UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
CALCULATOARE ȘI TECHNOLOGIA INFROMAȚIEI

PROIECT DE DIPLOMĂ

Topico - Optimizare a algoritmului *Latent Dirichlet Allocation*

Coordonator:
Conf.dr.ing. Cătălin LEORDEANU

Absolvent:
Andrei RAȚĂ

BUCUREȘTI

Julie 202

POLITECHNICA UNIVERSITY OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Topico - An optimization to *Latent Dirichlet Allocation*

Thesis supervisor:
Ass.Prof.Phd.eng. Catalin LEORDEANU

Absolvent:
Andrei RATA

BUCHAREST

July 2021

Table of Contents

Abstract

În această teză este prezentată o optimizare a algoritmului Latent Dirichlet Allocation, ce presupune posibilitatea de a alege din mai multe modele pre-antrenate, folosind un algoritm de similaritate între cuvinte, pentru a avea predicții mai bune și pentru a obtine topicuri mult mai relevante.

In this thesis is presented an optimization of the Latent Dirichlet Allocation algorithm, which implies the possibility of choosing between multiple pre-trained models, using a word similarity algorithm in order to make a better prediction and to obtain better topics that are more relevant.

1. Introduction

Nowadays, the quantity of unsupervised data that circulates the internet is growing exponentially everyday. There are moments when having this data structured can be of great help in different situations. To solve this issue, domains like *topic modelling* were invented. When it comes to *natural language processing* (**NLP**), *topic modelling* is a domain that helps computers extract abstract information or the central subjects from a given text. This kind of information is named *topic*, which offers a brief description about the entire provided text. The main domains in which *topic modelling* is used are marketing & ads recommendations, medicine, news article summarization, chat topic extraction etc. For example, when analyzing the transcript of a discussion between multiple people, *topic modelling* can be used to determine what subjects were discussed and classify the entire discussion into different given categories, such as sports, business, technology etc.

One of the mostly used solutions, when it comes to topic prediction, is *Latent Dirichlet Allocation*, also found under the acronym **LDA**. What this algorithm does is to find a bunch of words that will describe the best subtopics of a given corpus of documents. Most of the time, with this *topic modelling* solution, everyone can categorize their unsupervised data with great ease.

From multiple experiments and previous works of other scientists, it was observed that the **LDA** algorithm has some limitations, which in some cases, can lead to poor performance regarding the quality of the subtopics generated:

- **when a model is created, the subtopics should be known beforehand;**
- **after a model is trained, it can NOT be changed**

Having in mind the previous issues, this thesis proposes a solution in which there is no limit regarding the addition of new models and improving the prediction results. The results outlined in this project are encouraging and show the big difference between using a single model trained with all the data an user wants to use and dividing a model into more pieces, each one being centered around one domain.

The rest of the thesis is organized as it follows. Section 2 presents *Related work* and an analysis of previous research on *topic modelling* and **LDA**. Section 3 describes the concepts used and the proposed solution from a theoretical point of view, while Section 4 describes the implementation and the APIs which were used. Section 5 contains the test scenarios and experimental results, while Section 6 concludes the thesis and presents directions for further improvements.

2. Related Work

Nowadays, in the IT industry there are a lot of companies that are looking forward to optimizing their process of online working, taking into consideration that working remotely is an ascending trend. A way speculated by the author was that there are companies which want to categorize their online meetings, based on its transcripts. Of course, an approach is to use existing methods from **NLP** to achieve the desired results. However, like it is presented in the following sections, there are limitations that prevent the achievement of ideal results. One main objective of this thesis is to offer more flexibility and to achieve this result, it is necessary to analyze the existing work and see what the actual progress of the topic modelling domain is. As it is described in [1], the evolution of *topic modelling* is growing every year exponentially from 2007.

Also, there are solutions to this issue of categorizing unsupervised data that followed a linear path in terms of the algorithms and methods applied. In the article presented in [2], the authors wanted to outline how powerful a classical method like **LDA** can be in relation to categorizing data. The proposed solution was a framework, created by the authors to facilitate the work of the researchers when creating an **LDA** model, by using a given corpus of documents. Using *5-fold cross-validation*, they discovered on their dataset that the optimal number of topics would be twenty. Although the framework is of great use, they acknowledged that one limitation is the way topics are labeled, therefore not so optimal or nor even possible. Furthermore, at a closer analysis, when the training process is started, the relevant words of the corpus need to be known in order to run their framework, which can cause an overhead on large datasets.

Often, in *natural language processing*, topics are generated using the most relevant words from the data used as inputs, but these words are not analyzed from a semantical point of view. All the words are treated like chunks of characters that have a high relevant factor based on the corpus they belong to. In [3], the authors present a solution in which they design an algorithm combining **LDA** models that take into account ontology-based concepts. With these principles, words are linked one to another from a practical point of view. The authors used *DBpedia*, a large database that classifies most of the words into *classes, subclasses, domains* etc. Their solution is named *OntoLDA,* and it introduces another step into classical **LDA**, by replacing the extracted relevant words with their root class. Even though the results are encouraging and offer more descriptive information about the topics, overhead can be easily created when queries are computed to find the root class. A more detailed description about the performance of this kind of database can be found at [4]. Nevertheless, to run the entire configuration, the authors had to know their topics before.

Another form of data that needs to be taken into consideration when trying to optimize to the *topic modelling* domain is finding a way to process conversational dialogue records. This is considered a different situation, even though it is also unstructured data, because in dialogues there is a lot more *noise* (data that it's eliminated in the filter process) than in other situations. According to [5], in 2018, almost 277 million Americans received almost 3000 messages per month and those numbers are growing. In [6], the authors analyzed in detail how they could handle this kind of data using **LDA** and SVM in order to extract relevant data out of messages and to find which would be the best approach to this kind of situation. From their results, they outlined that an **LDA** model has better performance than the SVM. As expected, one limitation of the paper consisted of too many superfluous words like "It's alright", "Not bad", "Have a nice day".

In many situations, as previously presented, the meaning of words is more important than the actual words. If the dataset is poorly chosen, then the results can be of no value, even though the algorithm returns a bunch of words which will describe the generated topics. In [7], the authors proposed a solution in which every word is assigned a class of words with more meaning than the actual words. With this they experimented with a technique based on summarization on a summarization frame (put the information into order using a coefficient of importance). The advantage of this paper is that in the summarization frame not only words can be included, but also bags of words which can be described using only one word (e. g "Not wrong" associated with "Right").

