	fastTextR	text2vec
Topic Modeling	fastTextR	fastTextR	textmineR	textmineR
Sentiment Analysis	sent (C)	Rsent (S)	Rsent (C)	sent (S)

TABLE 22. Packages and versions information.

Package	Version
udpipe	0.8.8
textstem	0.1.4
openNLP	0.2-7
text2vec	0.6
lsa	0.73.2
tm	0.7-8
textmineR	3.0.5
textir	2.0-5
tau	0.0-24
fastTextR	0.0.2
sentimentr	2.9.0
cleanNLP	3.0.3
RSentiment	2.2.2

5.46 MB with the same document sizes. As a result, while assessing sentiment in bigger document sets, **Rsentiment** shows superior memory efficiency, making the most use of available resources.

Tables 19 and 20 give a summary of the packages for each technique while taking their runtime and memory usage into account. Researchers can use these tables to gain important insights for their analysis.

VI. CONCLUSION

This section will show the conclusions drawn based on the results, as well as the issues related to it.

Starting with Tokenization, the packages that stand out the most are **text2vec** and **tm**. The only **R** package that performs worse is **udpipe**. It is worth noting that **udpipe** could achieve better results working in parallel. To take advantage of the parallelization capabilities of contemporary hardware and maybe enhance the tokenization process, parallel computing utilizing packages like **parallel** [42] or **future** [4] could be investigated. Also, the **text2vec** package is one of the most efficient in the Stopwords technique applied to DTM, while **tau** is the most efficient for plain text.

As for the Lemmatization and Stemming techniques, the best package is **textstem**, but in Stemming the **tm** package is even better considering that it had to create and store in memory the Corpus. This package was already outstanding in Tokenization.

Regarding the POS Tagging technique, it is more difficult to make a choice, since **udpipe** makes a good use of memory but **openNLP** has better execution times. In contrast, this is not the case for the TF-IDF technique, where the **textir** package is a clear choice.

Applying Normalization with the **text2vec** package is more convenient as it does not use plain text. But **fastTextR** does not use so much memory and is also the best package for Topic Modeling thanks to the Supervised Classification algorithm.

Finally, the Sentiment Analysis technique with the **sentimentr** package will give better performance in terms of execution time, but **RSentiment** makes better use of memory if the documents to be analyzed are of a large volume.

Even though some packages proved to be better at certain strategies, there is still future work, especially in fields like parallel processing, hybrid approaches, and novel algorithms. It is possible to improve the effectiveness, scalability, and precision of text analysis tasks in the **R** environment by addressing these issues.

In conclusion, researchers should take into account the particular needs of their text mining activities and select the best R package based on performance metrics like execution time and memory utilization. These are the recommended packages for various tasks, as shown in Table 21:

- Tokenization: the **text2vec** package is advised for the best time performance.
- Stopwords: **text2vec** is the suggested package for managing stopwords within a DTM. On the other hand, **tau** is the suggested package if it needs to handle stopwords in plain text.
- Lemmatization: The recommended package is **textstem** in terms of time for effective lemmatization. However, **udpipe** with parallelism is the recommended option if memory usage is a concern.
- Stemming: In terms of time and memory requirements, the **tm** package is suggested for stemming tasks.
- POS Tagging: **openNLP** is the recommended package if quicker execution is a top priority. On the other hand, **udpipe** is the best option if memory use is an issue.
- TF-IDF: **textir** is the recommended package for TF-IDF calculations in terms of both time and memory.
- Normalization: The suggested package is **text2vec** for improved runtime efficiency. The **fastTextR** package is the best option if memory utilization is a problem.
- Topic Modeling: When it comes to time and memory requirements, the **fastTextR** (Supervised Classification) package is suggested for Topic Modeling
- Sentiment Analysis: **sentimentr** is the recommended package if computational effectiveness is a top concern. However, **RSentiment** is the best option for memory effectiveness.

These recommendations give researchers information on the advantages and disadvantages of several R packages for various text mining tasks, empowering them to make wise choices based on their unique requirements and available resources. The version of the analyzed packages can be found in Table 22 for reference by the researchers.

COMPUTATIONAL DETAILS

The results in this paper were obtained using **R** 4.1.1 [43] and used packages **udpipe** version 0.8.8 [61], **textstem** version 0.1.4 [47], **openNLP** version 0.2-7 [24], **text2vec** version 0.6 [49], **lsa** version 0.73.2 [62], **tm** version 0.7-8 [16], **textmineR** version 3.0.5 [26], **textir** version 2.0-5 [55], **tau** version 0.0-24 [8], **fastTextR** version 0.0.2 [48], **sentimentr** version 2.9.0 [46], **cleanNLP** version 3.0.3 [1] and **RSentiment** version 2.2.2 [6]. The benchmarks were performed on a Windows 10 × 86-64 Machine with Intel(R) Core(TM) i7-7700 CPU 3.60 GHz. **R** itself and all packages used are available from the Comprehensive **R** Archive Network (CRAN) at https://CRAN.R-project.org/.

