UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE
POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Extragere de competențe științifice

Ana-Maria Năstase

Coordonator științific:
Prof. Dr. Ing. Ciprian Dobre
Dr. ing. Gabriel Guțu-Robu

BUCUREŞTI

2024

NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
POLITEHNICA BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Extraction of Scientific Competences

Ana-Maria Năstase

Thesis advisor:
Prof. Ciprian Dobre, Ph. D.
Gabriel Guțu-Robu, Ph.D.

BUCHAREST

2024

CONTENTS

SINOPSIS

Extragerea de cuvinte cheie este o tehnică folosită in Prelucrarea Limbajului Natural in scopul rezumării unui text, constând în identificarea celor mai semnificative cuvinte sau sintagme. Lucrarea curentă detaliază dezvoltarea unei aplicații pentru extragerea cuvintelor cheie, menită să detecteze cele mai relevante sintagme din publicațiile unor cercetători, pentru a rezuma activitatea științifică a acestora și a le evidenția competențele. În acest scop, au fost utilizate două metode diferite: YAKE, un algoritm de extragere de cuvinte cheie, și Latent Dirichlet Allocation, o metodă de modelare a tematicii, ceea ce a condus la îmbunătățirea rezultatelor și, în același timp, a reprezentat un prilej de a compara cuvintele cheie obținute prin fiecare dintre metode. Evaluarea rezultatelor a fost realizată calculând metrici pe baza sintagmelor cheie extrase dintr-un set de sinopsisuri ale articolelor științifice scrise de către cercetători din cadrul Universității Naționale de Știință și Tehnologie POLITEHNICA din București, demonstrând relevanța cuvintelor cheie rezultate.

ABSTRACT

Keyword extraction is a technique used in Natural Language Processing for summarization purposes, consisting of retrieving the most significant words or phrases from a text. This document details the development of a keyword extraction application that effectively identifies keywords from researchers' scientific publications, to summarize their research activity and reflect their competences. Two methods were employed for achieving this purpose: YAKE, a keyword extraction algorithm, and Latent Dirichlet Allocation, a topic modeling method, which not only led to better results, but also provided an opportunity to compare the outcomes of the two. The evaluation involved calculating precision and recall metrics for keywords extracted from some abstracts of researchers at the National University of Science and Technology POLITEHNICA Bucharest, demonstrating the relevance of the resulting keywords.

1 INTRODUCTION

Natural Language Processing (NLP) is a branch of Artificial Intelligence that leverages linguistic and mathematical principles to extract meaning from speech or textual data, and even to automatically generate human language. One major use of NLP in today's information-abundant world, with over 1 billion websites on the Internet[1], is keyword extraction, the process of identifying the most representative words or phrases that concisely reflect the essence of a text. This paper focuses on keyword extraction, detailing my application that extracts relevant phrases from researchers' abstracts to summarize their scientific contributions. In this section, I aim to provide an overview of the document, addressing the background, the purpose and the objectives of the project.

1.1 Background

The first steps in the field of Natural Language Processing were made more than 70 years ago, around 1950, when researchers began the first attempts at automatic translation. Until the 1980s, NLP was mostly led by linguists, who implemented rule-based methods centered around syntax rather than semantics. Then, NLP transitioned to machine learning algorithms, relying on statistical and probabilistic methods (Nadkarni et al., 2011), that are still widely used nowadays, especially in Information Retrieval tasks such as document indexing, document classification and sentiment analysis. Statistical techniques were the baseline methods until NLP began incorporating neural networks, which led to the revolutionary emergence of Transformer models starting in 2018[2]. Today, BERT and GPT models achieve remarkable results in complex NLP tasks such as summarization, question answering, translation and text generation. This marks the beginning of an era where natural language processing models are available to the general public in the form of digital assistants and chatbots integrated in various web or mobile applications.

Information Retrieval (IR) is a field that focuses on identifying and ranking relevant information from vast document collections, in order to improve database queries, document searching and filtering and recommendations based on user preferences (Arora et al., 2016; Nomoto, 2022). Keyword extraction, the selection of significant terms from a text, and topic modeling, the uncovering of hidden topics in large document collections, are two tasks that are positioned at the intersection of IR and NLP, being useful for IR purposes, while making use of linguistic characteristics and NLP techniques, such as tokenization, lemmatization, part of speech tagging, named entity recognition and word vectors.

1.2 Problem

CRESCDI[3] is a platform dedicated to researchers at the National University of Science and Technology POLITEHNICA Bucharest (UNSTPB), where users can access profiles of

[1] https://siteefy.com/how-many-websites-are-there/
[2] https://huggingface.co/learn/nlp-course/chapter1/4
[3] https://crescdi.pub.ro/#/

researchers, including information about them and their scientific publications. Each profile contains a “Keywords” section, where researchers can manually add terms that best highlight their competences and research activity, so that other academic professionals or students can easily see them by accessing the CRESCDI profile. We want to automate the process of extracting keywords from each researcher’s publications. This is the focus of this paper, which I will detail further.

1.3 Objectives

In this project, I aimed to use state-of-the-art software tools and algorithms to extract the most relevant words or phrases from the abstracts of CRESCDI authors, terms that effectively capture the area of interest of each UNSTPB researcher.

In my implementation, I employed two distinct techniques to identify the keyphrases, Yet Another Keyword Extractor (YAKE), a statistical keyword extraction algorithm, and Latent Dirichlet Allocation (LDA), a probabilistic topic modeling method. Consequently, another purpose of this study is the comparison between these two methodologies.

1.4 Structure

This subsection briefly presents the chapters that will be elaborated in the subsequent chapters of this document. In the next chapter, “State of the Art”, I will describe the NLP techniques and software libraries employed in my implementation, as well as other similar available tools. The chapter is divided into five subsections, where the first four each represent one task of the project: language identification, particularly discussing the fastText Python package, text preprocessing, focusing on the spaCy NLP library, keyword extraction, subsection that mentions a couple of related algorithms and then largely describes YAKE, and topic modeling, which centers around LDA. In the last subsection, I will talk about Gensim, an NLP library specifically meant for topic modeling.

The third section illustrates the way I integrated the previously discussed state of the art NLP tools in my project, to implement a solution for the task of extracting key themes from scientific abstracts. The “Solution” section describes the filtering of the abstract list of an author, the keyword extraction with YAKE and the topic modeling with LDA, as well as the distinct steps of text preprocessing before applying YAKE and LDA. The fourth chapter provides more implementation details, including code examples.

The fifth section reveals some results of my application, along with a thorough evaluation using metrics, then presents a comparison between the keywords obtained using YAKE and LDA.

Finally, the results of this project are summarized in the “Conclusion” section, and some suggestions are presented, regarding future improvements and related work.

2 STATE OF THE ART

In the field of Natural Language Processing (NLP), keyword extraction is essential for text mining purposes, such as automatic indexing, summarization, classification, automatic filtering and topic detection (Zhang, 2008). The necessity of accurately extracting keywords and topics from documents has led to the development of numerous algorithms. However, unlike other computer science fields that have specific methods for performing tasks, NLP lacks a unique solution for a specific concern (Campos et al., 2018). There is no universal approach for effectively extracting semantic information without requiring either large manually labelled datasets for training or a comprehensive preprocessing of the analyzed texts, according to the specifics of the domain and of the individual document. Consequently, advanced NLP tasks require the usage of more than one software tool or algorithm in order to analyze multiple approaches aimed at achieving expected results. This chapter describes preprocessing techniques followed by the exploited NLP tools, namely YAKE![4] (Yet Another Keyword Extractor) for keyword extraction, and LDA (Latent Dirichlet Allocation) for topic modelling. The spaCy Python library[5] was used to perform preprocessing on texts, part of speech tagging and named entity recognition. The fastText library was used for language detection[6], and the Gensim Python package for the implementation of LDA[7].

2.1 Preprocessing Pipeline

spaCy is a state-of-the-art natural language processing Python library that offers a customizable processing pipeline, with components capable of tokenization, lemmatization, recognizing named entities, tagging parts of speech and dependency parsing (*SpaCy*, n.d.):

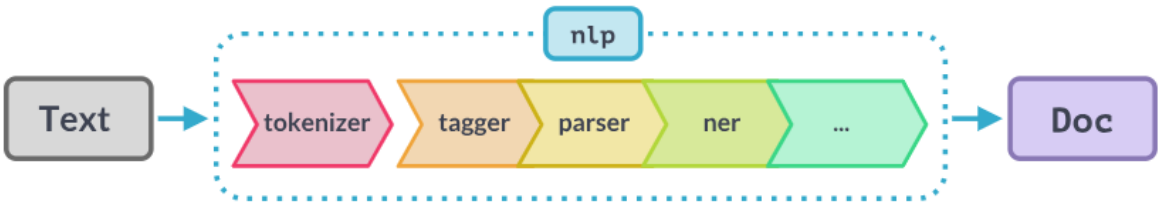

Figure 1: spaCy pipeline

Figure 1 shows the default components of the spaCy preprocessing pipeline. It may be modified by disabling or appending custom components, which are either developed by the user or available for use in the open-source community.

The motivation behind spaCy is explained by its creator, Matthew Honnibal, who wanted to develop a fast, efficient and compact language processing tool, suitable for production environments, as opposed to older libraries, which he believes offer too many features that are otherwise neither effective nor up to date with the advancements in computer science

[4] https://liaad.github.io/yake/
[5] https://spacy.io
[6] https://fasttext.cc/docs/en/language-identification.html
[7] https://radimrehurek.com/gensim/

(Matthew Honnibal, n.d.). Honnibal explained why he did not contribute to popular NLP library NLTK instead of starting a new project, stating that he thinks the maintainers of NLTK should not keep adding to the project, but “throw almost all of it away”.

The reason why numerous NLP packages, NLTK included, became so large yet not industry-grade is simply because this is not their target, given the fact that they are developed by researchers for researchers and students (Srinivasa-Desikan, 2018).

In the following sections, I will detail the default components of the spaCy pipeline, specifically the components of the trained model that I used in my project, “en_core_web_lg”, a large general-purpose pipeline, trained on web articles, with 514,000 word vectors (*SpaCy*, n.d.).

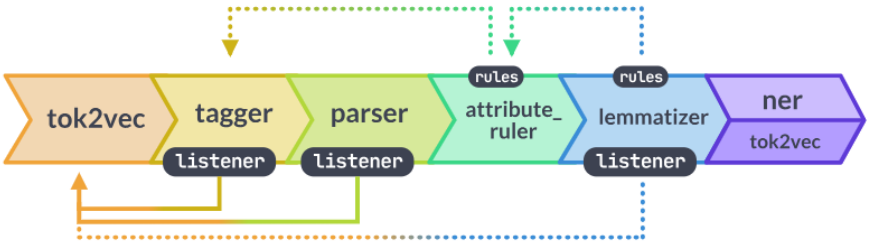

Figure 2: en_core_web_lg pipeline (*SpaCy*, n.d.)

The components of the common spaCy models are pictured in figure 2, along with their dependencies.

1) The Tokenizer

The tokenizer is not present in the figure above, since it is considered different from the other components in that it receives a string as an input and returns a Doc object, while the others receive the Doc and return a processed version of it. Despite this, the tokenizer is an essential pipeline step, and it is the first to run, before the tok2vec component in the case of the en_core_web_lg model.

The Doc returned by this pipeline element contains the tokens obtained by splitting the text into words, whitespaces and punctuation marks. The tokenizer treats special cases according to the language, for instance it splits “Let’s” or “don’t” into two tokens even though they are not delimited by a whitespace, while it considers acronyms like “U.K.” and “N.Y.” as single tokens.

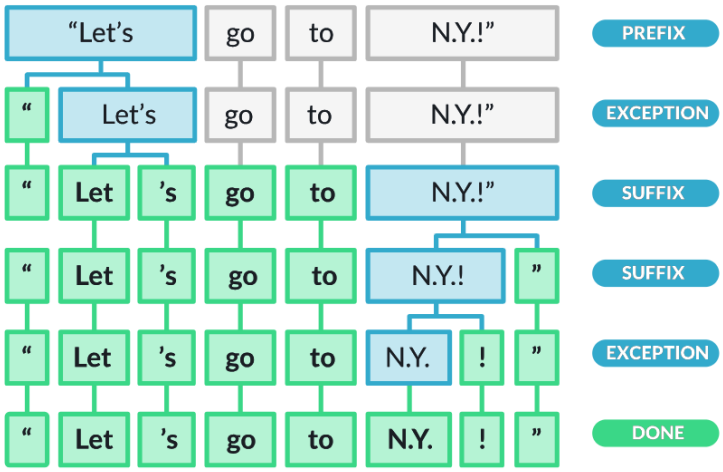

Figure 3: spaCy tokenizer

Figure 3, taken from spaCy’s official documentation (*SpaCy*, n.d.), shows how spaCy tokenizes a sentence: it splits it by whitespaces first, then it treats special cases such as exceptions (“Let’s” and “N.Y.”), prefixes (opening quotes) and suffixes (exclamation point, closing quotes) from left to right.