The presented articles were taken into consideration in achieving the desired optimization. In most of the articles analyzed, the main limitation identified was the flexibility of generating new topics. If a model was created with several topics, to make a change, the model needs to be recreated and retrained. In this thesis, the main objective is to create an environment, in which, adding more topics to be taken into consideration for further prediction should be much easier and efficient than the solutions that exist.

To demonstrate that the main objective was achieved, in the *Results section* an analysis was made, to show the differences between existing methods and the optimized method, a good dataset and a poorly chosen dataset and performance graphs that illustrate metrics of relevancy of the created models.

3. TOPICO - theoretical description

3.1. Natural language processing (NLP)

All the data gathered from human communication is of no use if a computer cannot understand the used language and rules used. What **NLP** does is to offer the possibility of communication between these two entities. Even though currently, the **NLP** methods are not ideal, the progress is increasing yearly. In order to achieve the understanding of human language, the most common solution uses machine learning algorithms and artificial intelligence. [8]

In the early days (about 1950), achieving the understanding of human languages by computers was harder, due to the limitations of that period. At first, for a long period of time, it was used the *Chinese room experiment* [9] proposed by John Searle. Through this experiment, it was proven that being given a series of questions with straight forward answers, a set of rules can be created, and every chunk of text can be processed, using pattern matching.[8]

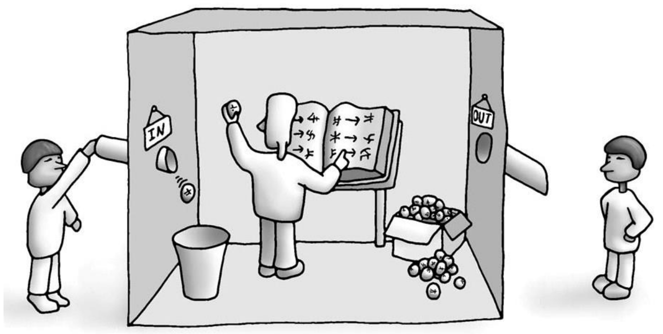


Fig. 3.1. Chinese Room (from [9])

Nowadays, the technologies have evolved, complex processing being done using complex rules. A great help in this domain are *deep neural networks*, offering the possibility of different algorithms to learn by the "try and fail" method to find better ways of analyzing and processing large amounts of data [8], [10]. Also, with *neural networks* problems like dimensionality and using hand-crafted features, are handled with ease. Some examples of the used methods are the following:

- ***Recursive Neural Network*** - graphs used to decompose sequential sentences and every node can have attached a weight learned from multiple iterations;
- ***Character Embeddings*** - mostly used to tag sequence of characters in words;
- ***Unsupervised Learning*** - based on existing root sentence, identify other sentences [8], [10].

The main challenge of **NLP** solutions, is that most of the data is unstructured and dependent on the language used, the sets of rules can create a great overhead. According to [11], there are almost 6,500 languages spoken in the entire world, each of them having their own complex set of rules and a wide vocabulary. A great advantage of humans over computers, is that they can easily understand ambiguity and by using intuition, it is much easier to apply the rules of a specific language [8], [10]. When it comes to computers, this cannot be replicated with great ease, because they do not have consciousness. For example, if an article speaks a lot about the cars that a lot of sportsmen possess, a computer may fail to identify the main subject, due to the great amount of details on topic of cars instead of sportsmanship. On the other hand, a human with the possibility of reading between the lines, may discover that the main subject is about the prosperity of an athlete that can be achieved from professional sports [8], [10], [12]

To achieve common language understanding, **NLP** takes into consideration the following concepts: *Semantics* - the meaning of the words and *Syntax* - the patterns applied to match language rules. Also, there is a bunch of algorithms that is used in order to extract most of the data that needs to be processed such as: *summarization* – simplification of the text for a better understanding, *topic modeling* - topic extraction from data, *text generation* - used for machine learning task, *sentiment analysis* - getting the feeling transmitted by chunks of text, *text-to-speech* - signal processing that transforms audio into words, matching letters frequency etc [8], [10]. Even though the applications of this domain may differ and be used for different purposes, they have in common some steps that need to be followed to achieve the final results:

- ***Text cleaning*** - removing all the unnecessary data, because it can affect the final results;
- ***Tokenizing*** - break the input, into lists of tokens;
- ***Stemming / Lemmatization*** - methods used to extract the root of words.

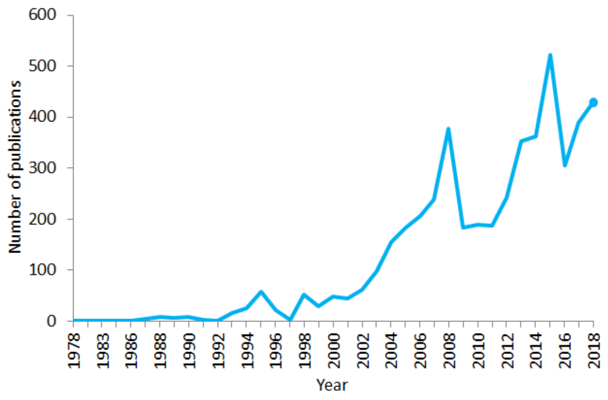


Fig. 3.2. The trend of publications about **NLP** (from [12])

As shown in Fig 3.2, **NLP** popularity is increasing over the years, so it became a standard in terms of communication between humans and computers. The domains in which it is the most popular are *sales*, *medicine*, *marketing*, *psychotherapeutic* etc [12].

3.2. Lemmatization / Stemming

To obtain better results when it comes to text processing, the data needed to be analyzed goes to various stages of filtering. One of them is *Lemmatization/Stemming***,** which has the main objective of reducing words to their root form [13].