REFERENCES

[1] T. Arnold, "A tidy data model for natural language processing using cleanNLP," *R J.*, vol. 9, no. 2, pp. 1–20, 2017. [Online]. Available: https://journal.r-project.org/archive/2017/RJ-2017-035/index.html
[2] M. Z. Asghar, A. Khan, S. Ahmad, and F. M. Kundi, "A review of feature extraction in sentiment analysis," *J. Basic Appl. Sci. Res.*, vol. 4, no. 3, pp. 181–186, 2014.
[3] A. Balahur and A. Montoyo, "A feature dependent method for opinion mining and classification," in *Proc. Int. Conf. Natural Lang. Process. Knowl. Eng.*, Oct. 2008, pp. 1–7, doi: 10.1109/NLPKE.2008.4906796.
[4] H. Bengtsson, "The R journal: A unifying framework for parallel and distributed processing in R using futures," *R J.*, vol. 13, pp. 273–291, 2021, doi: 10.32614/RJ-2021-048.
[5] M. W. Berry and M. Castellanos, *Survey of Text Mining: Clustering, Classification, and Retrieval*. New York, NY, USA: Springer, 2007, pp. 3–23.

[6] S. Bose and S. Goswami. (2018) *RSentiment: Analyse Sentiment of English Sentences, R Package Version 2.2.2*. [Online]. Available: https://CRAN.R-project.org/package=RSentiment

[7] E. Brill and M. Pop, "Unsupervised learning of disambiguation rules for part-of-speech tagging," in *Natural Language Processing Using Very Large Corpora*. Dordrecht, The Netherlands: Springer, 1999, pp. 27–42, doi: 10.1007/978-94-017-2390-9_3.

[8] C. Buchta, K. Hornik, I. Feinerer, and D. Meyer. (2021). *Text Analysis Utilities. R Package Version 0.0-24*. [Online]. Available: https://CRAN.R-project.org/package=tau

[9] R. K. Charania, "Exploring and benchmarking high performance & scientific computing using R R HPC packages and lower level compiled languages a comparative study," 2019, *arXiv:1904.03343*.

[10] T. Cover and J. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2012. [Online]. Available: https://books.google.es/books?id=VWq5GG6ycxMC

[11] J. J. Cuadrado-Gallego and Y. Demchenko, "Data science body of knowledge," in *The Data Science Framework: A View From the EDISON Project*, J. J. C. Gallego and Y. Demchenko, Eds. Cham, Switzerland: Springer, 2020, pp. 43–73.

[12] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, "Using latent semantic analysis to improve access to textual information," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*. New York, NY, USA: Association for Computing Machinery, 1988, pp. 281–285, doi: 10.1145/57167.57214.

[13] S. T. Dumais, "Latent semantic analysis," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, no. 1, pp. 188–230, 2004. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/aris.1440380105

[14] A. Esuli and F. Sebastiani, "SentiWordNet: A publicly available lexical resource for opinion mining," in *Proc. Int. Conf. Lang. Resour. Eval. (LREC)*. Genoa, Italy: European Language Resources Association, May 2006.

[15] I. Feinerer, "An introduction to text mining in R," *R News*, vol. 8, no. 2, pp. 19–22, 2008. [Online]. Available: https://cran.r-project.org/doc/Rnews/

[16] I. Feinerer and K. Hornik, (2020). *Text Mining Package. R Package Version 0.7-8*. [Online]. Available: https://CRAN.R-project.org/package=tm

[17] D. Meyer, K. Hornik, and I. Feinerer, "Text mining infrastructure in R," *J. Stat. Softw.*, vol. 25, no. 5, pp. 1–54, Mar. 2008. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v025i05

[18] C. Fox, "Lexical analysis and stoplists," in *Information Retrieval: Data Structures and Algorithms*. Upper Saddle River, NJ, USA: Prentice-Hall, 1992, pp. 102–130. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/FrakesB92.html#Fox92 and http://dl.acm.org/citation.cfm?id=129687.129694

[19] M. Gagolewski, "Stringi: Fast and portable character string processing in R," *J. Stat. Softw.*, vol. 103, no. 2, pp. 1–59, 2022. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v103i02, doi: 10.18637/jss.v103.i02.

