UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI TEHNOLOGIE POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Titlul proiectului de diplomă (ex: Șablon proiect de diplomă)

Subtitlu (ex: versiunea 2018)

Ioana Popescu

**Coordonator științific:**

Prof. dr. ing. Andrei Ionescu

BUCUREŞTI

2018

UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

COMPUTER SCIENCE DEPARTMENT

|  |  |
| --- | --- |
| upb | cs |

DIPLOMA PROJECT

Diploma Project Title (eg: Diploma project template)

Subtitle (eg: 2018 version)

Ioana Popescu

**Thesis advisor:**

Prof. dr. ing. Andrei Ionescu

BUCHAREST

2018

**CUPRINS**

[Sinopsis 2](#_Toc168927500)

[Abstract 2](#_Toc168927501)

[Mulțumiri 3](#_Toc168927502)

[1 Introducere 4](#_Toc168927503)

[1.1 Context 4](#_Toc168927504)

[1.2 Problema 4](#_Toc168927505)

[1.3 Obiective 4](#_Toc168927506)

[1.4 Structura lucrării 4](#_Toc168927507)

[2 Analiza și specificarea cerințelor 5](#_Toc168927508)

[3 STATE OF THE ART 6](#_Toc168927509)

[3.1 YAKE! (Yet Another Keyword Extractor) 6](#_Toc168927510)

[3.2 Latent Dirichlet Allocation 9](#_Toc168927511)

[3.3 spaCy 12](#_Toc168927512)

[3.3.1 The spaCy pipeline 12](#_Toc168927513)

[4 Soluția propusă 13](#_Toc168927514)

[4.1 Indicații formatare formule 14](#_Toc168927515)

[5 Detalii de implementare 15](#_Toc168927516)

[5.1 Indicații formatare tabele 15](#_Toc168927517)

[6 Studiu de caz / Evaluarea rezultatelor 17](#_Toc168927518)

[7 Concluzii 19](#_Toc168927519)

[8 REFERENCES 20](#_Toc168927520)

[9 Anexe 21](#_Toc168927521)

# Sinopsis

Sinopsisul proiectului are rol de introducere, conținând atât o descriere pe scurt a problemei abordate cât și o enumerare sumară a rezultatelor și a concluziilor. Se recomandă ca sinopsisul să fie redactat într-un limbaj accesibil unei persoane nefamiliarizate cu domeniul, dar în același timp destul de specific pentru a oferi rapid o vedere de ansamblu asupra proiectului prezentat.

Sinopsisul proiectului va fi redactat atât în română cât și în engleză. Ca dimensiunea recomandată aceasta secțiune va avea maxim 200 de cuvinte pentru fiecare variantă. Împreună, ambele variante se vor încadra într-o singură pagină.

# Abstract

The abstract has an introductory role and should engulf both a brief description of the issue at hand, as well as an overview of the obtained results and conclusions. The abstract should be formulated such that even somebody that is unfamiliar with the projects’ domain can grasp the objectives of the thesis while, at the same time, retaining a specificity level offering a bird’s eye view of the project.

The projects’ abstract will be elaborated in both Romanian and English. The recommended size for this section is limited to 200 words for each version. Together, both versions will fit in one page.

# Mulțumiri

(opțional) Aici puteți introduce o secțiunea specială de mulțumiri / acknowledgments.

# Introducere

Parametrii de formatare recomandați pentru lucrare:

* Font recomandat: Calibri; Dimensiune font: 12;
* Spațiere între linii: 1,15; Spațiere după paragraf: 8pt;
* Stil: Justified;
* Dimensiune pagină: A4; Margini: 2,54cm/ 2,54cm/ 2,54cm/ 2,54cm;
* Heading1: Calibri, 14, bold, all caps;
* Heading2: Calibri, 14, bold;
* Heading3: Calibri, 12.
* Font pentru formule: Cambria Math, 12.

În cadrul introducerii, este necesară abordarea următoarelor puncte care reprezintă de fapt familiarizarea cititorului (comisia, alți colegi sau experți în domeniu) cu tema proiectului, soluția propusa și cuprinsul/structura lucrării. Deși introducerea poate conține și unele elemente mai generale, se recomandă păstrarea unui limbaj tehnic, specific audienței care va citi lucrarea.