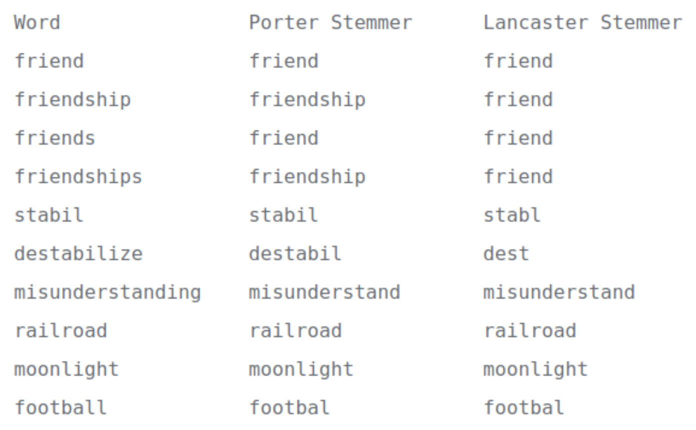
Every language has its way of deriving words from their dictionary form, to offer more grammatical categories to them. An example of this type of modification can be observed in *tense*, *gender* etc. From a **NLP** perspective, *conjugation* or other grammatical changes may affect the final results (e. g. *beautiful* and *beautifully* represent the same word, but one is an *adjective* and the other is an *adverb*). For this issue, *Stemming* comes in handy, because it represents an algorithm that removes all the inflected parts of a word, only the root form remaining from the word[14]. The most used algorithms that handle stemming are the following: *Porter Stemmer* and *Lancaster Stemmer* [13], [14].

The *Porter Stemmer* algorithm was developed by *Martin Porter* in 1980, and had a set of pre-defined rules, stripping the words of their suffix to obtain the root part of words [13]. The great advantage of this implementation is its speed, but since these rules are rudimentary, the obtained words may not be actual words from the dictionary. The rules represent a series of patterns applied to words, not taking into consideration the word from a semantical point of view.

In [15] it is presented the official paper of the algorithm written by Porter. What the stemmer basically does is follow a set of grammatical rules to remove the unnecessary parts of the words. For example, some rules defined in the paper are:

- **ATIONAL -> ATE** (e.g. *relational -> relate);*
- **ENTLI -> ENT** (e.g. *differently* -> different);
- **OUSLY -> OUS** (e.g. a*nalogously -> analogous)* [13], [16].

The *Lancaster Stemmer* algorithm was developed by *Chirs D. Paice* and represents another algorithm used for removing the inflected parts of the words. The way the algorithm works is that it takes constantly into consideration the last letter of every word, trying to pattern match its set of predefined rules. A major issue with this approach is that it creates a large amount of *over-stemming errors*, which means that from different words the same stem is obtained. Also, the fact that the algorithm takes into consideration the suffix letters of words, it causes heavy processing due to multiple iterations. In Fig. 3.3, it can be observed a comparison between the two *stemmers* presented [13],[14].



Word	Porter Stemmer	Lancaster Stemmer
friend	friend	friend
friendship	friendship	friend
friends	friend	friend
friendships	friendship	friend
stabil	stabil	stabl
destabilize	destabil	dest
misunderstanding	misunderstand	misunderstand
railroad	railroad	railroad
moonlight	moonlight	moonlight
football	footbal	footbal

Fig. 3.3. Porter vs Lancaster Stemmer (from [13])

Depending on the result, these *stemmers* have different use cases. If the main objective is to trim down the dataset, *Lancaster* is a good choice, but if speed is necessary, *Porter* is a much more viable solution.

Lemmatization does almost the same as the *stemmers* presented previously, but the inflected words keep their form from dictionary, having an understandable format. In **NLTK**, lemmatization is using the *WordNet* database, to obtain the root form of the words [13], [17]. Even though this solution of removing inflections of words is much slower than previous stemmers, in some cases, the meaning of the words needs to be preserved.

3.3. Tf-idf

Tf-idf stands for *term frequency–inverse document frequency*, which represents a function that calculates a weighting factor for every word in a corpus of documents to remove the most commonly used one. A word that appears in all the documents, may not have a great relevance in terms of presenting the main idea of a text. The more a word appears in a document, the less important it becomes. This method appeared, because removing the stop words from a text and then calculating counts for every word, to find the most used ones, in some cases may not be the ideal method [18], [19].

As presented in the previous paragraph, **tf-idf** is composed from two components: *term frequency* (1) and *inverse document frequency* (2). Basically, these are two factors that are calculated independently, for one word, and they are multiplied to obtain the desired factor [18], [19].

Term frequency stands for the number of occurrences of every word in a specific document related to the total number of every word of the document they belong to. With this factor, it is determined how a word is relevant to only one document, not taking into consideration the entire corpus. The equation used to compute this metric can be expressed as:

$$tf(t,d) = \frac{ft',d}{\sum_{t' \in d} ft',d} \tag{1}$$

where *t* stands for a term in a document, *d* represents a document, $f_{t',d}$ represents the number of occurrences of term *t* in document *d* and the sum represents the total number of words from the document *d* [19].

Even though the *term frequency* can provide a lot of information about the weight of a word in a document, there are times when the obtained factors are not that relevant. For example, words *this*, *no, and* can get in some situations higher scores according to the content of the document. To solve this issue, *inverse document frequency* was introduced as a heuristic that considers the quantity of information a word can give. Basically, with this factor, stop words or words with high occurrence rate, will have a lower score. A word being rare means that it can provide a lot of information compared to the others. The equation for this score is defined in [18], [19] as:

$$idf(t,D) = log \frac{N}{|\{d \,\varepsilon\, D : t \,\epsilon\, d\}|} \tag{2}$$

where *N* represents all the occurrences of the term *t* in the entire corpus and the modulo stands for the number of documents the term *t* appears in.

The reason why a log function is used is to dampen the effect of the *term-frequency factor*, is because it would not be efficient to use another linear function. Note that every mathematical existence rule needs to be respected, so the module should be a number which is at least bigger than zero. A good **tf-idf** factor will be obtained when the **idf** is lower and the **tf** is higher. In this way, a lot of unnecessary words will be filtered out of the given text. In [19] the final equation can be set as:

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,d) \tag{3}$$

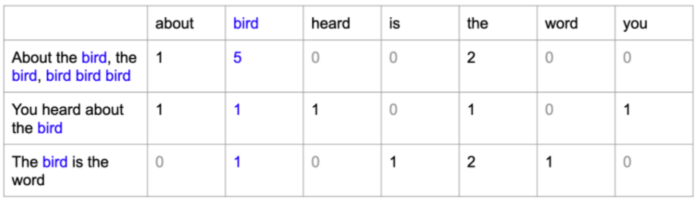
3.4. Bag of words

In *topic modelling*, every algorithm works with unstructured data and the main goal is to extract the most valuable data from the input. A mandatory step in this process is to structure the vocabulary in a way that creates an overview of the word distribution in the given text. *Bag of words* (**BoW**) represents a solution to this issue [20], [21], structuring all the words of a vocabulary within following format:

```
[{word : occurrences}]
```

Fig. 3.4. *Bag of words* format

The format is a list of **JSON** and every entity from the *bag of words* represents a pair containing every word with the according occurrences in each document of the corpus. Usually, when unstructured data needs to be processed, it is not used the raw version of it, but the *bag of words* generated from the input. So, if a corpus of documents needs to be processed, every document needs to be transformed. Note that every word is analyzed being related to the entire corpus of documents [20], [21]. So, the final structure will have all the words with the occurrences count in all the documents as is shown in Fig. 3.5. With this method, it is much more efficient to generate features necessary for the *topic modelling* algorithms. Also, if a word has a higher count than the other from the **BoW**, it does not mean that it is more relevant. The highest occurrence rate is achieved by stop words like *a*, *this*, *not*, which does not provide any information about the given data [20].