2) The tok2vec

The token-to-vector component transforms the tokens into word vectors, numerical representations that encapsulate the semantics of words. Other components that rely on word embeddings can use the results made available by the tok2vec layer through listeners, as it can be seen in figure 2 in case of the tagger, the parser and the lemmatizer. However, the named entity recognizer (NER) is intended to be independent of the other steps so that it can be used by itself, hence it contains its own tok2vec layer.

3) The Part of Speech Tagger

The POS tagger makes statistical predictions about the syntactic role of each token. The Token class contains two attributes that provide information about the part of speech: “pos_” and “tag_”, which goes into more detail. I will illustrate the tagger using the same sentence from figure 3.

Table 1: POS tags

TOKEN	POS	TAG
“	PUNCT	punct
Let	VERB	VB (verb, base form)
‘s	PRON	PRP (pronoun, personal)
go	VERB	VB
to	ADP (adposition)	IN (conjunction, subordinating or preposition)
N.Y.	PROPN (proper noun)	NNP (noun, proper singular)
!	PUNCT	punct
“	PUNCT	punct

I obtained the results in table 1 by applying the spaCy pipeline on the chosen sentence and printing the pos_ and tag_ attributes of each token, as well as the explanation of what the abbreviations mean, using spacy.explain().

4) The Dependency Parser

The dependency parser builds a dependency graph that models the syntactic structure of a phrase, where words are represented as nodes and the edges are labelled with the relationships between words.

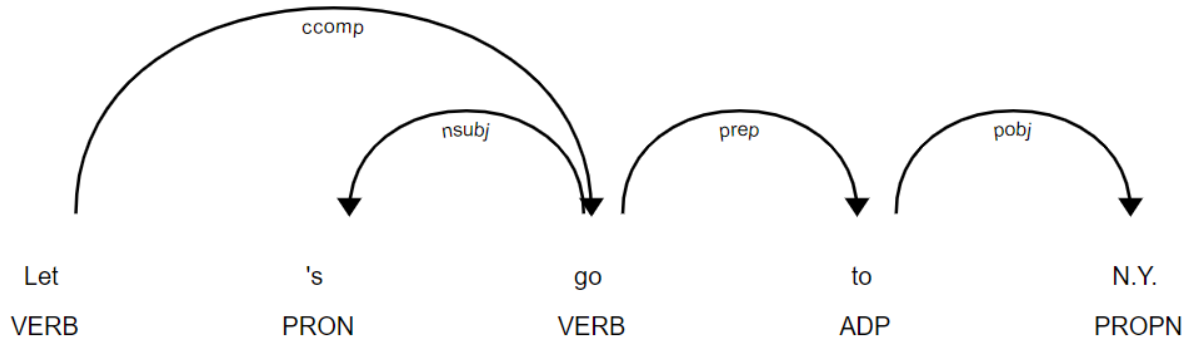

Figure 4: displaCy dependency visualizer

The following dependencies are pictured in figure 4, which I made using displaCy visualizer[8]:

- ccomp - clausal complement
- nsubj - nominal subject
- prep - prepositional modifier
- pobj - object of preposition

5) The Attribute Ruler

The attribute ruler layer allows users to specify some rules for setting token attributes that may be incorrectly handled by other pipeline components such as POS tagger, ner or lemmatizer. This step is suitable in case of domain specific contexts or unusual uses of some words.

6) The Lemmatizer

The lemmatizer assigns base forms to tokens based on the POS tag, morphological rules and dictionary lookups. For instance, the lemma of the "'s" token in the previously analyzed sentence "Let's go to N.Y.!" is "us", while the other lemmas coincide with the initial words.

[8] https://demos.explosion.ai/displacy

7) The Named Entity Recognizer

The entity recognizer is a neural network model capable of annotating different types of proper nouns and numerical values with the following labels: CARDINAL, DATE, EVENT, FAC, GPE, LANGUAGE, LAW, LOC, MONEY, NORP, ORDINAL, ORG, PERCENT, PERSON, PRODUCT, QUANTITY, TIME, WORK_OF_ART (*SpaCy*, n.d.).

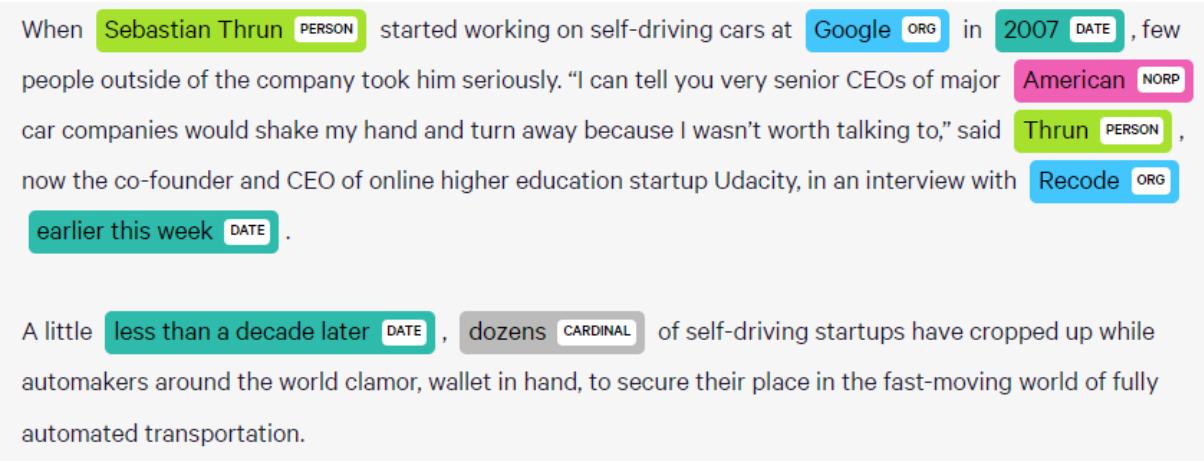

Figure 5: displaCy entity visualizer [9]

Figure 5 shows an example of a text labelled by the spaCy named entity recognizer.

According to the spaCy documentation, the part of speech tagger in en_core_web_lg has an accuracy of 0.97 and its named entity recognizer has a precision of 0.85, a recall of 0.86 and an F-score equal to 0.85.

2.2 Detecting the Language of a Text

fastText is a linear text classifier developed by Facebook AI Research (Joulin et al., 2016), that achieves an accuracy similar to that of deep neural networks classifiers, while being remarkably faster. The training of fastText lasts less than ten minutes for more than one billion words and the classification of 500,000 sentences takes less than one minute on an average multicore CPU.

fastText represents the text as a bag of n-grams in order to retain some information about the order of words, that the simple bag of words model does not consider. Words and n-grams are converted into word vectors.

During training, the probability distribution over classes is computed using hierarchical softmax. The classes are organized in a binary decision tree and the probability of a class is computed by multiplying the probabilities of all the nodes visited during a depth first traversal from root to the leaf node which corresponds to that class. The purpose of training is to learn

[9] https://demos.explosion.ai/displacy-ent

the configurations of matrixes A and B that minimize the loss between the predicted classes and the actual labels (Joulin et al., 2016):

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)) \quad (1)$$

In formula (1), N is the number of documents, y_n is a vector which indicates the correct label of document n ($y_n[k] = 1$ if document n belongs in class k, otherwise $y_n[k] = 0$), x_n is the normalized vector of n-gram features and f is the softmax function described above, thus $f(BAx_n)$ is the probability distribution over classes for document n. The algorithm uses stochastic gradient descent to find the parameters A and B that minimize the value of the negative log-likelihood formula (1) (Joulin et al., 2016).

Joulin, Grave, Bojanowski, Douze, et al. (2016) described some model compression techniques that can drastically reduce the memory usage without greatly impacting the accuracy: product quantization (splitting the word vectors into smaller subvectors and then quantizing them using nearest neighbour), pruning the vocabulary (keeping only a fixed number of the most relevant words and n-grams), hashing trick and Bloom filter (hashing words and n-grams in order to reduce the size of the dictionary and using Bloom filters to check the presence of a term in the dictionary in constant time).

fastText can be used for a variety of NLP tasks, out of which language identification of a text is a common scenario. In our case, it was used to detect the language of a publication.

I used the Python fastText library for the language identification purpose[10]. It provides two models capable of recognizing 176 languages, one with a size of 126MB and the compressed version with a size of 917kB. I chose to use the larger model, because it offers slightly superior speed and accuracy.

2.3 Keyword Extraction

Supervised methods have been the prevailing approach in keyword extraction, but they require extensive training and a lot of tedious work performed on a manual basis for annotating large collections of documents (Campos et al., 2020). Moreover, these machine learning models may not perform well on domains that are distinct from the ones they were trained on.

Despite being an unsupervised method, TF-IDF (Term Frequency-Inverse Document Frequency) needs an extended collection of documents since it assumes that a term is relevant in a specific text if it occurs frequently in that document. On the contrary, rare terms across the collection might emphasize more important terms. The formula used to measure the significance of a word is (Nomoto, 2022):

$$I_{ij} = f(w_{ij}) * \log_2 \frac{n(D)}{\sum_j g(w_{ij})} \quad (2)$$

[10] https://fasttext.cc/docs/en/language-identification.html

The first term in the equation (2) refers to Term Frequency (frequency of word w_i in document *j*), while the second term is known as the Inverse Document Frequency, where the numerator represents the total count of documents. The denominator is the number of documents that contain word w_i (Nomoto, 2022).

RAKE (Rapid Automatic Keyword Extraction) is an unsupervised keyword extractor that makes use of a co-occurrence matrix, thus rewarding with a higher score the words that appear not only frequently, but also in longer phrases (Rose et al., 2010):

$$score(w) = \frac{\deg(w)}{freq(w)} \tag{3}$$

In formula (3), the degree (deg) is the total length of the potential keyphrases that word w appears in, and the frequency represents the count of w in those sentences.

The score of a candidate keyphrase is computed by adding up the scores of the component words.

The subsequent unsupervised algorithm is YAKE, the approach that I chose for my implementation. YAKE ranks candidate keywords by calculating a score based on statistical text features. It does not rely on a pre-labelled corpus and it is equally effective regardless of domain or language, only requiring a stop word list specific to the language it is applied on (Campos et al., 2020). YAKE's ability of operating on individual documents without the necessity of additional corpora makes it suitable for extracting keywords from a set of technical abstracts, a continuously expanding and changing collection (Rose et al., 2010).

Algorithm 1 Keyword extraction procedure.

Input: *text*, *w*, *n*, θ, *language*
1: # (**Step 1**) Text pre-processing and candidate term identification
2: *sentences* = split *text* into sentences
3: **for** each *sentence* in *sentences* **do**
4: Pre-process (*sentence*, *language*)
5: **end for**
6: # (**Step 2**) Feature extraction & (**Step 3**) Term score
7: **for** each *term* in *sentences* **do**
8: Feature extraction
9: Compute *term* weight
10: **end for**
11: # (**Step 4**) n-gram generation
12: **for** each *sentence* in *sentences* **do**
13: *chunks* = split *sentence* into chunks
14: **for** each *chunk* in *chunks* **do**
15: Build *n*-gram *candidateKeywords* list
16: **end for**
17: **end for**
18: # (**Step 4**) Candidate keyword score
19: **for** each *candidate* in *candidateKeywords* **do**
20: Compute *candidateKeywords* weight
21: **end for**
22: # (**Step 5**) Data deduplication
23: **for** each *candidate1*, *candidate2* in *candidateKeywords* **do**
24: **if** DistanceSimilarity(*candidate1*, *candidate2*) $> \theta$:
25: Remove *candidate* from *candidateKeywords*
26: **end for**
27: # (**Step 5**) Ranking
28: *Keywords* = sort (*candidateKeywords*) by ascending score
Output: (*Keywords*, *score*)

Figure 6: YAKE Algorithm (Campos et al., 2020)

Figure 6 presents YAKE as pseudocode, highlighting the five steps of the algorithm (Campos et al., 2020):

1) Text pre-processing and candidate term identification

The text is split into sentences, sentences are split into chunks delimited by punctuation marks and chunks are divided into tokens. Then, stop words are removed and tokens are tagged with the following labels: d (digit or number), u (unparsable), a (acronyms), u (uppercase), p (parsable).

2) Feature extraction

The algorithm favors tokens that appear repeatedly in uppercase or abbreviated as an acronym and assumes a term is more important if it can be found closer to the beginning of the text. Therefore, it proceeds by calculating some statistics for each term: the frequency, the number of times that the word occurs capitalized inside sentences, the number of times that the term appears as an acronym and the positions of the sentences that contain it in the document.