În cadrul capitolelor următoare, veți regăsi o serie notații de forma [Dezvoltare de produs], [Cercetare]. Acest tip de formatare este utilizat exclusiv în acest template pentru a marca sfaturi și cerințe specifice pentru lucrări de diploma cu specific diferit. În pregătirea documentului vostru, nu veți utiliza aceste marcaje.

Elementele pe care trebuie să le abordați în introducere sunt descrise în cadrul subcapitolelor de mai jos.

## Context

O scurtă introducere a proiectului, motivație, explicație de ce este relevant domeniul proiectului.

## Problema

Care este problema pe care proiectul o va rezolva.

## Obiective

Care sunt obiectivele proiectului/soluției/abordării/ideii; Ce creșteri sau evoluții determină rezolvarea proiectului.

## Structura lucrării

Un paragraf în care fiecare dintre secțiunile următoare este prezentată în 1-2 fraze, punând accentul pe elementele cele mai semnificative din fiecare secțiune.

# Analiza și specificarea cerințelor

[Dezvoltare de produs] Acest capitol va analiza cerințele produsului din prisma potențialilor clienți și a scenariilor de utilizare preconizate, urmând a fi generată o lista de funcționalități.

[Cercetare] Acest capitol va introduce motivația realizării proiectului propus.

Dacă proiectul de licență face parte dintr-un proiect mai amplu (de exemplu un proiect complex, la care lucrează 2 studenți (ex: 1 student la front-end-ul aplicației, 1 student la back-end-ul aplicației), în acest capitol va fi explicat pe scurt ansamblul proiectului și ce parte din proiect este adresată de lucrarea propusă.

Criterii pentru calificativul Nesatisfăcător:

* [Dezvoltare de produs] Cerințele sunt imaginate de student pe baza unei analize a pieței;
* [Cercetare] Nu se oferă o motivație valida.

Criterii pentru calificativul *Satisfăcător*:

* [Dezvoltare de produs] Există un interviu, un client, analiza cerințelor este elaborată pe baza interviului;
* [Cercetare] Motivația este doar personala.

Criterii pentru calificativul *Bine*:

* [Dezvoltare de produs] Proces iterativ pe baza unor interviuri cu mai mulți clienți, dezvoltare MVP, reevaluare cerințe;
* [Cercetare] Motivația este legata de o necesitate științifica / tehnica explicită.

# STATE OF THE ART

In the field of Natural Language Processing (NLP), keyword extraction is essential for text mining purposes, such as automatic indexing, automatic summarization, automatic classification, automatic clustering, 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, this area is still not as precise as other computer science fields (Campos et al., 2018). There is no universal approach for effectively extracting semantic information without requiring either large manually labelled data sets for training or a comprehensive preprocessing of the text, according to the specifics of the domain and of the individual document. Therefore, one needs to use more than one tool in order to achieve the expected results. In this chapter, I will discuss the tools integrated in my project: YAKE! (Yet Another Keyword Extractor) for keyword extraction, LDA (Latent Dirichlet Allocation) for topic modelling, the spaCy Python library for preprocessing, part of speech tagging and named entity recognition, fastText for language detection and the Gensim package for the implementation of LDA.

## YAKE! (Yet Another Keyword Extractor)

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

Despite being an unsupervised method, TF-IDF (Term Frequency-Inverse Document Frequency) also 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 but is otherwise rare across the collection. The formula used to measure the significance of a word is (Nomoto, 2022):

(1)

The first term in equation (1) is the Term Frequency (frequency of word in document j), while the second term is known as the Inverse Document Frequency, where the numerator represents the total count of documents and the denominator is the number of documents that contain word (Nomoto, 2022).

RAKE (Rapid Automatic Keyword Extraction) is another 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):

(2)

In formula (2), 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 words 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).

O imagine care conține text, captură de ecran, Font, document

Descriere generată automat

Figure 1: YAKE Algorithm (Campos et al., 2020)

Figure 1 presents YAKE as pseudocode, highlighting the five steps of the algorithm (Campos et al., 2020):

### 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).

### Feature extraction

The algorithm favours 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:

(3)

In formula (3), evaluates the casing dimension, taking into account the frequency of occurrence with an initial capital letter and the frequency of the acronym. The 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” () 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” (), 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 (“term different sentence”), which rewards tokens that appear in numerous sentences.