	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Fig 3.5. *Bag of words* representation on corpus of documents (from [22])

Even though this format is useful for many reasons, it has downsides that can affect the performance of the *topic modelling* algorithms. For example, if the corpus is large and the majority of words are not duplicates, the generated **BoW** can cost a lot of space memory. This is considered an issue because the *topic modelling* domain consists of processing large amounts of data. Also, an optimal version of **BoW** represents using a *Hashing dictionary*, in which every word is mapped to a generated hash. Some applications in which **BoW** is used are: *spam filtering* – removing the spam words from a document, *text-mining* etc [20]-[22].

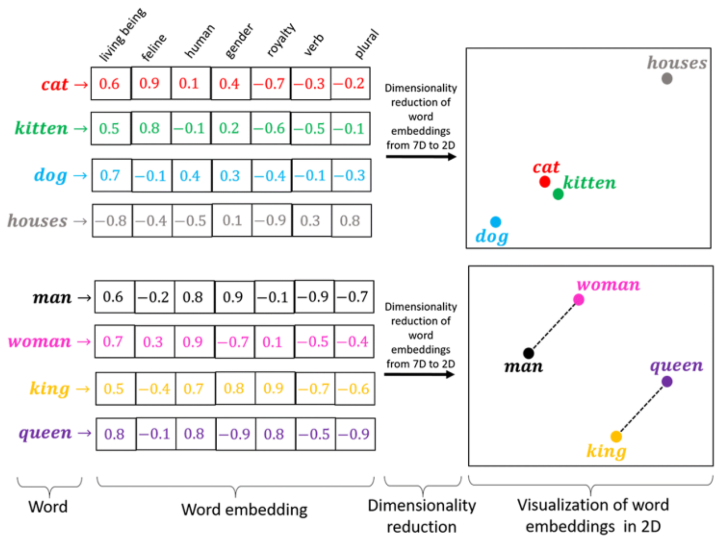
One of the most used implementations of **BoW** is provided by *sklearn* because it provides a great level of abstraction, easy to set up and use [20], [22]. The model provided by the framework is named *CountVectorizer*. To run the vectorization, all that is needed is just to call the *fit_transform* method, which will learn the vocabulary and return the desired representation [20].

3.5. Word2vec

To provide a better understanding of this notion, it is necessary to firstly first a list of definitions:

- *word embedding* (**WE**) **-** representation of word using a vector of numbers;
- *softmax function* (**SF**) - normalizing a vector in order to obtain a percentage or probability distribution;
- *neural network* (**NE**) - digital simulation of how a brain work, taking decisions based on specific metrics [23]-[25].

Word2vec represents an algorithm that generates **WE** for a vocabulary. With this representation, synonyms can be found with great ease because between vectors mathematical operations like *addition*, *subtraction* etc. can be computed In this way, a similarity measure can be defined because words with almost identical **WE** have a higher probability having the same meaning[23]-[26].



	living being	feline	human	gender	royalty	verb	plural
cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8
man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
Dimensionality reduction of word embeddings from 7D to 2D
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Fig. 3.6. Word to word embedding representation in 2D (from [26])

Also, **WE** are commonly used to *deep learning* because every index of the obtained vector represents a *feature*, which means a piece of information that can describe the desired object needed to be analyzed. In Fig. 3.6 are illustrated some examples of *features* (*feline*, *human, living* etc.). In order to obtain these vectors, a **NE** is necessary. At first, the vectors for every word are initialized with a **hot vector -** having the same size of the vocabulary with **1** on the target word and 0 on the others. What a **NE** can do, is to update these values, till the final results are relevant and the synonyms obtained are precise. When it comes to *Word2vec* this can be accomplished by using: *Continuous Bag-of-Words Model* and *Continuous Skip-gram Model* [23]-[26].

Continuous Bag-of-Words Model (**CBOW**) is an algorithm which, based on the other words of the document, can find the best match of a focus word (e.g. If the corpus is *A*, *wonderful, human, day, makes, is, happy, foggy* with the sentence is *A wonderful day make a human happy* and the targeted word is *day*, will return words like *happy, a, make*). This is done using a **NE** (Fig. 3.7) that in the end it will output a series of *weights* which will represent the **WE** of the words. Also, the final result will be the targeted word, in the defined context. Usually, a *window* is set to show how long the context should be. For example, for a sentence like *It is raining outside*, a *window size* of 2 would look like: [*raining, outside*], [*is, raining*]. This model being based on predictions made by the **NE**, is not perfect, but there are already pre trained models that can be used, which have good performance [27]-[29].

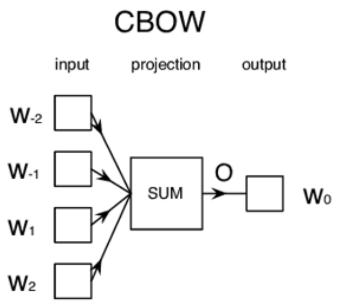


Fig 3.7 Neural network used for **CBOW** to find the target word (from [27])

Continuous Skip-gram Model is an algorithm that is very similar to the previous one, but instead of finding a target, it generates all the possible options for a given word. For example, if the vocabulary is *bank, account, mortgage, juice, orange*, *money*, if the target word is *money*, the results obtained from the prediction will be words like *bank, account, mortgage*. Also, this model uses a **NE** too, with a similar workflow as described in Fig. 3.8. This model is useful, when the words are rarer, because it can discover other rare words that provide a lot of information [27], [29], [30].