Afterwards, YAKE calculates five features for each token and combines these attributes into the following formula which quantifies the significance of a token:

$$S(t) = \frac{T_{Rel} * T_{Position}}{T_{Case} + \frac{TF_{Norm}}{T_{Rel}} + \frac{T_{Sentence}}{T_{Rel}}} \quad (4)$$

In formula (4), T_{Case} evaluates the casing dimension, taking into account the frequency of occurrence with an initial capital letter and the frequency of the acronym. The $T_{Position}$ factor measures the importance of a word based on all of its positions in the text, more precisely the indices of the sentences it occurs in. The "term frequency normalization" (TF_{Norm}) represents the frequency of a word, normalized to ensure that, in large documents, this element does not overweigh the other features in the formula. The fourth component, "term relatedness to context" (T_{Rel}), aims to downgrade the words that have similar characteristics to those of stop words, specifically a high number of occurrences with many different words around them. The last feature in the formula is $T_{Sentence}$ ("term different sentence"), which rewards tokens that appear in numerous sentences.

3) Computing term score

The score of each token is determined using formula (4), the most significant terms having the lowest score.

4) Generating n-grams and calculating the scores of potential keyphrases

In the fourth stage, the YAKE algorithm finds potential keywords by extracting n-grams (with the maximum n given as input to the algorithm) from the chunks obtained in the first step, choosing only the phrases that do not contain tokens marked as "digit" or "unparsable" and do not start or end with a stop word.

The score of a keyword kw is determined as:

$$S(kw) = \frac{\prod_{t \in kw} S(t)}{KF(kw) * (1 + \sum_{t \in kw} S(t))} \quad (5)$$

In equation (5), the multiplied scores of the component tokens of kw are divided by the sum of these scores amplified by the frequency KF of the keyword. Consequently, the algorithm can differentiate between potential keywords that contain the same words reordered and choose the most frequent one.

5) Handling duplicates

After ranking the keywords by the lowest score, YAKE removes duplicates using a similarity measure, which can be either the Levenshtein similarity, the Jaro-Winkler similarity or the sequence matcher. Two keywords are considered duplicates if their distance, computed using one of the three previous methods, surpasses a deduplication threshold chosen before starting the algorithm and given as parameter. When two keyphrases exceed this similarity threshold, the one with the higher score is removed from the final keyword list.

YAKE returns a list of tuples that contain a keyword and its score, ranked by their importance, so from the lowest score to the highest.

2.4 Topic Modeling

Topic modeling methods aim to discover “hidden” (latent) topics from large collections of documents. Each topic consists of a set of words that frequently occur together and thus it is assumed that they are conceptually related. Given its capacity of capturing semantic features of corpora, topic modeling is an effective tool in text mining and information retrieval, used in text summarization and sentiment analysis applications (Abdelrazek et al., 2023).

There are four categories of topic models: algebraic, fuzzy, probabilistic and neural (Abdelrazek et al., 2023).

The most notable algebraic topic modeling method is Latent Semantic Indexing (LSI) (Deerwester et al., 1990), that represent a collection of documents as a term-document matrix, where an element of the matrix is the frequency of a word in a document (a row corresponds to a word, while documents are placed on columns). Deerwester et al. (1990) decomposed the term-document matrix using singular value decomposition (SVD) in order to obtain a vectorial representation for words and texts. Geometrically, the distance (calculated as cosine similarity) between these vectors indicates the degree of semantic correlation between two terms, two documents or a term and a document. Although LSI is proficient in capturing synonymy, it does not handle polysemy with the same efficacy (Deerwester et al., 1990).

LSI is not based on a solid statistical foundation and a flaw that it presents is the false assumption that words and documents follow a joint Gaussian distribution (Abdelrazek et al., 2023). These concerns are solved by the probabilistic approaches, which, according to

Abdelrazek et al. (2023), dominated the field before neural topic models began their rising around 2015.

Latent Dirichlet Allocation (LDA), a generative probabilistic model developed by Blei et al. (2003), remains one of the most popular topic modeling methods. LDA is characterized by a series of assumptions:

- documents and words are exchangeable, their order does not matter, concept known as the bag of words model (Blei et al., 2003);
- each document is a probability distribution over topics and the distributions of topics in all documents share the same Dirichlet prior (Jelodar et al., 2017);
- each topic is a distribution over words and the distributions of words across topics are characterized by a common Dirichlet prior (Jelodar et al., 2017).

The generative process of LDA begins by choosing the Dirichlet priors α and β, corpus-level parameters. Afterwards, for each document, the algorithm chooses the distribution θ of topics in the current document from a Dirichlet distribution of parameter α. Then, for each of the N words of the document, a topic z_n is sampled from distribution θ and a word w_n is chosen from the distribution of this topic (Blei et al., 2003). Thus, the generative process of LDA iteratively calculates the probability distribution of topics in documents and the distribution of words within topics, by randomly assigning them based on a probability distribution, until the results converge.

Griffiths & Steyvers (2004) used LDA to extract topics from scientific abstracts and achieved meaningful results, as in the following figure:

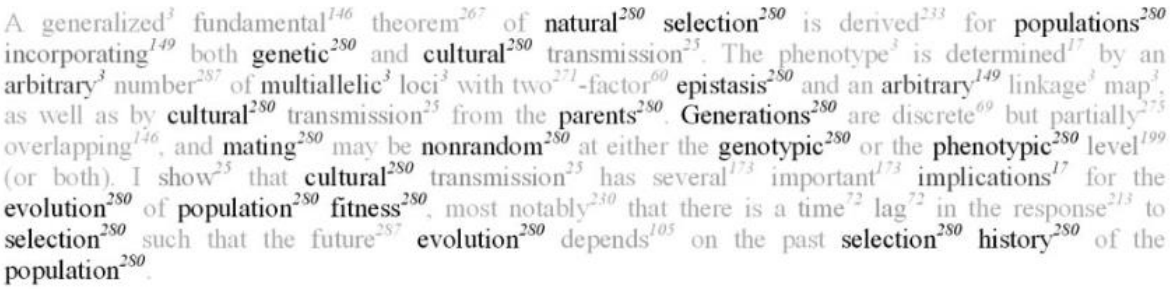

Figure 7: An abstract with the words labelled according to the topic they belong in (Griffiths & Steyvers, 2004)

In figure 7, the words in an abstract are tagged with a number corresponding to the topic they belong in. The highlighted words are extracted from the topic with the highest probability, hence they can be considered as keywords of the abstract. According to Griffiths & Steyvers (2004), these words accurately summarize the content of the document.

The limitations of LDA stem from the variability of its parameters which need to be selected before running the algorithm: the Dirichlet priors α and β and the number of topics. Although these parameters can be fine-tuned using variational inference or Gibbs sampling, their variations significantly impact the performance of the algorithm (Griffiths & Steyvers, 2004).

The value of α determines the number of topics being assigned to a document (a higher value leads to more topics), while the value of β controls the sparsity of the topic-word distributions (a higher β determines the algorithm to assign similar probabilities to more words). Regarding the number of topics, a small number may lead to underfitting, while a large number of topics generally results in overfitting the data (Tijare & Rani, 2020).

Another considerable issue is that LDA requires a thorough clean-up and preprocessing of the documents; stop word removal and lemmatization are very important, otherwise stop words will occur among the words that define topics and different forms of the same word will be considered separately (Moreno-Ortiz, 2024).

Tijare & Rani (2020) compared the results of LDA on social media data, using the implementations from popular Python packages Gensim and scikit-learn, and visualizing them with pyLDAvis, a tool which graphically represents topics. They also computed coherence scores, using some statistical formulas to evaluate topics. Röder et al. (2015) discussed multiple coherence measurements for topic models, comparing their results with human ratings, considered the gold standard for evaluating topics. They proposed a new coherence measurement, CV, which is a combination of other existing formulas and, according to Röder et al. (2015), achieves the best performance in evaluating topic models, having the strongest correlation with the human ratings.

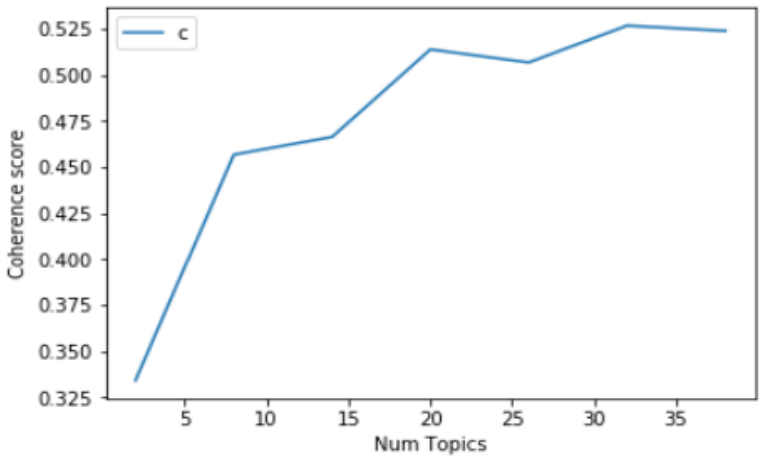

Figure 8: Coherence score for scikit-learn LDA (Tijare & Rani, 2020)

Figure 8 shows the coherence score depending on the number of topics. Despite the score rising, Tijare & Rani (2020) observed that topics begin to overlap when there are more than 10 of them.

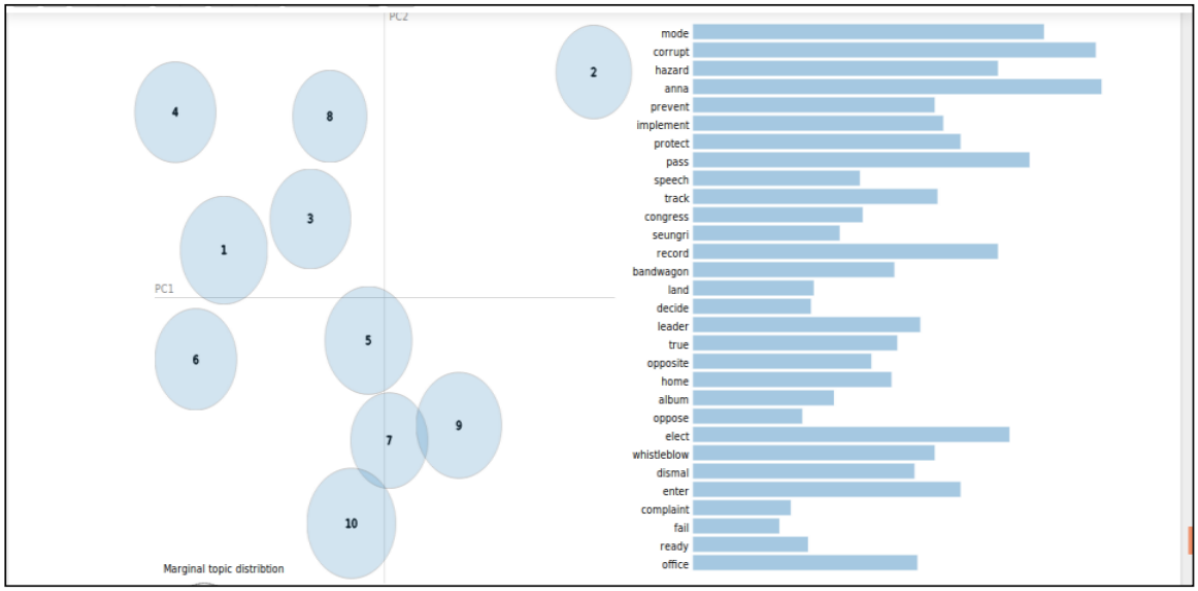

Figure 9: pyLDAvis representation of LDA results obtained with Gensim

In figure 9, which depicts 10 topics extracted using the Gensim implementation of LDA, topics 7, 9 and 10 slightly intersect, indicating they share common terms.

Tijare & Rani (2020) also compared LDA with LSI and NMF, another topic modeling technique, and concluded that LDA achieves better coherence scores. The CV score for LDA was 0.47531674689 and 0.28887862599 for LSI. LDA has the higher score, thus the higher coherence.

2.5 Gensim

Gensim is an open-source natural language processing Python library[11] focused on topic modeling, specifically unsupervised semantic modeling methods like the ones previously discussed in this paper: Latent Semantic Indexing/Analysis and Latent Dirichlet Allocation. Radim Řehůřek, the creator of Gensim, acknowledged the lack of easy to use and efficient NLP libraries[12] and committed to bridging the "gap between academia and ready-to-use software packages" (Řeh uřek & Sojka, 2010).

Gensim is based on two fundamental ideas, according to Řeh uřek & Sojka (2010): document streaming, which guarantees memory independence by not storing the whole large corpus in RAM, and transformations between vector spaces. Gensim relies on the Vector Space Model, the paradigm of representing documents as vectors of features. This model allows for easy estimation of document resemblance through cosine similarity and facilitates the transformation of documents from one model representation to another (for instance from TF-IDF to LSA) by performing a vector space translation.