### Computing term score

The score of each token is determined using formula (3), the most significant terms having the lowest score.

### 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:

(4)

In equation (4), 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.

### 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.

## Latent Dirichlet Allocation

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 is sampled from distribution θ and a word 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:

O imagine care conține text, Font, alb, chitanță

Descriere generată automat

Figure 2: An abstract with the words labelled according to the topic they belong in

In figure 2, 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.

O imagine care conține linie, Interval, diagramă, captură de ecran

Descriere generată automat

Figure 3: Coherence score for scikit-learn LDA (Tijare & Rani, 2020)

Figure 3 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.

O imagine care conține captură de ecran, cerc, diagramă, proiectare

Descriere generată automat

Figure 4: pyLDAvis representation of LDA results obtained with Gensim

In figure 4, 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 and for LSI. LDA has the higher score, thus the higher coherence.

## spaCy

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.):

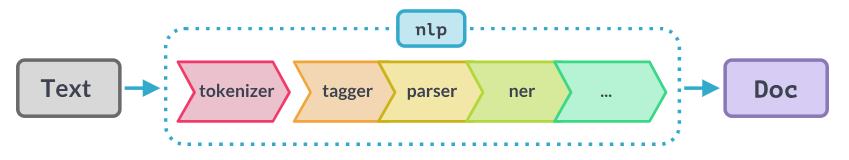


Figure 5: spaCy pipeline

Figure 5 shows the default components of the spaCy pipeline, which can be modified by disabling some components or by adding custom components, developed by the user or available 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 the older libraries, which he believes offer too many features that are otherwise nor effective nor up to date with the advancements in computer science (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.).

O imagine care conține text, captură de ecran, diagramă, linie

Descriere generată automat

Figure 6: en\_core\_web\_lg pipeline (*SpaCy*, n.d.)

The components of the common spaCy models are pictured in figure 6, along with their dependencies.

### 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.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figure 7: spaCy tokenizer

Figure 7, 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.

### 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 6 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.

### 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 7.

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().

### 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.

O imagine care conține schiță, diagramă

Descriere generată automat

Figure 8: displaCy dependency visualizer

The following dependencies are pictured in figure 8, which I made using displaCy visualizer[[1]](#footnote-1):

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

### 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.

### 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 analysed sentence “Let’s go to N.Y.!” is “us”, while the other lemmas coincide with the initial words.

### 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.).

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figure 9: displaCy entity visualizer [[2]](#footnote-2)

Figure 9 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.

## fastText

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

(5)

In formula (5), N is the number of documents, is a vector which indicates the correct label of document n ( if document n belongs in class k, otherwise ), is the normalized vector of n-gram features and f is the softmax function described above, thus 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 (5) (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).

I used the Python fastText library for the language identification purpose[[3]](#footnote-3). 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.

## Gensim

Gensim is an open-source natural language processing Python library[[4]](#footnote-4) 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[[5]](#footnote-5) 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.

I used the LdaMulticore model from Gensim, a parallelized version of the LDA algorithm, capable of significantly reducing the training time[[6]](#footnote-6):

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.

# PROPOSED 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 chapter 3.

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.

## 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:

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figure 10: Named entity recognizer on input text

Figure 10 represents the output of the displaCy visualizer and shows an example of a text 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:

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Figure 11: YAKE output with person names

Figure 11 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.

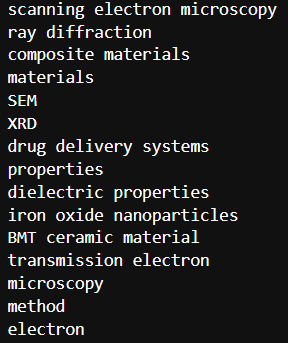


Figure 12: YAKE output without names

In figure 12, 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.

## 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.

### 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”.

O imagine care conține text, captură de ecran, Font, proiectare

Descriere generată automat

Figure 13: Significant terms incorrectly classified as ORG

Figure 13 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:

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Figure 14: keyword results after removing ORGs

In figure 14, “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.

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Figure 15: keyword results without removing ORGs

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 figure 15.

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”.

### Keyword extraction

After preprocessing the text, the program proceeds with the keyword extraction using the yake Python library[[7]](#footnote-7). 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.