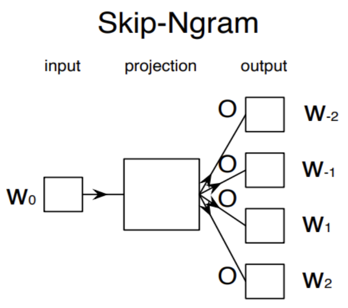


Fig. 3.8. Neural network used for *Skip-Gram* to find the target words (from [30])

3.6. Latent Dirichlet Allocation

To define *Latent Dirichlet Allocation*, the following notions are needed for a proper understanding:

- k - 1 simplex;
- Γ(x) - Gamma Function of x;
- k-dimensional Dirichlet Distribution.

Gamma function of a number x, represents the factorial value of the given number as follows:

$$\Gamma(x) = (x - 1)! \qquad (4)$$

The equation (4) is true, if and only if *x* is a *real number*. Also, there is a formula for *irrational numbers*, but for *Latent Dirichlet Allocation* it is not necessary [31].

Given a vector $\theta = (0,1,2 \ldots \mathrm{n})$, $\forall\, n \geq 0$ and $\forall\, \theta_i \geq 0$, a *simplex* distribution will represent $\sum_{i=1}^{n} \theta_i = 1$ [31].

Having a simplex distribution of θ, in a **k** dimension and with a vector α, a *Dirichlet distribution* in [31] is defined as:

$$p(\theta \mid \alpha) = \frac{\Gamma(\sum_{i=-1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \cdot \theta_1^{\alpha_1 - 1} \ldots \theta_k^{\alpha_k - 1} \tag{5}$$

Being given a set of points, a *Dirichlet distribution* positions these into the given space using the previous formula, depending on the chosen vector α. For example, given a 2d space as triangle, this means that all the points distribution may converge uniformly to the corner or to the center of the environment, as shown in fig 3.9 [32].

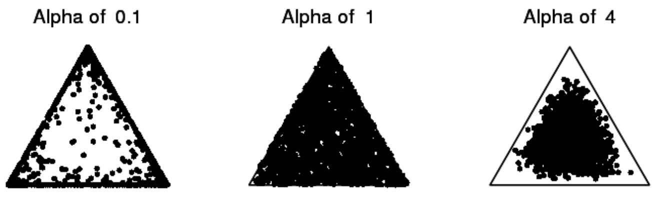


Fig. 3.9. Alpha parameter effects on *Dirichlet distribution* (from [32])

Having these concepts presented, *Latent Dirichlet Allocation* represents a generative model, based on *Dirichlet distributions* which extract topics from a given text. The way the model works is that having a specific document, that needs to be processed, there are multiple attempts to recreate it with a predefined β and α which offer a distribution of words over over topics and a distribution of topics over words. As presented in (6), the document is represented by a probability. Trying multiple possibilities of these factors, best settings lead to obtaining a document closer to the initial document [32], [33].

$$p(\theta, z, w \mid \alpha, \beta) = p(\theta \mid \alpha) \prod_{i=1}^{N} p(z_n \mid \theta) \cdot p(w_n \mid z_n, \beta) \tag{6}$$

Fig 3.10 presents the blueprint of the *Latent Dirichlet Allocation*, where $\alpha$is the distribution that assigns documents to topics, $\boldsymbol{\theta}$ distribution for picking topics, η distribution for assigning topics to words and β distribution for obtaining words. Using these distributions combined, it results in **z** and **w** that generates the document from the initial ones. Note that the final topics generated by the topics will be a bunch of words obtained from the picking of words made from distributions [32], [33].

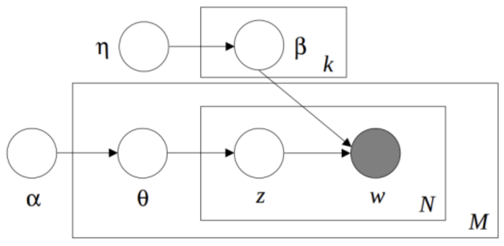


Fig. 3.10. *Blueprint* of smoothed **LDA** model (from [32])

The advantage of *Dirichlet Distribution* are the associations between the analyzed data and the space generated. For example, α distribution which assigns topics to documents and there are only three topics, the *Dirichlet space* will be a triangle with the edges representing the topics. This distribution generates points inside the triangle and being close to an edge means that the document is much more likely to belong to that topic, as shown in Fig 3.11. Furthermore, every distribution used in the Blueprint of the model is represented as in the previous case [33].

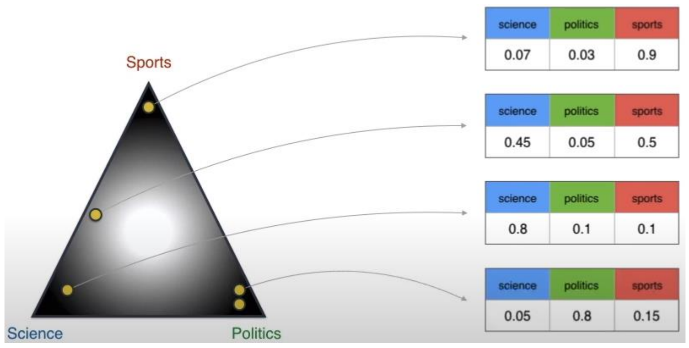


Fig. 3.11. 2D space with three topics distribution example (from [33])

The size of the *Dirichlet distribution* environment represents the number of topics that can be found in the document corpus. So, to compute the algorithm, the number of topics needs to be known, in order to define the space in which the relationships between *topics* and *words* are established [32], [33].

3.7. WordNet

WordNet is an open-source lexical database that has strong relations between words, like a digital dictionary. One of the biggest advantages of this database is that every word is connected in a graphical way, based on the semantical meaning of the words. If two words are close one to another, it means that they are synonyms. Every element of the database is called *synset***,** which can lead to other words based on the same meaning. Also, *WordNet* is similar with the ontology defined between words. For example, *nouns* are in relationship with *verbs* and can represent categories which can lead to other existing nouns [34]-[36].

This database is part of the **NLTK** corpus, which represents a large framework implemented in python for *natural language processing*. Having this layer of abstraction, *WordNet* can be easily included in any project, in order to solve multiple problems when it comes to analyzing unstructured data. For example, according to Fig. 3.12, it's the necessary code to obtain the *sysnet* of any word from a given dictionary. The results obtained have the following structure: *word.type.number*, where *word* represents the word searched in *WordNet*, *type* represents the lexical form of the word(n - Noun, v - Verb, r - Adverb, a - Adjective) and *number* the semantic value used to create relationships [34]-[36].

```
from nltk.corpus import wordnet
syns = wordnet.synsets(word)
```

Fig. 3.12. Code for obtaining the sysnet of *word*

Note that *WordNet* offers more flexibility when it comes to finding how similar two words can be from a semantical point of view. *Path similarity* is an algorithm which calculates the distance between two words, using *WordNet* and based on the final result obtained, it can settle how close these words can be [34]-[36].