[11] https://radimrehurek.com/gensim/index.html
[12] https://radimrehurek.com/gensim/intro.html

For our experiment, the LdaMulticore model from Gensim was used for training LDA. It is a parallelized version of the LDA algorithm, capable of significantly reducing the training time[13]:

Table 2: LdaMulticore and LdaModel training time

Algorithm	Training time
LdaMulticore(workers=1)	2h30m
LdaMulticore(workers=2)	1h24m
LdaMulticore(workers=3)	1h6m
old LdaModel()	3h44m
iterating over the corpus	20m

Table 2, taken from the Gensim documentation, shows the duration of training LdaMulticore and the old LdaModel on all the English articles from Wikipedia, measured on an i7 server with 4 physical cores. After deducting the 20 minutes spent iterating over the documents, it can be concluded that LdaMulticore, even when it runs on a single thread, is almost 3 times faster than the basic LdaModel, while with 2 workers it achieves a speed-up equal to 6 and with 3 cores the speed-up reaches 8.35.

[13] https://radimrehurek.com/gensim/models/ldamulticore.html

3 SOLUTION

The proposed solution integrates a variety of natural language processing techniques and tools that include statistical, probabilistic and machine learning methods, both supervised and unsupervised. Together, along with a carefully thought-out preprocessing, tailored to the specific NLP tools employed and the collection of data to be processed, they form an effective system, capable of discovering themes from researchers' abstracts. I implemented two different methods to achieve this: keyword extraction with YAKE and topic modeling using LDA, two approaches that complement each other and manage to extract meaningful words that reveal the sphere of specialization for any author.

I chose to develop my solution using the Python programming language, the most suitable for natural language processing applications for its speed, versatility and collection of the most powerful NLP libraries, as described in section 2.

In the current chapter, I will outline the architecture of my solution, which includes three main steps for obtaining the keywords from a researcher's abstracts: cleaning up the abstract collection, extracting keywords with YAKE, which contains another preprocessing step at the abstract level, and extracting significant words with LDA, which also integrates a clean-up stage.

3.1 Filtering the Abstract List

Before preprocessing the abstracts and extracting themes from them, it is important to remove the texts that do not contain any useful information, to prevent them from negatively impacting the results and to reduce the unnecessary use of memory and computational resources.

During this step, the program takes an author's list of abstracts, extracted from the research platform, and first filters out the empty abstracts or the abstracts that are not composed of words. Among the top 20 authors on the platform, with the most publications, there were 3416 abstracts extracted from the database that were either null or did not contain any alphabetic characters.

From the non-empty texts that contain letters from the English alphabet, abstracts that are not written in English are removed, using the fastText library for language prediction. Besides the fact that the presence of these non-English abstracts in the corpus would unnecessarily take up resources, it would also impact the results. The English spaCy pipeline cannot parse and label non-English texts correctly and the YAKE algorithm is not able to handle this kind of documents either, since it relies on a static stop word list to identify the irrelevant terms. With fastText, I found 269 non-English abstracts among the top 20 researchers. Given the fact that LDA computes a higher probability of appearance within a topic for the more frequent words in the corpus, this means that there is a high chance that some of the most frequent non-English words will appear among the keywords extracted with LDA. For instance, I tried applying LDA to the abstracts of the author who has the most abstracts not written in English,

to extract one topic, without priorly removing the non-English texts, and the Romanian preposition “de” was among the top 15 words.

The next step in the corpus preprocessing entails the use of spaCy for the named entity recognition (NER) pipeline, in order to remove the abstracts that contain more person and organization names than useful information, step that I introduced after discovering that 77 of the input texts were nothing more than an enumeration of personal names and institutions. The spaCy named entity recognizer does not always label those names correctly, especially given the fact that they are not part of a clear context, for instance:

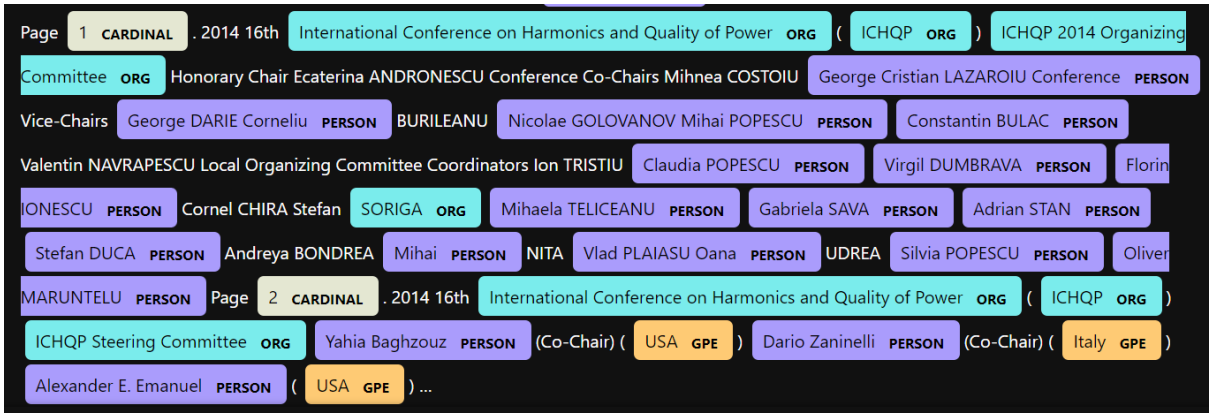

Figure 10: Preview of the NER functionality

Figure 10 represents the output of the displaCy visualizer and shows an example of an abstract in which the spaCy NER does not label all the named entities properly. “Mihnea COSTOIU” is one of the names incorrectly not labelled as “PERSON”, and it appears among the keywords extracted with YAKE:

Table 3: YAKE output with person names

scanning electron microscopy
ray diffraction
composite materials
materials
SEM
XRD
drug delivery systems
properties
dielectric properties
Chairs Mihnea COSTOIU
Chairs BURILEANU Valentin
iron oxide nanoparticles
BMT ceramic material
transmission electron
microscopy

Table 3 shows two researchers' names in the output of YAKE, as proper nouns are more likely to be extracted by YAKE, because it considers capitalized words more important.

Table 4: YAKE output without names

scanning electron microscopy
ray diffraction
composite materials
materials
SEM
XRD
drug delivery systems
properties
dielectric properties
iron oxide nanoparticles
BMT ceramic material
transmission electron
microscopy
method
electron

In table 4, the output of YAKE for the same author, after removing those input texts, the two personal names do not occur among the keywords anymore. Therefore, it is essential to filter out texts that contain mostly names before extracting keywords.

3.2 Extracting Keywords with YAKE

The next step of the proposed solution is the keyword extraction using YAKE. This algorithm does not require a comprehensive preprocessing, since it uses punctuation and stop words to identify candidate keyphrases. However, I removed some words that I did not want to appear in the resulting keyword list.

3.2.1 Preprocessing

Firstly, before extracting the keywords, I concatenated all the abstracts of a researcher into a single string, because YAKE operates on individual documents. For preprocessing the resulting text, I used the spaCy NER pipeline component to remove the following named entities: personal names, nationalities, religious or political groups, buildings, geopolitical and other types of locations, date and time entities, monetary values and other numerical figures. I did not eliminate organizations, as I observed that a lot of relevant terms that were written with initial uppercase letters were frequently classified by spaCy as organization names, such as "Big Data", "Cloud Computing" or "Natural Language Processing".

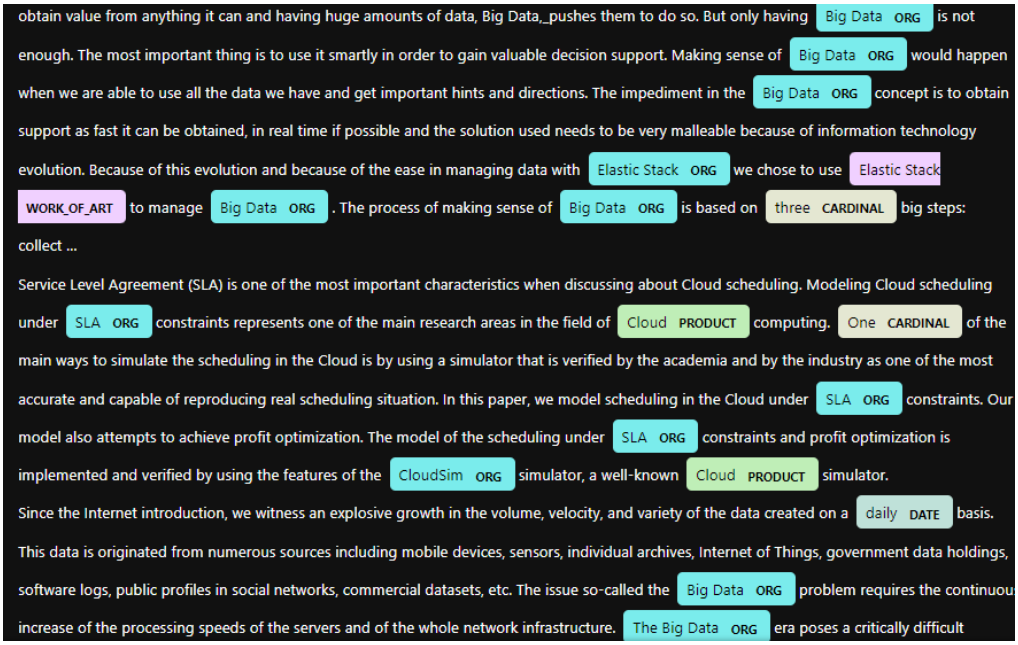

Figure 11: Significant terms incorrectly classified as ORG

Figure 11 pictures an abstract of one of the top 20 authors, where the “Big Data” term is labelled seven times as an organization, suggesting how significant concepts like this would be lost if I chose to remove “ORG” entities during preprocessing. To illustrate this, I extracted keywords from the publications of the same author, adding organizations to the list of entities to remove, and I obtained the keyphrases below:

Table 5: keyword results after removing ORGs

Cloud computing
Cloud systems
service level agreement
Cloud
data
grid scheduling algorithms
Cloud service providers
distributed systems resources
system
computing distributed systems
scheduling
Data management systems
data processing
big data platforms
big data

In table 5, “big data” appears once as a keyword and once inside another keyphrase, but it only appears in lowercase and at the bottom of the keyword list.

Table 6: keyword results without removing ORGs

Cloud computing
Big Data processing
Big Data
Cloud systems
data
Cloud
service level agreement
Cloud service providers
Data processing systems
distributed systems resources
grid scheduling algorithms
computing distributed systems
Identification System data
system
Big Data environments

On the other hand, when I kept the entities labelled as organizations in the text, “Big Data” appeared in three keyphrases, two of them being in the top 3 most relevant keywords, as we can see in table 6.

Nevertheless, there are some institution names that should be removed, because they are very frequent in these academia writings, such as names of universities and faculties. I prevented their occurrence among keywords by adding the words “university” and “faculty” to the YAKE’s internal stop word list, along with numerous other words specific to technical and scientific papers, that I encountered many times in keyword extraction results, such as “conference”, “workshop”, “experiment”, “professor” and “research”.

3.2.2 Keyword Extraction

After preprocessing the text, the program proceeds with the keyword extraction using the yake Python library[14]. I extracted 15 keywords for an author, because I think this number is enough to summarize the research activity of an author, without being so large that meaningless keyphrases are extracted, especially in the case of researchers with fewer publications.

[14] https://pypi.org/project/yake/

Table 7: Result of YAKE with 20 keywords

monitoring project budgets
project budget monitoring
percentage execution budgets
external drive magnet
cold plastic deformation
Human Resources Development
efficient project management
Sectoral Operational Program
magnetic drive pumps
plastic deformation equipments
total manufacturing cost
product procurement price
induced magnetic field
imposed major penalties
clauses imposed major
Process planning consists
budget chapters
avoiding fluid leakage
considerable advantages
project

Table 7 shows the results of my attempt to extract 20 keywords from the abstracts of an author who has 10 publications. Starting from the fourteenth keyword, there are some phrases that are not only irrelevant as keywords, but also meaningless from a grammatical perspective, for example “imposed major penalties” and “Process planning consists”.

Campos et al. (2020), the creators of YAKE, observed that the vast majority of keywords extracted by humans (gold keywords or ground-truth keywords) are unigrams, bigrams and trigrams. They experimentally obtained the best results with YAKE when extracting n-grams with the maximum value for n being equal to 3. Thus, I also chose trigrams as the maximum n-gram size in my solution.

For the similarity measure, I used the default option of the YAKE implementation, the Levenshtein distance. According to Campos et al. (2020), when using the Levenshtein similarity, the algorithm gives the best results if the widow size is equal to 1 (the number of words before and after a term which are counted when determining the number of words this term co-occurs with) and the deduplication threshold has a value of 0.8. However, for the purpose of my project I thought a threshold equal to 0.5 leads to the best results, because a bigger value produces too similar keywords and other relevant terms are lost, while a smaller

value causes some significant keyphrases to be excluded because of their similarity to other keywords. A lower deduplication threshold may be particularly problematic in the case of researchers with a few abstracts. Instead of selecting relevant keyword candidates with high similarity, the algorithm may extract some meaningless keyphrases in its attempt to remove duplicates.