[20] B. Grün and K. Hornik, "Topicmodels: An R package for fitting topic models," *J. Stat. Softw.*, vol. 40, no. 13, pp. 1–30, 2011. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v040i13, doi: 10.18637/jss.v040.i13.

[21] V. Gurusamy and S. Kannan, "Preprocessing techniques for text mining," in *Proc. RTRICS*, India, Oct. 2014.

[22] D. Harman, "How effective is suffixing?" *J. Amer. Soc. Inf. Sci.*, vol. 42, no. 1, pp. 7–15, 1991, doi: 10.1002/(SICI)1097-4571(199101)42:1<7::AID-ASI2>3.0.CO;2-P.

[23] W. P. Headden, D. McClosky, and E. Charniak, "Evaluating unsupervised part-of-speech tagging for grammar induction," in *Proc. 22nd Int. Conf. Comput. Linguistics (COLING)*. Manchester, U.K.: Coling 2008 Organizing Committee, Aug. 2008, pp. 329–336. [Online]. Available: https://aclanthology.org/C08-1042

[24] K. Hornik. (2019). *OpenNLP: Apache OpenNLP Tools Interface. R Package Version 0.2-7*. [Online]. Available: https://CRAN.R-project.org/package=openNLP

[25] A. Jivani, "A comparative study of stemming algorithms," *Int. J. Comput. Technol. Appl.*, vol. 2, pp. 1930–1938, Nov. 2011.

[26] T. Jones. (2021). *R: Functions for Text Mining and Topic Modeling. R Package Version 3.0.5*. [Online]. Available: https://CRAN.R-project.org/package=textmineR

[27] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, "kernlab—An S4 package for kernel methods in R," *J. Stat. Softw.*, vol. 11, no. 9, pp. 1–20, 2004. [Online]. Available: https://www.jstatsoft.org/index.php/jss/article/view/v011i09, doi: 10.18637/jss.v011.i09.

[28] H. Kotthaus, I. Korb, M. Lang, B. Bischl, J. Rahnenführer, and P. Marwedel, "Runtime and memory consumption analyses for machine learning R programs," *J. Stat. Comput. Simul.*, vol. 85, no. 1, pp. 14–29, Jan. 2015.

[29] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse Process.*, vol. 25, nos. 2–3, pp. 259–284, Jan. 1998, doi: 10.1080/01638539809545028.

[30] T. K. Landauer and S. T. Dumais, "A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge," *Psychol. Rev.*, vol. 104, no. 2, pp. 211–240, Apr. 1997.

[31] B. Liu, *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge, U.K.: Cambridge Univ. Press, 2020.

[32] H. Liu, T. Christiansen, W. A. Baumgartner, and K. Verspoor, "BioLemmatizer: A lemmatization tool for morphological processing of biomedical text," *J. Biomed. Semantics*, vol. 3, no. 1, pp. 1–29, Dec. 2012.

[33] H. Liu and P. Singh, "ConceptNet—A practical commonsense reasoning tool-kit," *BT Technol. J.*, vol. 22, pp. 211–226, Jun. 2004, doi: 10.1023/B:BTTJ.0000047600.45421.6d.

[34] R. T.-W. Lo, B. He, and I. Ounis, "Automatically building a stopword list for an information retrieval system," *J. Digit. Inf. Manag.*, vol. 3, no. 1, pp. 3–8, 2005.

[35] S. Menaka and N. Radha, "Text classification using keyword extraction technique," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 12, 2013.

[36] M. Vijayarani, "Preprocessing techniques for text mining—An overview," *Int. J. Comput. Sci. Commun. Netw.*, vol. 5, no. 1, pp. 7–16, 2015.

[37] J. Ooms. (2020) *Hunspell: High-Performance Stemmer, Tokenizer, and Spell Checker, R Package Version 3.0.1*. [Online]. Available: https://CRAN.R-project.org/package=hunspell

[38] P. Patheja, A. Waoo, and R. Garg, "Analysis of part of speech tagging," *Int. J. Comput. Appl.*, vol. 1, pp. 1–5, Jan. 2012.

[39] F. Peng, N. Ahmed, X. Li, and Y. Lu, "Context sensitive stemming for web search," in *Proc. 30th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.* Amsterdam, The Netherlands: Association for Computing Machinery, Jul. 2007, pp. 639–646, doi: 10.1145/1277741.1277851.

[40] K. Plevoets, "svs: Tools for semantic vector spaces," R Found. Stat. Comput., Vienna, Austria, Tech. Rep., 2015.