Even though *WordNet* is highly used, it has limitations. There are words that are harder to achieve a good representation in the *WordNet* graph, considering they share almost the same area of interest. For example, *emotions* are not well distributed in this graph [34]-[36].

4. Implementation details

The proposed thesis describes **TOPICO**, a solution which optimizes existing *topic modelling* prediction systems. This algorithm implies several predefined **LDA** (*Latent Dirichlet Allocation*) pre-trained models with desired data. When a text needs to be categorized, a choice is made between those models, so that the final result will lead to more relevant topics. The project consists of a software application divided in two main parts: *topico-algorithm (main core)* and *topico* - web application.

4.1. Topico - Algorithm

For the algorithm to work, each model needs to be trained with specific data that is centered around a subject. (Ex: Model trained only with data about Sports, Music or Art). When a text is processed and relevant topics need to be predicted, the next steps are executed. Firstly, the relevant topic domain must be chosen, which is done by calculating a similarity factor between the keywords extracted from the input text and each keywords of the available models. Afterwards, using the identified model, a prediction is made and the most relevant subtopic is found. Thus, the algorithm can be described using the following steps:

- **Input filter** - elimination of unnecessary tokens;
- **Relevant word extraction** – extracting the most used words;
- **Model selection** – selection of the most relevant model based on a similarity factor calculated between the keywords of every model and the keywords of the input text;
- **Topic prediction** - actual prediction of the subtopics, with emphasis of the most relevant.

4.1.1 Input Filter

To obtain the best results, the text needs to be filtered because most of the time, raw data contains a lot of unnecessary words that need to be excluded. If this step is not properly done, the results will not be relevant.

	Type	Meaning
1.	fl	File type
2.	df	Dataframe type
3.	ls	List type

Fig. 4.1. Type Raw Data

First, the input type that is given needs to be identified. The types that are accepted are presented in Fig.4.1.

The type is passed as an argument to the function that starts the processing. If no type is provided, raw data is a directory (This method was used only for experimental reasons). The most used format is **fl**, due to performance. As is shown in Fig 4.2, the first step of filtering represents tokenizing the given data, which means breaking the texts into words and making a list from it. The separator used for this process is a blank space. Next step is to remove all the stop words (most used words as "the", "as", "to" and so on.). To achieve this step, all tokens from the text are transformed into lowercase words. This step is necessary because words like "The" and "the", would be considered different words, even though they are the same. The list of known stop words is provided by **NLTK**, and all of them are lowercase words. The list is used to know which token needs to be eliminated. For better performance, punctuation, alpha numeric strings, and words with length smaller than three characters are also removed from data.

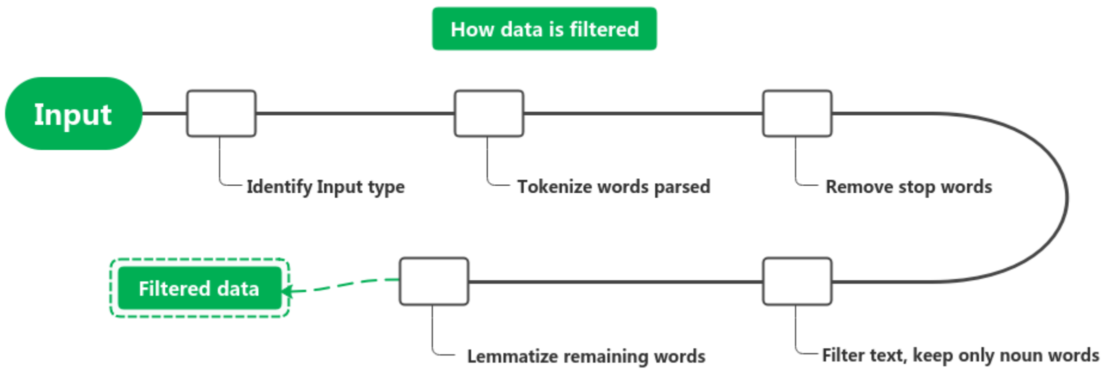


Fig. 4.2. Filtering process

Now that the stop words are removed, the next step is to only keep nouns to our data, because other types of words would not be relevant (adjectives like “beautiful” or “nice” are too generic for the meaning of a topic). In this case, **NLTK** is used to tag every word, using *pos_tag* speech tagger for English. Using a very large dictionary, *pos_tag* is capable of identifying almost every word from a language (e.g. English) and assigning a tag for each word. After the assignment, only the words with the tag **NN** (Singular noun), are kept for further processing.

In the final step, every remaining token is lemmatized, as Fig.4.3 shown, which means that every word is replaced with his form from the dictionary.

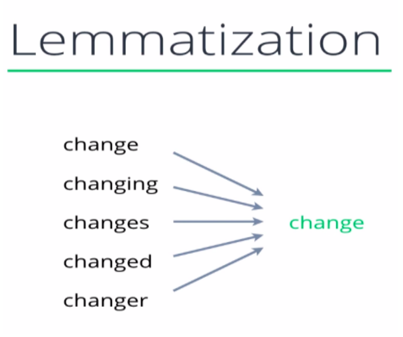


Fig. 4.3. Lemmatization (from [37])

4.1.2 Relevant words

Now that the data is filtered, the most relevant words need to be extracted. To complete this step, a *CountVectorizer* model is used, provided by *sklearn* (Machine Learning Library). The main purpose of the model is to count all the occurrences of every word from a corpus of documents. In this case, the corpus is composed only from one document, created from the filtered tokens. For this model to work, it is necessary to construct a string from the processed tokens, each one separated by a space. After calling the *fit_transform* method of the model, an array with all the occurrences number of every word is created. For the next step, only 10 of the most used words will be selected.

Another tested solution was using a *TfidfVectorizer* model, but the relevant words extracted were not as useful as expected (as shown in Fig. 4.4). The major issue is that this kind of model works well with predefined corpus of documents but has low performance when we want to obtain the most relevant words from a new text, not from the initial corpus. In fact, the **tf-idf** algorithm takes into consideration the number of documents used in the corpus. In our case, **TOPICO** predicts the topics for only one text at a time, so our corpus will consist of one document, every time. Also, based on the training data, a word that is important for the text we want to predict, it might be ignored because it does not appear in the trained model. In other words, the ideal dataset to train a *TfIdfVectorizer* to contain all the words with their real relevance, is almost impossible to find.