Table 8: Comparison among deduplication threshold values

dedup_threshold = 0.2	**dedup_threshold = 0.5**	**dedup_threshold = 0.8**
Natural Language Processing	Natural Language Processing	Natural Language Processing
Cohesion Network Analysis	machine learning models	machine learning models
FPGA Spartan III	Language Processing techniques	Natural Language
models	language models	Language Processing
NLP	learning	Language Processing techniques
data	Cohesion Network Analysis	language models
learning techniques	FPGA Spartan III	learning models
system	game learning environment	trained language models
BERT	models	learning
finetuned BERT model	BERT model	language processing tools
information	cellular automata model	BERT language model
development	neural network models	Cohesion Network Analysis
Graph Convolutional Network	interactive natural language	FPGA Spartan III
social	Smart Learning Ecosystems	Language
cellular automata hardware	Latent Semantic Analysis	game learning environment

Table 8 compares the keyphrases extracted for a top 20 author with 380 papers, with three different values for the deduplication threshold. On the left, there are the keywords obtained for a low threshold of 0.2, which include some frequent but not very significant words like “data”, “system”, “information” and “development”. The terms on the right column, which resulted after setting the threshold to 0.8, are too similar, with the word “language” appearing in 9 out of 15 keywords, in approximately the same context (“Natural Language Processing”, “Natural Language”, “Language Processing” etc.). Therefore, a high deduplication threshold produces some redundant keywords and excludes some relevant ones (for instance “neural network model” and “Latent Semantic Analysis”), that are captured by choosing a 0.5 threshold, as pictured in the center column of the table above.

After setting these parameters, the program calls the keyword extraction function from the yake package and the 15 keyphrases are returned and printed.

3.3 Topic Modeling with Latent Dirichlet Allocation

The second tool that I used for extracting meaningful themes from research abstracts is Latent Dirichlet Allocation. Despite keyword extraction not being the purpose of LDA, which is typically used for indexing, summarization or classification, it can be successfully employed as a keyword extractor. The most representative words of the uncovered topics can be treated as keywords of the document collection, as long as the preprocessing step brings the corpus to an adequate form.

3.3.1 Preprocessing

Preprocessing plays a critical role when working with LDA, therefore this time I used the whole spaCy pipeline, not only the named entity recognizer. Firstly, stop word removal is especially important. Given the generally high frequency of stop words, their probabilities of occurring within topics may surpass that of meaningful terms. I removed the stop words from the default spaCy list, as well as the custom stop words that I discovered, the same used for YAKE. The named entity removal is also similar to the implementation described in the previous chapter about YAKE, as I removed the same entities.

Justeson & Katz (1995) studied the particularities of technical and scientific terminology and found that approximately 95.6% of terms are noun phrases. Among 800 terms, they found only 32 adjectives and 3 verbs, the rest of them being noun phrases. Thus, I used the part of speech tags assigned by the spaCy POS tagger to keep only the nouns and the adjectives, removing all the other parts of speech from the documents.

In the final tokenized corpus, I added the uppercase lemmas of the remaining tokens, so that different forms of the same word, such as “Language” and “languages”, result in the same token.

Afterwards, I used the Phrases collocation detection from Gensim, a statistical method that identifies words that generally co-occur, to build n-grams. By repeating the steps of training the model and running it on the corpus twice, I obtained bigrams, trigrams and even some 4-grams, which resulted from combining two bigrams. After adding the collocations to the corpus, I removed all of the unigrams, since around 70% of nouns or noun phrases that represent technical terms contain more than one word (Justeson & Katz, 1995).

Table 9: Comparison between LDA with and without unigrams

SYSTEM	COMPUTER SUPPORTED COLLABORATIVE LEARNING
MODEL	NATURAL LANGUAGE PROCESSING
METHOD	E LEARNING
ANALYSIS	SOCIAL NETWORK
KNOWLEDGE	CHAT CONVERSATION
LANGUAGE	READERBENCH FRAMEWORK
USER	NATURAL LANGUAGE
INFORMATION	ARTIFICIAL INTELLIGENCE
TOOL	LEARNING PROCESS
APPLICATION	NATURAL LANGUAGE PROCESSING NLP
LEARNING	LEARNING ENVIRONMENT
LEARNER	NATURAL LANGUAGE PROCESSING TECHNIQUE
PROCESS	POLYPHONIC MODEL
TIME	TEXTUAL COMPLEXITY INDEX
TOPIC	COMPUTER SCIENCE

I extracted the keywords for an author first with the unigrams present in the corpus and then without them, and the results are compared in table 9. If I kept the single-word tokens,

collocations would not occur among the keywords. While the words on the left column may be relevant in some topic modeling applications, the keyphrases on the right are much more meaningful for the current project. However, for authors with a reduced number of abstracts, the removal of unigrams might compress the document collection too much and LDA would not have enough input to work on, so I decided to exclude single-word tokens only if there are more than 100 terms left after their removal.

3.3.2 Topics and Keywords Extraction

After preprocessing, the tokens collection is a list of lists, each inner list containing the tokens from a document. LDA requires an integer form of the tokens and a bag of words representation of the document collection, so my proposed solution proceeds by creating a Dictionary object with Gensim, which assigns a unique ID to each token, and then building the bag of words, a representation of the corpus where the initial tokens are replaced by tuples of token ID and its frequency in the document.

Afterwards, I applied the LdaMulticore model from the Gensim library. Since I worked on a computer with 4 cores, I set the "workers" parameter to 3, as the official documentation[15] mentions that the optimal number of workers is one less than the physical cores. I discussed the scalability of LdaMulticore in the Gensim chapter (table 2).

The parameters "passes" and "iterations" control the number of iterations over the corpus and over the individual documents, respectively. The number of passes is the number of times that LDA iterates over the document collection, refining the document-topics and topic-words distributions each time, while the iterations parameter controls the number of times that the algorithm iterates over a document during a single pass through the collection. Increasing the values of these parameters generally leads to a better coherence of topics and a more accurate distribution of topics over the corpus, since more iterations give LDA the opportunity to learn better distributions. On my particular issue, I found that the passes and iterations do not influence the results notably. I ran LDA on the abstracts of two authors, a top 20 author with 646 publications and a researcher with only 5, and I discovered that there are almost no changes in the extracted keywords even if I greatly vary the parameters.

[15] https://radimrehurek.com/gensim/models/ldamulticore.html

Table 10: Comparison between passes and iterations values

5 publications		**646 publications**	
passes = 1 iterations = 1	**passes = 50 iterations = 300**	**passes = 1 iterations = 1**	**passes = 50 iterations = 300**
DEFORMATION	DEFORMATION	COMPUTER SUPPORTED COLLABORATIVE LEARNING	COMPUTER SUPPORTED COLLABORATIVE LEARNING
STEEL	STEEL	NATURAL LANGUAGE PROCESSING	NATURAL LANGUAGE PROCESSING
MECHANICAL	MECHANICAL	E LEARNING	E LEARNING
CHARACTERISTIC	CHARACTERISTIC	SOCIAL NETWORK	SOCIAL NETWORK
TEMPERATURE	TIME	CHAT CONVERSATION	CHAT CONVERSATION
STRUCTURE	TEMPERATURE	READERBENCH FRAMEWORK	READERBENCH FRAMEWORK
TECHNOLOGICAL	TECHNOLOGICAL	NATURAL LANGUAGE	NATURAL LANGUAGE
MITTAL	PARAMETER	NATURAL LANGUAGE PROCESSING NLP	ARTIFICIAL INTELLIGENCE
TIME	MITTAL	LEARNING PROCESS	LEARNING PROCESS
PARAMETER	STRUCTURE	ARTIFICIAL INTELLIGENCE	NATURAL LANGUAGE PROCESSING NLP
DEGREE	DEGREE	LEARNING ENVIRONMENT	LEARNING ENVIRONMENT
CHEMICAL	WELDABLE	NATURAL LANGUAGE PROCESSING TECHNIQUE	NATURAL LANGUAGE PROCESSING TECHNIQUE
SCIENCE	TREATMENT	TEXTUAL COMPLEXITY INDEX	POLYPHONIC MODEL
REGIME	CHEMICAL	POLYPHONIC MODEL	TEXTUAL COMPLEXITY INDEX
GALATI	TREATMENTS	COMPUTER SCIENCE	COMPUTER SCIENCE

The two columns on the left of table 10 show the keywords extracted for the author with fewer abstracts, the first obtained with passes and iterations both equal to 1 and the second with passes equal to 50 and iterations equal to 300. Only three keywords differ, however their relevance is difficult to estimate, since none of the keywords extracted with LDA for this author is very meaningful, given that there are only 5 abstracts to extract from. On the right of the table, there are the results of the same process applied for the author with 626 abstracts, showing no difference in the list of keywords, other than slight variations in their order. Therefore, I set both the passes and the iterations parameters to 1, for time optimization.

Finally, probably the most important parameter that one has to set when working with Latent Dirichlet Allocation is the number of topics, which affects the coherence of the resulting topics and may lead to underfitting or overfitting, as I discussed in the LDA section of chapter “State of the art” (2.4). In my project, I did not focus very much on the coherence of topics but rather on the keywords that I aimed to obtain.

I used the topic visualization tool pyLDAvis to analyze the extracted topics and their similarity. I applied LDA for the 20 authors with the most publications, setting the number of topics to 5, and I graphically represented them to see how often the topics intersect. All of the authors had overlapping topics at least once out of 3 algorithm runs, indicating that 5 topics are too many, even in the case of the researchers with the largest document collections.

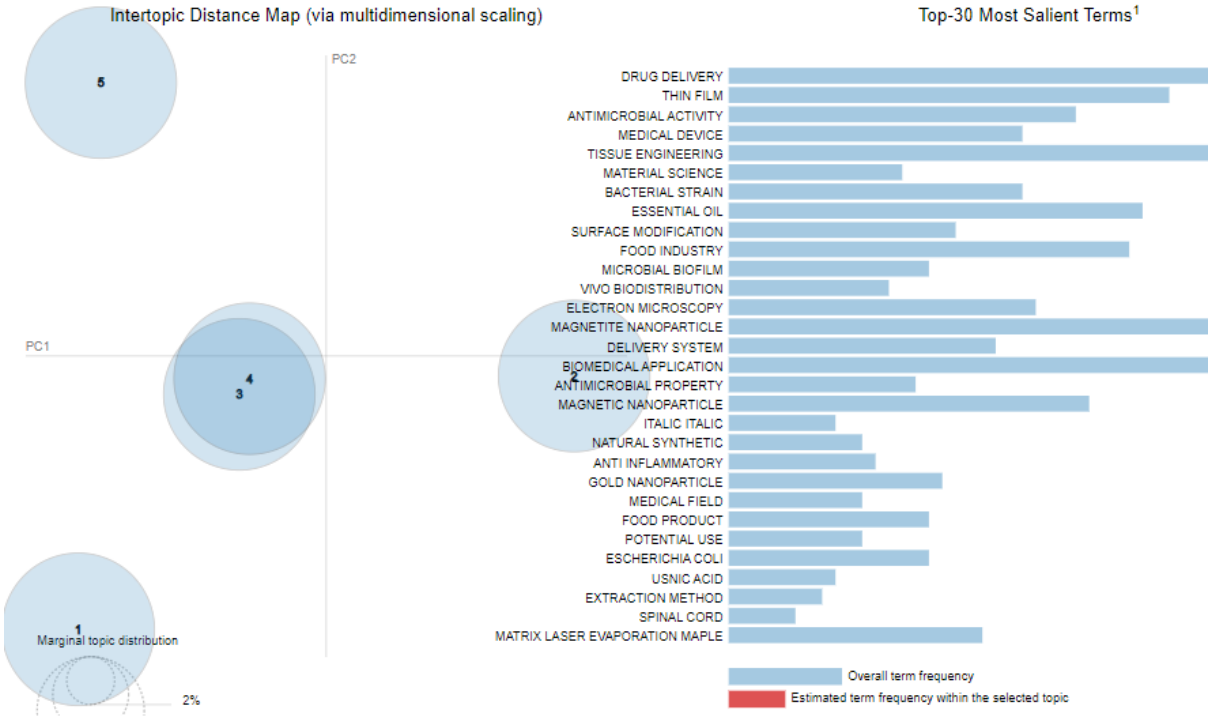

Figure 12: pyLDAvis graphic of 5 topics

Figure 12 illustrates the 5 topics extracted from the abstracts of the author who has the second most publications (712), showing that topics 3 and 4 are almost identical.

I reduced the number of topics to 2, but they were still too similar. By extracting 2 topics for each top 20 author and counting the duplicates between the topics, I observed that the 2 topics shared at least 11 common words out of 15, for every author. Consequently, the most relevant and non-redundant key terms are obtained when extracting a single topic.

4 IMPLEMENTATION DETAILS

In the current chapter, I will elaborate on the source code, I will present the input and some code snippets, and I will discuss the challenging issues of choosing a spaCy model and removing named entities during text preprocessing. The source code details will mostly focus on aspects that are not strictly related to the two main tools used for extracting keywords, YAKE and LDA, since all the parameters and the fine-tuning of these algorithms were detailed in section 3.