[41] J. Plisson, N. Lavrac, and D. Mladenic, "A rule based approach to word lemmatization," in *Proc. IS*, vol. 3. Ljubljana, Slovenia, 2004.

[42] *R: A Language and Environment for Statistical Computing*, R Core Team, R Found. Stat. Comput., Vienna, Austria, 2021. [Online]. Available: https://www.R-project.org/

[43] *R: A Language and Environment for Statistical Computing*, R Core Team, R Found. Stat. Comput., Vienna, Austria, 2022. [Online]. Available: https://www.R-project.org/

[44] C. Ramasubramanian and R. Ramya, "Effective pre-processing activities in text mining using improved porter's stemming algorithm," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 2, no. 12, 2013.

[45] T. W. Rinker. (2018). Lexicon: Lexicon data, version 1.2.1. Buffalo, NY, USA. [Online]. Available: http://github.com/trinker/lexicon

[46] T. W. Rinker. (2021). Sentimentr: Calculate text polarity sentiment, version 2.9.0. Buffalo, NY, USA. [Online]. Available: https://github.com/trinker/sentimentr

[47] T. W. Rinker. (2018). Textstem: Tools for stemming and lemmatizing text, version 0.1.4. Buffalo, NY, USA. [Online]. Available: http://github.com/trinker/textstem

[48] F. Schwendinger and E. Hvitfeldt. (2020). *FastTextR: An Interface to 'FastText' Library, R Package Version 2.0*. [Online]. Available: https://CRAN.R-project.org/package=fastTextR

[49] D. Selivanov, M. Bickel, and Q. Wang. (2020). *Text2Vec: Modern Text Mining Framework for R, R Package Version 0.6*. [Online]. Available: https://CRAN.R-project.org/package=text2vec

[50] D. Sharma, "Stemming algorithms: A comparative study and their analysis," *Int. J. Appl. Inf. Syst.*, vol. 4, no. 3, pp. 7–12, 2012.

[51] A. Sidorova, N. Evangelopoulos, J. S. Valacich, and T. Ramakrishnan, "Uncovering the intellectual core of the information systems discipline," *MIS Quart.*, vol. 32, no. 3, pp. 467–482, 2008. [Online]. Available: http://www.jstor.org/stable/25148852

[52] J. Silge and D. Robinson, *Text Mining With R: A Tidy Approach*. Sebastopol, CA, USA: O'Reilly, 2017. [Online]. Available: https://books.google.es/books?id=7bQzMQAACAAJ

[53] J. Silge and D. Robinson, "Tidytext: Text mining and analysis using tidy data principles in R," *J. Open Source Softw.*, vol. 1, no. 3, p. 37, Jul. 2016, doi: 10.21105/joss.00037.

[54] M. Straka and J. Straková, "Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe," in *Proc. CoNLL Shared Task: Multilingual Parsing from Raw Text Universal Dependencies*. Vancouver, BC, Canada: Association for Computational Linguistics, Aug. 2017, pp. 88–99. [Online]. Available: https://aclanthology.org/K17-3009
[55] M. Taddy, "Multinomial inverse regression for text analysis," *J. Amer. Stat. Assoc.*, vol. 108, no. 503, pp. 755–770, Sep. 2013.
[56] S. Theußl, I. Feinerer, and K. Hornik, "A tm plug-in for distributed text mining in R," *J. Stat. Softw.*, vol. 51, no. 5, pp. 1–31, 2012, doi: 10.18637/jss.v051.i05.
[57] Z. Tong and H. Zhang, "A text mining research based on LDA topic modelling," in *Proc. Int. Conf. Comput. Sci., Eng. Inf. Technol.*, vol. 6, May 2016, pp. 201–210, doi: 10.5121/csit.2016.60616.
[58] L. Vadicamo, F. Carrara, A. Cimino, S. Cresci, F. Dell'Orletta, F. Falchi, and M. Tesconi, "Cross-media learning for image sentiment analysis in the wild," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2017, pp. 308–317, doi: 10.1109/ICCVW.2017.45.
[59] C. Van Rijsbergen, *Information Retrieval*. London, U.K.: Butterworth, 1979. [Online]. Available: https://books.google.es/books?id=t-pTAAAAMAAJ
[60] J. Webster and C. Kit, "Tokenization as the initial phase in NLP," in *Proc. 14th Int. Conf. Comput. Linguistics*, Jan. 1992, pp. 1106–1110, doi: 10.3115/992424.992434.
[61] J. Wijffels. (2021). *Udpipe: Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing With the 'UDPipe' 'NLP' Toolkit. R Package Version 0.8.8*. [Online]. Available: https://CRAN.R-project.org/package=udpipe
[62] F. Wild. (2020). *LSA: Latent Semantic Analysis. R Package Version 0.73.2*. [Online]. Available: https://CRAN.R-project.org/package=lsa
[63] G. Wilson and M. Heywood, "Use of a genetic algorithm in brill's transformation-based part-of-speech tagger," in *Proc. 7th Annu. Conf. Genet. Evol. Comput.*, Jun. 2005, pp. 2067–2073, doi: 10.1145/1068009.1068352.
[64] C. Xu, S. Cetintas, K.-C. Lee, and L.-J. Li, "Visual sentiment prediction with deep convolutional neural networks," 2014. [Online]. Available: http://arxiv.org/abs/1411.5731
[65] J. Xu and W. B. Croft, "Corpus-based stemming using cooccurrence of word variants," *ACM Trans. Inf. Syst.*, vol. 16, no. 1, pp. 61–81, Jan. 1998, doi: 10.1145/267954.267957.
[66] Y. Xu, G. Jones, J. Li, B. Wang, and C. Sun, "Study on mutual information-based feature selection for text categorization," *J. Comput. Inf. Syst.*, vol. 3, no. 3, pp. 1007–1012, 2007.