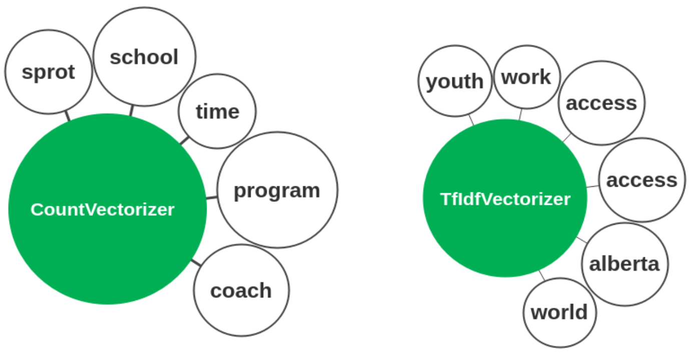


Fig. 4.4. *CountVectorizer* vs *TfIdfVectorizer*

4.1.3 Model selection

The main feature of the algorithm is the possibility of choosing between multiple pre trained models and finding the best match for the text. When a model is created, a name should be assigned to it, for easier identification. This name should be descriptive and relevant to the model, to obtain best results. When the relevant words are extracted, the next step is to create a phrase from them and find the similarity factor between it and every topic name created. Fig. 4.5 reveals the biggest similarity factor computed will determine which model name is chosen.

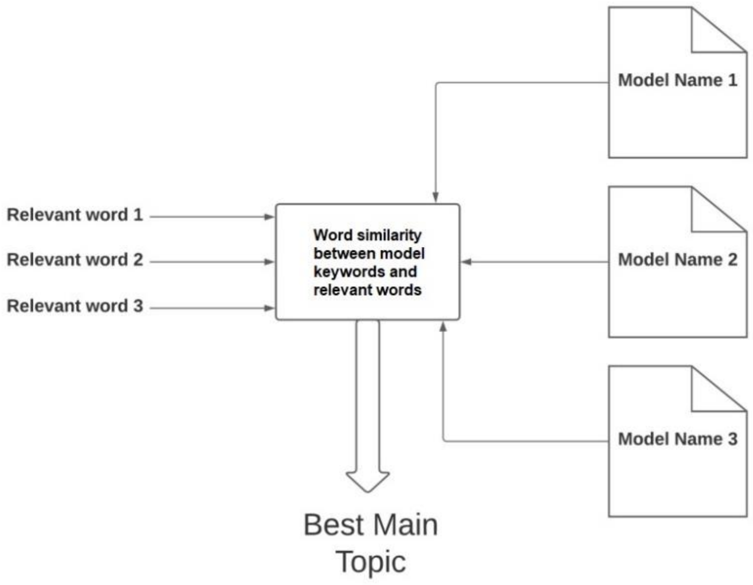


Fig. 4.5. Word similarity algorithm workflow

Word similarity can be computed because of the *WordNet* database. Every word from English is stored in a hierarchical way (as is illustrated in Fig. 4.6), being connected with most of its synonyms. In this way, words can be compared from a lexical point of view. To determine if words are synonymous or not, *path similarity* is the used algorithm. In the graph provided by *WordNet*, every word being connected, there is a distance from one word to another. What path similarity does, is to compute this distance for two words that need to be analyzed.

As a first solution to find the best match, the word similarity implementation used in **TOPICO** considered every topic model name as a sentence. To obtain the best result, for every relevant word, a search in the *WordNet* graph is computed to find the parent meaning. After the list of parent words is created, the algorithm calculates path similarity between every topic name and all the tokens. The saved result represents the maximum value between every path similarity factor calculated, at an iteration. After every factor is calculated for each topic, the model's name with maximum value, is the best choice.

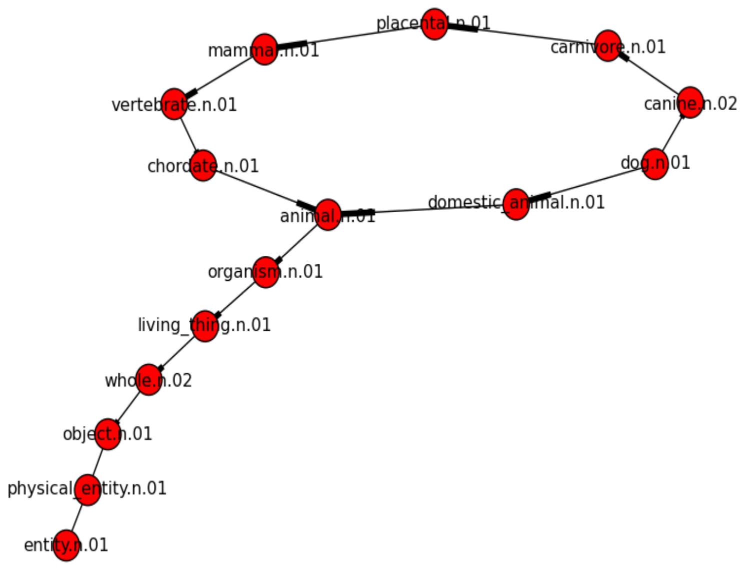


Fig. 4.6. *WordNet* distribution of words (from [38])

Taking into consideration that one word cannot be very relevant to the entire corpus and the results were not as good as expected, another solution was used in the final implementation. For every model created, 30 of the most used words of the corpus are used to compute the word similarity algorithm. Using a *CountVectorizer* model, the most used words are extracted. The algorithm calculates the *Path Similarity* between every key word of the processed text and every model keywords with the final score representing the similarity factor. This step is repeated for every model and the best similarity factor represents the model used for further prediction.

In the training phase, taking into consideration that the models are trained using *sklearn,* a *CountVectorizer* is trained too with the desired data, to be fed to the *LatentDirichletAllocation* model. Having the counter model created, in this way, the most used words are extracted for each model, when it is created. Every array of key words is associated with the model name, in order to know which should be used in the next steps. This provides a better solution for model selection and it is currently used by **TOPICO**.