4.1 The Source Code

The program expects three command line arguments: the paths to the two input files and the name of the output file. The input files contain data in CSV format about the authors and about their publications respectively, information extracted from the database of the CRESCDI platform. The first one contains columns such as the id, the author's last name and first name, their email, their publication count and many more, while the second file contains information about the publications of the researchers in the first file, for instance the researcher's id, used to link this table with the authors one, the title of the paper and the abstract.

4.1.1 Parsing the Input Files

The source code of my keyword extractor application is a Python script called "keywords.py", available on GitHub[16]. I parsed the two CSV files using the pandas library[17] and I extracted the important data that the program needs to extract the keywords: the authors' ids, their names and their abstracts. Then, the script iterates through the authors and filters the abstract list, then extracts keywords with YAKE and then with LDA.

4.1.2 Filtering the Abstract List

The *clean_abstracts* function takes the list of abstracts extracted from the input file and performs the clean-up described in chapter 3.1. The following code block shows how I used fastText to detect the language of each abstract and remove non-English abstracts:

```
# load fastText model
model = fasttext.load_model('lid.176.bin')

new_abstracts = []

for abstract in abstract_list:
    if abstract and isinstance(abstract, str) and re.match('^(?=.*[a-zA-Z])',
abstract):
        # predict the language
        predictions = model.predict(abstract)
        language = predictions[0][0].replace('__label__', '')
        # keep only texts written in English
```

[16] https://github.com/AnaNastase/licenta
[17] https://pandas.pydata.org/

```
        if language == 'en':
            new_abstracts.append(abstract)
```

Firstly, the *lid.176.bin* language detection model is loaded. The *load_model* function requires the path to the fastText model, so I downloaded it from the documentation website[18]. Before predicting the language, the program first makes sure that the abstract is a string object that contains at least one letter of the English alphabet.

The next filtering step consists in the removal of the texts that do not actually contain abstracts, but an enumeration of names of researchers, faculties, universities or conferences. These are detected using the spaCy named entity recognizer, so I loaded the *en_core_web_lg* model, including only the NER pipeline component, like this:

```
nlp = spacy.load('en_core_web_lg', exclude=['tok2vec', 'tagger', 'parser',
'attribute_ruler', 'lemmatizer'])
```

The function then applies the spacy pipeline on each abstract and compares the number of words that are either personal names or organization names to the count of the other words and, if the first one is bigger, the current text is not appended to the final abstract collection.

4.1.3 Extracting Keywords with YAKE

The script proceeds to the keyword extraction with YAKE, calling the *extract_keywords_yake* function, that takes the clean abstract list and concatenates the abstracts into a single string. The spaCy model is loaded the same way as in the code snippet above, then the text is processed. Firstly, the named entities in *remove_entities* are deleted from the document:

```
doc = nlp(text)

remove_entities = ['PERSON', 'NORP', 'FAC', 'GPE', 'LOC', 'DATE', 'TIME',
'PERCENT', 'MONEY', 'QUANTITY', 'CARDINAL', 'ORDINAL']

transformed_text = ' '.join([token.text for token in doc if token.ent_type_ not in
remove_entities])
```

Afterwards, the parameters for the keyword extractor are set, with the values discussed in section 3.2.2, the custom stop words are added to the internal list of the *KeywordExtractor* object, and then the YAKE algorithm is applied in order to extract the 15 keywords:

```
# set parameters for yake keyword extractor
max_ngram = 3
deduplication_threshold = 0.5
keywords_nr = 15
windows_size = 1

kw_extractor = yake.KeywordExtractor(lan="en", n=max_ngram,
dedupLim=deduplication_threshold, top=keywords_nr, windowsSize=windows_size)


# add custom stop words to the default set
kw_extractor.stopword_set.update(set(STOP_WORDS))
```

[18] https://fasttext.cc/docs/en/language-identification.html

```
# extract keywords
kw = kw_extractor.extract_keywords(transformed_text)
```

Below is the list of stop words that I created by observing the words that occurred many times in the abstracts and sometimes appeared among the extracted keywords, but do not bring a meaningful contribution to the keyword list:

```
STOP_WORDS = ['abstract', 'al', 'amount', 'approach', 'article', 'available',
'base', 'based', 'benefit',
'bucharest', 'case', 'category', 'condition', 'conference', 'context', 'copyright',
'datum', 'demonstrate', 'demonstrates', 'demonstrated',
'different', 'difficult', 'et', 'experiment', 'experimental', 'faculty', 'helpful',
'high', 'ieee', 'importance', 'important', 'inconvenience', 'interest',
'interested', 'interests', 'jat', 'jats', 'laboratory',
'main', 'multiple', 'new', 'obtain', 'obtained', 'obtains', 'old', 'order',
'organization', 'paper', 'people', 'policy', 'politehnica', 'polytechnic',
'present', 'presents', 'presented', 'privacy', 'professor', 'propose', 'proposes',
'proposed', 'quality', 'range', 'ranges', 'real',
'recent', 'research', 'researcher', 'result', 'scale', 'show', 'shows', 'showed',
'student', 'study', 'subject', 'studies', 'studied', 'task',
'teacher', 'term', 'text', 'title', 'type', 'unavailable', 'university', 'useful',
'workshop']
```

The function returns a list of the resulting keywords.

4.1.4 Extracting Keywords with LDA

After obtaining the YAKE results, the program calls the *extract_keywords_lda* function. This also begins with the preprocessing of the documents, this time using the whole spaCy pipeline to tag parts of speech, recognize named entities, remove stop words and lemmatize, as follows:

```
# load spacy model
nlp = spacy.load('en_core_web_lg')

# keep only adjectives and nouns
remove_pos = ['ADV', 'PRON', 'PART', 'DET', 'SPACE', 'NUM', 'SYM', 'ADP', 'VERB',
'CCONJ', 'INTJ']
remove_entities = ['PERSON', 'NORP', 'FAC', 'GPE', 'LOC', 'LANGUAGE', 'DATE',
'TIME', 'PERCENT', 'MONEY', 'QUANTITY', 'CARDINAL', 'ORDINAL']

# preprocess documents
tokens = []
for abstract in abstract_list:
    nlp.max_length = len(abstract) + 1000
    doc = nlp(abstract)
    t = [token.lemma_.upper() for token in doc if token.is_alpha and
token.ent_type_ not in remove_entities
         and token.lemma_.lower() not in STOP_WORDS and token.pos_ not in
remove_pos and not token.is_stop]
    tokens.append(t)
```

The code above removes the same named entities as in the case of YAKE, but also removes parts of speech other than nouns and adjectives. I justified all of these decisions in subchapter 3.3.1.

The next preprocessing step is the addition of n-grams to the token collection, using the Phrases model from gensim:

```
# add n-grams to the token list
bigram = Phrases(tokens, min_count=2, delimiter=' ', threshold=1)
tokens = [bigram[text] for text in tokens]
trigram = Phrases(tokens, min_count=2, delimiter=' ', threshold=1)
tokens = [trigram[text] for text in tokens]
```

The model is trained the first time in order to find the most probable two-word collocations, then it is applied to the token list to add the bigrams. The *min_count* parameter specifies how many times two words must co-occur to be considered as an eligible collocation. I set it to 2, so that more n-grams are created, but the chances of joining two words that only co-occur by coincidence are reduced. The *threshold* is another parameter that controls the number of n-grams created, representing a minimum score that a collocation is required to have. I chose a low threshold of 1, because I needed a large number of multi-word tokens in the corpus. The Phrases model is applied one more time on the new tokens collection, to form trigrams and 4-grams.

The last step in preprocessing is the removal of the unigrams, if the word count after removal is at least 100. Otherwise, the corpus would be too small to apply Latent Dirichlet Allocation.

Finally, LdaMulticore is applied, after building the dictionary and the corpus:

```
# create dictionary
dictionary = Dictionary(tokens)
# create corpus
corpus = [dictionary.doc2bow(text) for text in tokens]
# apply LDA
lda_model = LdaMulticore(corpus=corpus, id2word=dictionary, iterations=1,
num_topics=1, workers=3, passes=1)
```

The function returns the top 15 words from the single topic extracted.

4.1.5 The Output

The results are formatted in a tabular structure using pandas DataFrame, with the following columns: the author's ID and name, the keywords extracted using YAKE and the keywords extracted with LDA. The results are then written to a CSV file having the name given as a command line argument to the program.

ID	Name	keywords YAKE	keywords LDA
943	PARVU Corneliu	ATOMIC FORCE MICROSCOPY; SCANNING ELECTRON MICROSCOPY; STRUCTURE; FUNCTIONAL; DESIGN; PRODUCT; MEMBRANE;	
698	RINDASU OVIDIU VIOREL	MITTAL STEEL GALATI; POST DEFORMATION MAINTAINING; THERMO I STEEL; DEFORMATION; MECHANICAL CHARACTERISTIC; PAR	
854	SINDILA Gheorghe	FINITE ELEMENT ANALYSIS; REDUCING SHEARING FORCES; CALCULA FORCE; HUMAN; MATERIAL; PROCESS; DEFORMATION; LEVE	
549	TIRIPLICA Petre Gheorghe	MONITORING PROJECT BUDGETS; PROJECT BUDGET MONITORING; F METHODOLOGY; PRODUCT; COMPONENT; PROJECT; SYSTEI	
916	Funar STEFAN PETRU	OHS MANAGEMENT SYSTEM; EUROPEAN CONFORMITY ASSESSMENT SYSTEM; RISK; PROCESS; PROJECT; FLOW; MANAGEMENT SY	
802	DANCIU TIBERIU-DINU	CHEMICAL ENGINEERING EDUCATION; CHEMISTRY TEACHING STAFF CHEMICAL ENGINEERING; E LEARNING; DEEP OXIDATION; IN	
736	TROFIN Roxana	DIGITAL TECHNOLOGIES INTEGRATION; TEACH SPECIFIC COMPETEN COMPETENCE; SPECIFIC; TRANSLATION; TECHNOLOGY; LAN	
367	TARCEA Claudia Ionela	SCANNING ELECTRON MICROSCOPY; ATOMIC FORCE MICROSCOPY; HEAVY METAL; X RAY DIFFRACTION XRD; IRON OXIDE NANOF	
69156	GUTU-ROBU MARIUS GABRIEL	NATURAL LANGUAGE PROCESSING; SUPPORTED COLLABORATIVE LE COMPUTER SUPPORTED COLLABORATIVE LEARNING; CHAT,	

Figure 13: Output file

Figure 13 shows an example of the results of my keyword extraction application, saved in a CSV file.

4.2 Choosing a spaCy Model

The selection of a spaCy model was challenging with regard to the accuracy of the named entity recognizer. As I previously discussed in section 3.2.1, the largest standard spaCy model, *en_core_web_lg*, has difficulties in recognizing named entities, as it mistakenly labels some technical terms written with initial uppercase letters as organization names. This was problematic, because I wanted to remove organizations during text preprocessing, in order to eliminate entities such as universities, faculties or conferences. Due to this, a lot of key terms were removed from the text.

Improved performance in named entity recognition can be achieved by using a transformer model, like *en_core_web_*trf. The transformers implementation from spaCy is essentially a wrapper for the Hugging Face transformer library[19], allowing the use of the most powerful NLP models, such as BERT and GPT, through the spaCy pipeline[20]. The Transformer architecture is a deep neural network structure, created by Google researchers in 2017 (Vaswani et al., 2023), that contains attention layers, mechanisms that give the model the capacity to assume which words are more important in a specific context.

I tried using the *en_core_web_*trf spaCy transformer pipeline and I compared it to the *en_core_web_lg* in terms of the accuracy of NER and the running time. For the same author, with 428 publications, *en_core_web_lg* incorrectly labelled the term “Big Data” as organization 42 times, while the *en_core_web_*trf recognized it as “ORG” only 1 time in all the abstract collection. However, the preprocessing with the transformer model had a duration of 90.05 seconds, more than 31 times larger compared to the processing time of the standard model, which lasted approximately 2.29 seconds. Similarly, for the author with 1009 papers, the most on the CRESCDI platform, *en_core_web_lg* tagged the term “Earth Observation” or its acronym “EO” as organization 259 times, taking 4.86 seconds to run, drastically less than the 253.31 seconds that the *en_core_web_*trf model took. On the other hand, the transformer only considered “Earth Observation” an organization 4 times. Even for a researcher with 10 publications, the preprocessing using the transformer model ran for 4.11 seconds, about 37 times slower than the word-vector model which processed the text in 0.11 seconds.

Given the significant overhead introduced by using the transformer model in preprocessing the abstract list, I decided to utilize the *en_core_web_lg* model and not remove the “ORG” entities, to avoid the loss of some significant terms incorrectly labelled. The alternative approach that I found was to add some words that denominate institutions to the stop word list, such as “university”, “polytechnic” or “IEEE”.

[19] https://huggingface.co/docs/transformers/index
[20] https://explosion.ai/blog/spacy-transformers

5 EVALUATION OF RESULTS

In this chapter, I will estimate the performance of my solution by computing two metrics commonly used in Information Retrieval evaluation: precision and recall, and I will compare the results of these metrics to those found in other related work. I will also assess the two methods of keyword extraction employed in my project individually and draw a parallel between the two of them regarding the previous metrics, and the quality of keyword extraction depending on the size of the document collections.