CARLOS J. HELLÍN received the B.S. degree in computer science engineering and the M.S. degree in agile software development for the web from the University of Alcalá, Spain, in 2022 and 2023, respectively. He is currently pursuing the Ph.D. degree in information and knowledge engineering. He is also a Researcher with the Department of Computer Science, University of Alcalá. His research interests include the optimization of antennas, the analysis of radio propagation in rural and urban environments, gamification, data science, and artificial general intelligence.

ADRIÁN VALLEDOR received the B.S. degree in information systems engineering and the M.S. degree in agile software development for the web from the University of Alcalá, Spain, in 2021 and 2023, respectively. He is currently pursuing the Ph.D. degree in information and knowledge engineering. He is also a Researcher with the Department of Computer Science, University of Alcalá. His research interests include the optimization of antennas, the analysis of radio propagation in rural and urban environments, artificial intelligence, neuronal networks, and voice processing.

JUAN J. CUADRADO-GALLEGO is currently an Associate Professor in computer science and artificial intelligence with the Department of Computer Science, University of Alcalá, and an Affiliate Associate Professor with the Department of Computer Science and Software Engineering, Faculty of Engineering and Computer Science, Concordia University, Montreal, Canada. He is also researching in the fields of artificial intelligence and data science. He has made more than 200 scientific publications, many of which have been in journals indexed in the JRC Science Edition. He has also participated, as a principal investigator or a researcher, in numerous research projects. He has also been an External Evaluator of projects in computer science, of the Natural Sciences and Engineering Research Council of Canada, since 2014, and an Evaluator of the National Agency for Evaluation and Prospective, the General Directorate of Scientific and Technical Research of Spain, Ministry of Economy, Industry and Competitiveness of Spain.

ABDELHAMID TAYEBI was born in 1983. He received the B.S. and M.S. degrees in telecommunications engineering from the University Polytechnic of Cartagena, Spain, in 2005 and 2007, respectively, and the Ph.D. degree in telecommunications engineering from the University of Alcalá, Spain, in 2011. In 2011, he was an Assistant Professor with the University of Alcalá, where he has been a Professor, since 2019. He was a Faculty Researcher with The University of Hong Kong, in 2011, and the Instituto de Telecomunicaçoes, Lisbon, in 2014. He has participated in 40 research projects with Spanish and European companies. He has published 31 articles in peer-reviewed journals, a book, a book chapter, and around 45 conference contributions at national and international symposia. His research interests include the design and optimization of antennas, electromagnetic radiation and scattering, and the development of web tools for the analysis of radio propagation in rural and urban environments.

JOSEFA GÓMEZ was born in 1984. She received the B.S. and M.S. degrees in telecommunications engineering from the Polytechnic University of Cartagena, Spain, in 2005 and 2007, respectively, and the Ph.D. degree in telecommunications engineering from the University of Alcalá, Spain, in 2011. She was a Faculty Researcher with The University of Hong Kong, in 2011, and the Instituto de Telecomunicaçoes, Lisbon, in 2014. Since 2012, she has been an Assistant Professor with the University of Alcalá. She has participated in 42 research projects with Spanish and European companies. She has published 33 articles in peer-reviewed journals, a book, two book chapters, and more than 45 conference contributions at national and international symposia. Her research interests include the optimization and analysis of antennas, the design of graphical user interfaces, and the study of propagation for mobile communications or wireless networks in both outdoor and indoor environments.