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Figure 16: Result of YAKE with 20 keywords

Figure 16 shows the results of my attempt to extract 20 keywords from the abstracts of an author who has only 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.

O imagine care conține text, captură de ecran, Font, software

Descriere generată automat

Figure 17: Comparison between deduplication threshold values

Figure 17 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 side, 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 centre of the figure above.

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

## 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.

### 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).

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Figure 18: Comparison between LDA with and without unigrams

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 figure 18. If I kept the single-word tokens, collocations would not occur among the keywords. While the words on the left 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.

### 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[[8]](#footnote-8) 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 trough 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.

O imagine care conține text, captură de ecran, software

Descriere generată automat

Figure 19: Comparison between passes and iterations values

The two columns on the left of figure 19 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 figure, 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” (3.2). In my project, I did not focus very much on the coherence of topics, and I did not rely on automated tools to estimate the accuracy or reliability of the solution, but I rather focused on the results that I expected to achieve.

I used the topic visualization tool pyLDAvis to analyse 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.

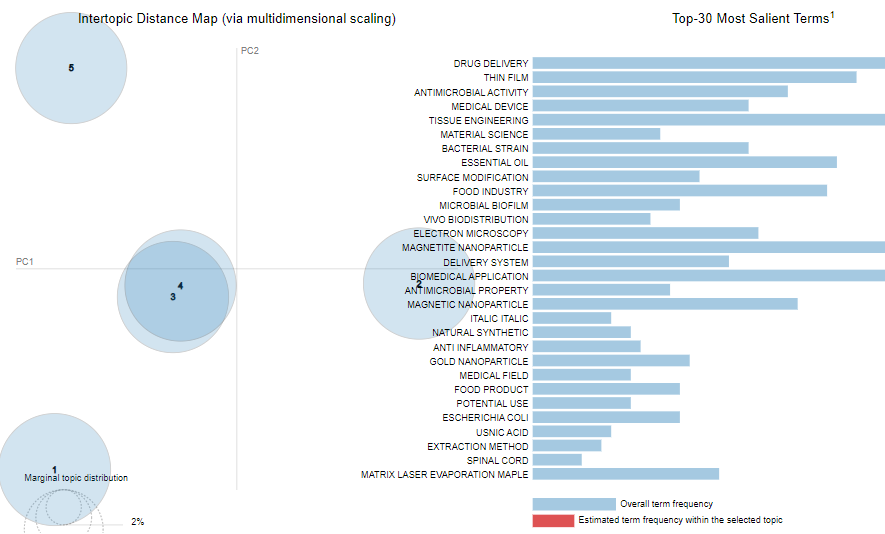


Figure 20: pyLDAvis graphic of 5 topics

Figure 20 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.

# 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 the chapter 4.

## 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.

### Parsing the input files

The source code of my keyword extractor application is a Python script called “keywords.py”, available on GitHub[[9]](#footnote-9). I parsed the two CSV files using the pandas library[[10]](#footnote-10) 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.

### 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 4.1. 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  
 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[[11]](#footnote-11). 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.

### 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 4.2.2, 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))  
# 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.

### 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 4.3.1.

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.

### 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.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Figure 21: Output file

Figure 21 shows an example of the results of my keyword extraction application, saved in a CSV file.

## 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 4.2.1, 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[[12]](#footnote-12), allowing the use of the most powerful NLP models, such as BERT and GPT, through the spaCy pipeline[[13]](#footnote-13). 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 “faculty”.

# Studiu de caz / Evaluarea rezultatelor

Acest capitol trebuie să răspundă, în principiu, la **2 întrebări** și să se încheie cu **o discuție** a rezultatelor obținute. Cele doua întrebări la care trebuie sa se răspundă sunt:

1) **Merge corect**? (Conform specificațiilor extrase în capitolul 2);

Evaluarea dacă merge corect se face pe baza cerințelor identificate în capitolele anterioare.

2) Cât de bine merge / cum se compară cu soluțiile existente? (pe baza unor metrici clare).

Evaluarea cât de bine merge trebuie să fie bazată pe procente, timpi, cantitate, numere, **comparativ cu soluțiile prezentate în capitolul 3**. Poate fi vorba de performanță, overhead, resurse consumate, scalabilitate etc.