Table 11: Keyword extraction results for some top 20 authors with the most publications

Name	YAKE	LDA
DOBRE Ciprian Mihai	DISTRIBUTED SYSTEMS; MOBILE CLOUD COMPUTING; DATA; CLOUD COMPUTING SYSTEMS; MOBILE BIG DATA; OPPORTUNISTIC NETWORKS; MOBILE DEVICES; NETWORK MANAGEMENT SERVICES; BIG DATA PROCESSING; MOBILE SOCIAL NETWORKS; NETWORK; SYSTEMS; SERVICES; MOBILE OPPORTUNISTIC CLOUD; MOBILE	MOBILE DEVICE; OPPORTUNISTIC NETWORK; LARGE SYSTEM; CLOUD COMPUTING; FAULT TOLERANCE; BIG DATA; SINGLE SALE ACCOUNT MANAGEMENT; INFORMATION PRODUCT SERVICE NOTE; END USER; SCHEDULING ALGORITHM; SITE AGREEMENT; THANK REVIEWER PROFIT WORLD; LARGE TECHNICAL PROFESSIONAL TECHNOLOGY; HUMANITY RIGHT USE WEB; CROSS SECTION
POP Florin	CLOUD COMPUTING; BIG DATA PROCESSING; BIG DATA; CLOUD SYSTEMS; DATA; CLOUD; SERVICE LEVEL AGREEMENT; CLOUD SERVICE PROVIDERS; DATA PROCESSING SYSTEMS; DISTRIBUTED SYSTEMS RESOURCES; GRID SCHEDULING ALGORITHMS; COMPUTING DISTRIBUTED SYSTEMS; IDENTIFICATION SYSTEM DATA; SYSTEM; BIG DATA ENVIRONMENTS	SCHEDULING ALGORITHM; CLOUD COMPUTING; BIG DATA; RESOURCE MANAGEMENT; SMART CITY; CLOUD SYSTEM; LARGE SYSTEM; TIME SERIES; GENETIC ALGORITHM; WEB SERVICE; SMART ENVIRONMENT; COMPLEX SYSTEM; FAULT TOLERANCE; SCHEDULING MODEL; CLOUD SERVICE
TRAUSAN-MATU Stefan	NATURAL LANGUAGE PROCESSING; SUPPORTED COLLABORATIVE LEARNING; COMPUTER SUPPORTED COLLABORATIVE; LANGUAGE PROCESSING TECHNIQUES; LATENT SEMANTIC ANALYSIS; COHESION NETWORK ANALYSIS; ADVANCED NATURAL LANGUAGE; LEARNING; ANALYSIS; LANGUAGE; LEARNING CHAT CONVERSATIONS; COLLABORATIVE LEARNING TOOLS; LEARNING ENVIRONMENT; MACHINE LEARNING SYSTEM; LEARNING MANAGEMENT SYSTEMS	COMPUTER SUPPORTED COLLABORATIVE LEARNING; E LEARNING; NATURAL LANGUAGE PROCESSING; CHAT CONVERSATION; SOCIAL NETWORK; NATURAL LANGUAGE; READERBENCH FRAMEWORK; LEARNING ENVIRONMENT; ARTIFICIAL INTELLIGENCE; NATURAL LANGUAGE PROCESSING TECHNIQUE; LEARNING PROCESS; WEB PAGE; NATURAL LANGUAGE PROCESSING NLP; POLYPHONIC MODEL; TEXTUAL COMPLEXITY INDEX

Table 11 shows some results obtained for three authors with some of the biggest numbers of publications among UNSTPB researchers. The relevance of these keywords can be assumed by correlating them with the authors' publication titles. For example, by looking at some of Ciprian Dobre's titles: "Intelligent services for Big Data science", "Big Data and Cloud Computing: A Survey of the State-of-the-Art and Research Challenges", "Social Aspects to Support Opportunistic Networks in an Academic Environment", "A Simulation Framework for Dependable Distributed Systems", there is evidence to say that "Big Data", "Cloud Computing", "Opportunistic Networks" and "Distributed Systems" are indeed keywords of his research papers.

A first observation that can be made regarding the comparison between YAKE and LDA is the generality of the keywords extracted using LDA, as opposed to the more specific terms produced by YAKE. For instance, from Ciprian Dobre's abstracts, LDA extracted "Big Data",

and YAKE extracted “Mobile Big Data” and “Big Data processing”, because YAKE rewards longer keyphrases with better scores, while LDA favors the most frequent ones.

5.1 Evaluation Metrics

The difficulty in quantifying the effectiveness and reliability of Information Retrieval (IR) systems is generally acknowledged by researchers in the field (Hripcsak & Rothschild, 2005; Maier & Simmen, 2001; Nguyen & Kan, 2007). The challenge of evaluating keyword extractors using exact measures stems from the subjective character of keywords (Nguyen & Kan, 2007), the necessity of having reliable gold standards selected by humans, the issue of the exact match (Campos et al., 2020), which refers to the fact that different forms of the same concept are not considered identical keywords when automatically calculating metrics, and the variety of keywords that can be extracted from a text, particularly a large one.

The most used evaluation metrics for keyword extraction are precision and recall, two traditional machine learning measures that can be particularized for IR or keyword extraction applications.

$$precision = \frac{number\ of\ relevant\ keywords\ retrieved}{number\ of\ keywords\ retrieved} \tag{6}$$

Equation (6), taken from Nguyen & Kan (2007) and adapted for the context of keyword extraction, is the formula for the precision, defined as the proportion of correctly identified keywords out of the total extracted keywords. It is important to note that the precision only takes into account the “positives”, as the relevant keywords retrieved are equivalent to the “true positives”, while the total keywords retrieved is the sum of true and false positives, terms generally encountered when evaluating binary classifiers.

$$recall = \frac{number\ of\ relevant\ keywords\ retrieved}{number\ of\ gold\ keywords} \tag{7}$$

Similarly, the recall, calculated as in formula (7), represents the fraction of correctly identified keywords from the number of gold-standard keywords, so this time the denominator differs in that it also takes into consideration the “negatives”, being the sum of true positives and false negatives. According to Hripcsak & Rothschild (2005), the number of “negative cases” is challenging to assess in Information Retrieval, as the number of false negatives, used in the recall formula, is influenced by the reliability of the gold keywords, while the number of true negatives is practically impossible to calculate.

For the evaluation, I used my implementation to extract keywords from the abstracts of two sets of authors: one set consisting of the top 20 researchers with the most publications on the CRESCDI platform, and another set that includes 9 authors with fewer papers, ranging from 3 to 36. For the gold standard, I used the keywords provided by the authors for each publication. This resulted in a very large amount of gold keywords, with hundreds and even thousands of distinct keywords for every researcher in the top 20, so I sorted them by the number of occurrences and only considered a fraction.

For evaluating the overall quality of my keyword extractor, with the combined results from YAKE and LDA, I used the most frequent 60 gold keywords for each author and obtained 144 extracted keywords that are also among the gold standard words, 426 false positives and 1056 false negatives. These numbers lead to a precision of 0.25 and a recall of 0.12, values that are comparable to other results in related work. Kumbhar et al. (2019) compared the results of six state of the art keyword extraction methods on four different types of datasets. Looking at the precisions and recalls obtained on scientific questions and answers from Stack Exchange, the precisions of the six extractors range from 0.11 to 0.27, only the top value being higher than the precision that I obtained. The recalls are between 0.1 to 0.25, the latter being equal to the recall of my keyword extractor. Similarly, Liu et al. (2020) conducted a comparison of five keyword extractors on comments from student forums. Their results have a very wide range for precision, between 0.11 and 0.85, and recall, between 0.04 and 0.83.

Individually, YAKE provides a good precision on the first 60 gold keywords, with a value equal to 0.39, while the recall is expected to be low, 0.098, since the number of corresponding keywords is divided by 60, the count of the reference keywords. LDA has lower values, 0.13 for precision and 0.03 for recall.

For the set of authors with fewer papers, the recall remained the same, but the precision decreased to 0.16. However, these values are not very suggestive of the quality of the extracted keywords, given there are only 9 authors in this set and some of them have only 3 or 5 publications, therefore the number of gold keywords is also reduced.

To conclude, the precision of 0.25 achieved on the abstracts of the top 20 authors is a good result, and the 0.39 of YAKE even more so, taking into account that in this context the reference data is not really an exact list of the most relevant keywords, so it is not a true gold standard. I chose to focus more on achieving a higher precision, rather than a higher recall, since I believe it is more important for my keyword extractor application to capture as many correct keywords as possible, that provide a good overview of one researcher's domain of activity. Additionally, the focus on the positives, the extracted keywords that match the reference, is supported by the common agreement that the negative cases are difficult to decide objectively (Hripcsak & Rothschild, 2005; Nguyen & Kan, 2007), as manually assigned keywords are not the only valid options, given the subjective nature of keyword extraction and the versatility of natural language. This being said, I expected the lower values for the recall, firstly because I chose a big number of gold keywords so that the number of true positives is maximized, and secondly because of the high chance that the false negatives are not truly negatives, but they were not included in the gold standard.

For a comprehensive evaluation of the quality of keyword extraction, the human perspective is maybe more important than the numerical measurements. Going forward, I will analyze the keywords extracted by my program for some authors in comparison with the keywords displayed in the CRESCDI profiles of those researchers, if they appear on a blue background,

which means they were validated by the author, or in comparison with the keywords displayed in the researchers' descriptions from Google Academic.

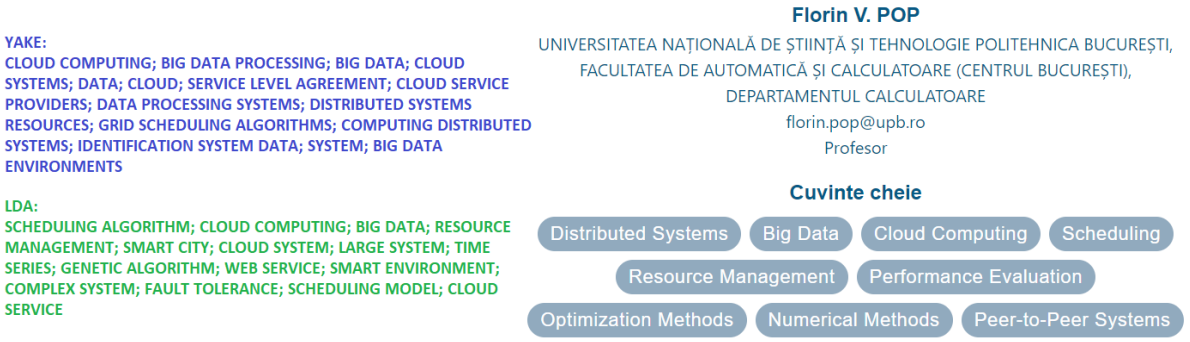

Figure 14: Comparison between keywords

In figure 14, there are the keywords extracted using my implementation on the left and the keywords displayed on CRESCDI on the right[21]. Out of the 9 keywords validated by the author, my program extracted 3 exact matches ("Big Data", "Cloud Computing" and "Resource management"), whereas 2 other concepts are present in different forms. My results do not include "Distributed Systems" exactly, but the term is included in "Distributed Systems Resources" and in "Computing Distributed Systems". Likewise, "Scheduling" is not present by itself among the extracted keywords, but it appears in "Scheduling Algorithm" and "Grid Scheduling Algorithms".

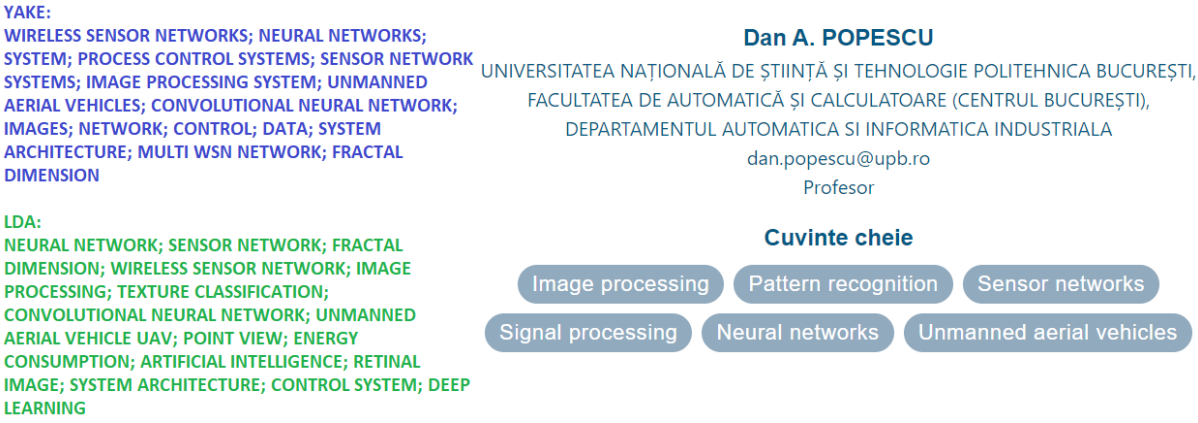

Figure 15: Comparison between keywords

Only 2 out of the 6 manually validated keywords[22] in figure 15 were not retrieved by my implementation ("Pattern recognition" and "Signal processing").