În realizarea discuției, se vor utiliza tabele cu procente, rezultate numerice și grafice. În mod obișnuit, aici se fac comparații și teste comparative cu alte proiecte similare (dacă există) și se extrag puncte tari și puncte slabe. Se ține cont de avantajele menționate și se demonstrează viabilitatea abordării / aplicației, de dorit prin comparație cu alte abordări (dacă acest lucru este posibil). Cuvântul cheie la evaluare este „metrică“: trebuie să aveți noțiuni măsurabile și cuantificabile. În cadrul procesului de evaluare, explicați datele, tabelele și graficele pe care le prezentați și insistați pe relevanța lor, în următorul stil: „este de preferat ... deoarece …“; explicați cititorului nu doar datele ci și semnificația lor și cum sunt acestea interpretate. Din această interpretare trebuie să rezulte poziționarea proiectului vostru printre alternativele existente, precum și cum poate fi acesta îmbunătățit în continuare.

Criterii pentru calificativul *Nesatisfăcător*:

* Aplicația este testată dar rulează pe calculatorul studentului, nu există posibilități de testare, nu a fost validată cu clienți / utilizatori;
* Nu au fost realizate comparații cu alte sisteme similare.

Criterii pentru calificativul *Satisfăcător*:

* [Dezvoltare de produs] Există teste unitare și de integrare, există o strategie de punere în funcțiune (*deployment*), există validare minimală cu clienții / utilizatorii.
* [Cercetare] Principalele componente și soluția în ansamblu au fost evaluate din punct de vedere al performanței, însă nu sunt folosite seturi de date standard, există unele erori de interpretare a datelor.
* [Ambele] Discuție minimală asupra relevanței rezultatelor prezentate, comparație minimală cu alte sisteme similare.

Criterii pentru calificativul *Bine*:

* [Dezvoltare de produs] Teste unitare și de integrare, instrumente de punere in funcțiune (*deployment*) utilizate și care arată lucru constant de-a lungul semestrului, lucrare validată cu clienții / utilizatorii, produs în producție.
* [Cercetare] Componentele și soluția în ansamblu au fost evaluate din punct de vedere al performanței, folosind seturi de date standard și cu o interpretare corectă a rezultatelor.
* [Ambele] Discuție cu prezentarea calitativă și cantitativă a rezultatelor, precum și a relevanței acestor rezultate printr-o comparație complexă cu alte sisteme similare.

# Concluzii

În acest capitol este sumarizat întreg proiectul, de la obiective, la implementare, si la relevanta rezultatelor obținute. În finalul capitolului poate exista o subsecțiune de „Dezvoltări ulterioare“.  
Criterii pentru calificativul *Nesatisfăcător*:

* Concluziile nu sunt corelate cu conținutul lucrării;

Criterii pentru calificativul *Satisfăcător*:

* Concluziile sunt corelate cu conținutul lucrării, însă nu se oferă o imagine asupra calității și relevantei rezultatelor obținute;

Criterii pentru calificativul *Bine*:

* Concluziile sunt corelate cu conținutul lucrării, și se oferă o imagine precisa asupra relevantei și calității rezultatelor obținute în cadrul proiectului.

# 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

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

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

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

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.

# Anexe

1. https://demos.explosion.ai/displacy [↑](#footnote-ref-1)
2. https://demos.explosion.ai/displacy-ent [↑](#footnote-ref-2)
3. https://fasttext.cc/docs/en/language-identification.html [↑](#footnote-ref-3)
4. https://radimrehurek.com/gensim/index.html [↑](#footnote-ref-4)
5. https://radimrehurek.com/gensim/intro.html [↑](#footnote-ref-5)
6. https://radimrehurek.com/gensim/models/ldamulticore.html [↑](#footnote-ref-6)
7. https://pypi.org/project/yake/ [↑](#footnote-ref-7)
8. https://radimrehurek.com/gensim/models/ldamulticore.html [↑](#footnote-ref-8)
9. https://github.com/AnaNastase/licenta [↑](#footnote-ref-9)
10. https://pandas.pydata.org/ [↑](#footnote-ref-10)
11. https://fasttext.cc/docs/en/language-identification.html [↑](#footnote-ref-11)
12. https://huggingface.co/docs/transformers/index [↑](#footnote-ref-12)
13. https://explosion.ai/blog/spacy-transformers [↑](#footnote-ref-13)