[21] https://crescdi.pub.ro/#/profile/562
[22] https://crescdi.pub.ro/#/profile/872

YAKE:
NATURAL LANGUAGE PROCESSING; SUPPORTED COLLABORATIVE LEARNING; ADVANCED NATURAL LANGUAGE; COHESION NETWORK ANALYSIS; LANGUAGE PROCESSING TECHNIQUES; COMPUTER SUPPORTED COLLABORATIVE; COLLABORATIVE LEARNING TOOLS; RAGE ECOSYSTEM PORTAL; LEARNING MANAGEMENT SYSTEMS; PROCESSING TOOL READER; PROCESSING SEMANTIC MODELS; CSCL CHAT CONVERSATIONS; GAME LEARNING ENVIRONMENT; KNOWLEDGE PROCESSING SYSTEMS; SEMANTIC ANNOTATION COMPONENT

LDA:
COMPUTER SUPPORTED COLLABORATIVE LEARNING; CHAT CONVERSATION; NATURAL LANGUAGE PROCESSING NLP; NATURAL LANGUAGE; NEURAL NETWORK; COHESION NETWORK ANALYSIS; READERBENCH FRAMEWORK; RAGE ECOSYSTEM; NATURAL LANGUAGE PROCESSING; IMPLICIT LINK; MULTI PARTICIPANT; ADVANCED NATURAL LANGUAGE PROCESSING; SEMANTIC SIMILARITY; FILTERING CRITERION; SEMANTIC ANNOTATION

Gabriel Gutu-Robu
University Politehnica of Bucharest
Adresă de e-mail confirmată pe upb.ro
Natural Language Processing Computer-Supported Colla…
Recommender Systems Discourse Analysis Human-Computer

Figure 16: Comparison between keywords

I took the keywords on the right side of figure 16 from Google Academic[23]. The first 2 keywords, "Natural Language Processing" and "Computer-Supported Collaborative Learning", are also found among the keyphrases extracted using my implementation.

5.2 Comparison between YAKE and LDA

In terms of metrics, on the set of top 20 researchers YAKE had a precision of 0.39 and a recall of 0.0975, while LDA demonstrated a precision equal to 0.127 and a recall equal to 0.032, so the former achieved a better performance. On the other hand, for the other 9 authors with a smaller number of papers, LDA had improved values for precision and recall, 0.259 and 0.096 respectively, and YAKE had lower values, 0.081 and 0.03. This may seem unexpected, since LDA is known for operating on very large document collections, whereas YAKE presents itself as an algorithm that performs well independent of the size of the input text. However, this happens because the list of gold keywords contains numerous single-word terms, which are better captured by LDA on smaller document collections, where there are almost no collocations generated. YAKE attempts to extract n-grams up to three words and sometimes they are not relevant when working with a smaller corpus.

[23] https://scholar.google.com/citations?hl=ro&user=-pGqq7QAAAAJ

Table 12: Comparison YAKE vs LDA using keywords extracted from a collection of 3 abstracts

YAKE	LDA
ATOMIC FORCE MICROSCOPY	STRUCTURE
SCANNING ELECTRON MICROSCOPY	FUNCTIONAL
POSS COMPOUND DIRECTLY	DESIGN
CARBON NANOTUBES COMPOSITE	PRODUCT
AMINO CARBON NANOTUBES	MEMBRANE
NANOTUBES COMPOSITE MEMBRANES	CARBON NANOTUBES
UDMA MATRIX LEADS	HYBRID
FUNCTIONALIZED MULTIWALLED CARBON	POSS
HYBRID MATERIAL TRANSPARENCY	TAXONOMY
ENGINEERING DESIGN ACTIVITY	CONVERSION
DEVELOPING FUNCTIONAL TAXONOMIES	SYSTEM
SUPPORTING ENGINEERING DESIGN	CHEMICAL
COMPOUND DIRECTLY INFLUENCES	SPECTROSCOPY
ELECTRONIC CONDUCTIVE CHEMICAL	MICROSCOPY
CONDUCTIVE CHEMICAL SPECIES	DSC

In table 12, there are the keywords that I extracted with YAKE, in the left column, and with LDA, on the right, from the abstracts of a researcher who has 3 papers in the CRESCDI database. The ones extracted with YAKE provide more details about the content of this author's publications, containing only trigrams, but there are also meaningless keywords among them, from a grammatical point of view, such as "compound directly influences", which is an incomplete phrase. LDA, on the other hand, produced only 1 bigram and 14 unigrams. Together, the LDA keywords give a general idea about the researcher's activity, but individually most of them do not seem relevant, for instance "system" or "product", which are very general and ambiguous concepts.

Table 13: Comparison YAKE vs LDA using keywords extracted from 646 abstracts

YAKE	LDA
NATURAL LANGUAGE PROCESSING	COMPUTER SUPPORTED COLLABORATIVE LEARNING
SUPPORTED COLLABORATIVE LEARNING	E LEARNING
COMPUTER SUPPORTED COLLABORATIVE	NATURAL LANGUAGE PROCESSING
LANGUAGE PROCESSING TECHNIQUES	CHAT CONVERSATION
LATENT SEMANTIC ANALYSIS	SOCIAL NETWORK
COHESION NETWORK ANALYSIS	NATURAL LANGUAGE
ADVANCED NATURAL LANGUAGE	READERBENCH FRAMEWORK
LEARNING	LEARNING ENVIRONMENT
ANALYSIS	ARTIFICIAL INTELLIGENCE
LANGUAGE	NATURAL LANGUAGE PROCESSING TECHNIQUE
LEARNING CHAT CONVERSATIONS	LEARNING PROCESS
COLLABORATIVE LEARNING TOOLS	WEB PAGE
LEARNING ENVIRONMENT	NATURAL LANGUAGE PROCESSING NLP
MACHINE LEARNING SYSTEM	POLYPHONIC MODEL
LEARNING MANAGEMENT SYSTEMS	TEXTUAL COMPLEXITY INDEX

Table 13 depicts another parallel between the keywords extracted using YAKE and LDA, from the abstracts of an author in the top 20. One limitation of YAKE observed in this case is that it extracts some keyphrases that are only partially correct, due to the chosen maximum length of n-grams being 3, hence it extracts “Supported Collaborative Learning” and “Computer Supported Collaborative”, while LDA extracts the whole term “Computer Supported Collaborative Learning”. Although the two sets of keywords are similar and convey the same general idea about the author’s papers, there are no exact matches apart from “Natural Language Processing” and “learning environment”, so this is a good example for illustrating how these two algorithms complete each other in finding meaningful keywords.

To conclude the comparison between the two methods, concerning their performance on different sizes of document collections, YAKE is definitely better for authors with fewer publications, as it captures longer and more significant terms, while LDA might be more suitable for larger corpuses, depending on how specific we expect the results to be, since it generally extracts more concise keywords (LDA is more likely to extract a term like “Cloud Computing”, as opposed to YAKE’s preference for longer phrases like “Cloud Computing systems”).

6 CONCLUSION AND FUTURE WORK

In this document, I amply described the process of extracting the most meaningful words from researchers' abstracts to accurately illustrate the topics that they focus on in their research. To achieve this, I used two distinct methods, YAKE and LDA. Both approaches produced relevant results that individually captured the core themes of the publications, and combining the resulting keywords from both methods provided an even more comprehensive picture. The results were evaluated using rigorous metrics, but the best indication of their accuracy is the consistency of the keywords when compared to the author's publication titles and the keywords displayed in their Google Scholar or CRESCDI profile, if they have manually validated their keywords on the platform.

Besides the extraction of meaningful terms, another objective of this document consisted in the comparison between the two methods integrated in my implementation. While both of them were effective in the extraction of keywords, YAKE was significantly better at extracting more descriptive phrases from smaller texts, whereas LDA might be preferred for larger document collections, if the resulting keywords are expected to be more generic.

In conclusion, the keywords extracted using my application are relevant and capable of providing an overview of a researcher's field of activity on the CRESCDI platform. Improvements that can be made on this work include evaluating the results for a larger number of authors, refining the stop word list, which might be automatically generated instead of hard-coded, and implementing a filtering mechanism in order to further clean the resulting keyword list of an author and remove some less meaningful terms or combine multiple keywords into a single one.

Some related features that can be integrated into the CRESCDI platform are: the possibility of searching researchers based on keywords, a recommender system that gives authors suggestions about potential collaborators, and the creation of some graphical representations, including visualizations of research trends and maps that illustrate clusters of authors with similar interests.

7 REFERENCES

Abdelrazek, A., Eid, Y., Gawish, E., Medhat, W., & Hassan, A. (2023). Topic modeling algorithms and applications: A survey. *Information Systems*, *112*, 102131. https://doi.org/https://doi.org/10.1016/j.is.2022.102131

Arora, M., Kanjilal, U., & Varshney, D. (2016). Evaluation of information retrieval: precision and recall. *International Journal of Indian Culture and Business Management*, *12*, 224. https://doi.org/10.1504/IJICBM.2016.074482

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*(Jan), 993–1022.

Campos, R., Mangaravite, V., Pasquali, A., Jorge, A. M., Nunes, C., & Jatowt, A. (2018). YAKE! Collection-Independent Automatic Keyword Extractor. *Information Retrieval Technology*, 806–810. https://doi.org/10.1007/978-3-319-76941-7_80

Campos, R., Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C., & Jatowt, A. (2020). YAKE! Keyword extraction from single documents using multiple local features. *Information Sciences*, *509*, 257–289. https://doi.org/10.1016/J.INS.2019.09.013

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407.

Griffiths, T. L., & Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, *101*(suppl_1), 5228–5235. https://doi.org/10.1073/pnas.0307752101

Hripcsak, G., & Rothschild, A. (2005). Agreement, the F-Measure, and Reliability in Information Retrieval. *Journal of the American Medical Informatics Association : JAMIA*, *12*, 296–298. https://doi.org/10.1197/jamia.M1733

Jelodar, H., Wang, Y., Yuan, C., & Feng, X. (2017). *Latent Dirichlet Allocation (LDA) and Topic modeling: models, applications, a survey*.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). FastText.zip: Compressing text classification models. *ArXiv Preprint ArXiv:1612.03651*.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *ArXiv Preprint ArXiv:1607.01759*.

Justeson, J., & Katz, S. (1995). Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text. *Natural Language Engineering*, *1*, 9. https://doi.org/10.1017/S1351324900000048

Kumbhar, A., Savargaonkar, M., Nalwaya, A., Bian, C., & Abouelenien, M. (2019). *Keyword Extraction Performance Analysis*. 550–553. https://doi.org/10.1109/MIPR.2019.00111

Liu, F., Huang, X., Huang, W., & Duan, S. (2020). Performance Evaluation of Keyword Extraction Methods and Visualization for Student Online Comments. *Symmetry*, *12*, 1923. https://doi.org/10.3390/sym12111923

Maier, A., & Simmen, D. (2001). DB2 Optimization in Support of Full Text Search. *IEEE Data Eng. Bull.*, *24*, 3–6.

Matthew Honnibal. (n.d.). *Dead Code Should Be Buried*. Retrieved June 9, 2024, from https://explosion.ai/blog/dead-code-should-be-buried

Moreno-Ortiz, A. (2024). *Keywords* (pp. 59–102). https://doi.org/10.1007/978-3-031-52719-7_4

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464

Nguyen, T., & Kan, M.-Y. (2007). *Keyphrase Extraction in Scientific Publications*. 317–326. https://doi.org/10.1007/978-3-540-77094-7_41

Nomoto, T. (2022). Keyword Extraction: A Modern Perspective. *SN Computer Science*, *4*. https://doi.org/10.1007/s42979-022-01481-7

Řeh uřek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New for NLP Frameworks*, 45–50.

Röder, M., Both, A., & Hinneburg, A. (2015). Exploring the Space of Topic Coherence Measures. *WSDM 2015 - Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, 399–408. https://doi.org/10.1145/2684822.2685324

Rose, S., Engel, D., Cramer, N., & Cowley, W. (2010). Automatic Keyword Extraction from Individual Documents. In *Text Mining: Applications and Theory* (pp. 1–20). https://doi.org/10.1002/9780470689646.ch1

spaCy. (n.d.). Retrieved June 9, 2024, from https://spacy.io/

Srinivasa-Desikan, B. (2018). *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd.

Tijare, P., & Rani, P. (2020). Exploring popular topic models. *Journal of Physics: Conference Series*, *1706*, 12171. https://doi.org/10.1088/1742-6596/1706/1/012171

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need*.

Zhang, C. (2008). Automatic keyword extraction from documents using conditional random fields. *Journal of Computational Information Systems*, *4*(3), 1169–1180.