4.1.4 Topic prediction

Knowing what is the best model name, the prediction step can be established. The models are persisted in a database and they are identified by the name obtained in the similarity process. Every model is an **LDA**, so the data that is fed needs to be vectorized. Using the *CountVectorizer* model used previously, we can provide the data to the model. After learning the input data, a distribution of the subtopics is generated. The highest factor obtained represents the most suitable subtopic. Every model is created using *sklearn***,** so learning a document and finding its matched topic is done by calling the method *transform* on the vectorized and filtered data.

4.2. Topico - Web Application

The algorithm previously presented represents the core engine for the main application (workflow described in Fig 4.7), which is a web page in which every user can create an account and start creating his own **LDA** models to start categorizing unsupervised data into topics.

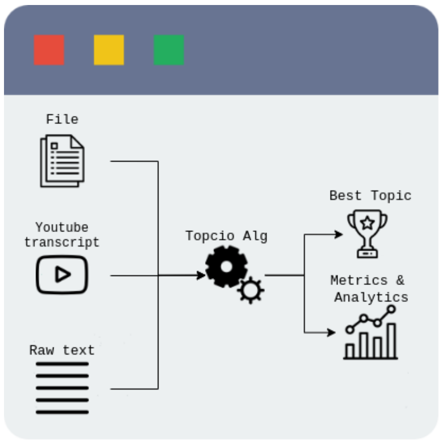


Fig. 4.7. Under the hood - **TOPICO** Web Page

4.2.1 Architecture

The application runs on *Flask server*, with endpoints that are called by the UI, built using *ReactJS* framework. There are three main components used:

- **main_page** - upload data to make predictions;
- **topics** - display and create topic models with prediction metrics;
- **statistics_and_logs** - display performance info and history with all predictions.

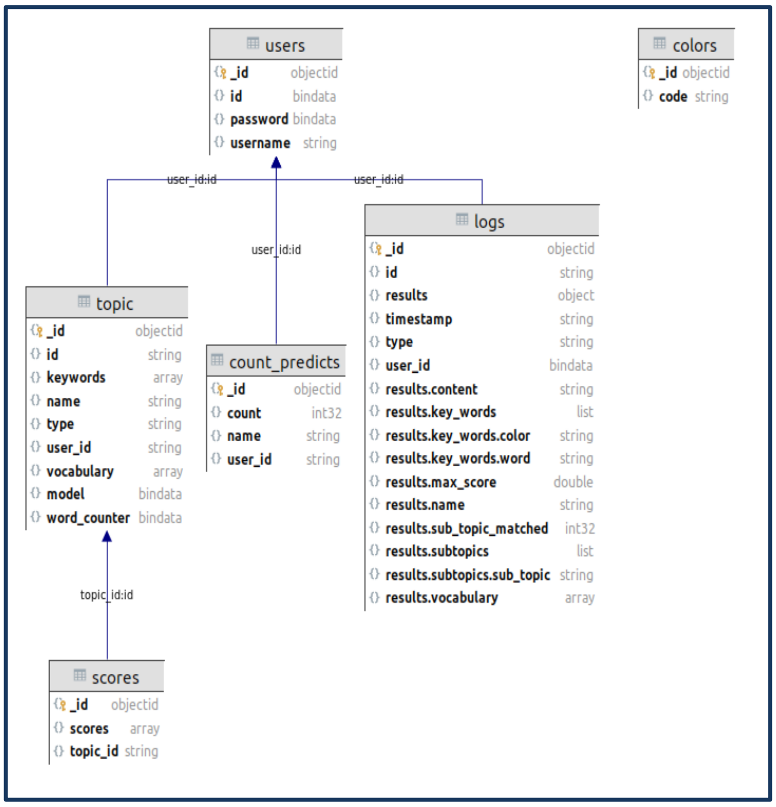


Fig. 4.8. Database Schema of Web Application

Taking into consideration that **LDA** models need to be stored and the format is binary, a database document oriented is required to be use, due to less overhead. So, the choice for **TOPICO** is **MongoDB**. If the *schema* is empty, the tables will be created with default data. The database setup is created using a *docker container* with *mongo* and *mongo-express*. At first, when the server runs and the database is empty, these tables are created (with the relations described in Fig. 4.8):

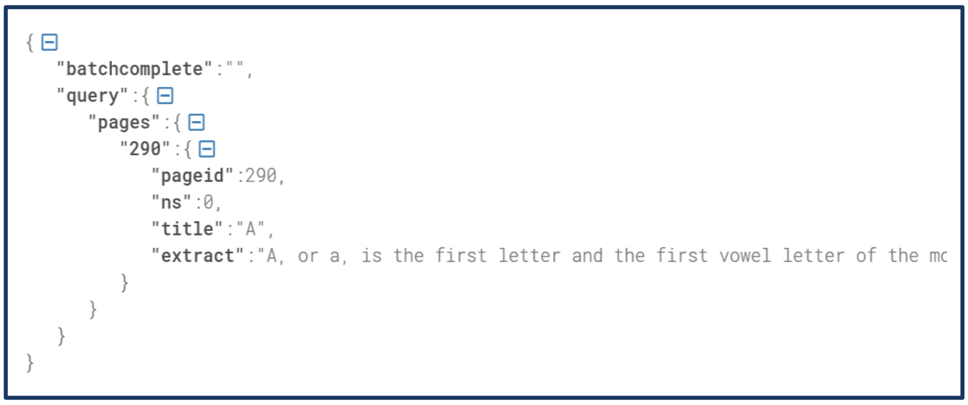
- **topic** - topic tables initialized with five generic topic models;
- **colors** - color code use to highlight relevant words after prediction;
- **count_predicts** - counters for generic topics;
- **logs** - history with all predictions done by a user;
- **scores** - performance score calculated when a model is created;
- **users** - users created.

For a better understanding, in this section the file structure of the project is described with explanations for every package and why every solution was chosen in the final version of **TOPICO**.

- ***Data_gen***

To run the application, default data is required which will be used for further tasks. In this package, the subpackages *history*, *most_generic*, *music*, *painting*, *sports* represent the directories that contain articles from *Wikipedia*, content that will be processed for the initial **LDA** models. The usage of the models is explained in more detail in the further sections.

Data_get and *data_parse*, are the data getters and parsers used in the application. In the *wikipedia.py*, found in the *wikipedia* subpackage is implemented the necessary bot in order to obtain articles. Firstly, the algorithm checks if the desired path for the articles is created. If not, it will be created. The endpoint called is https://en.wikipedia.org/w/api.php?format=json&action=query&prop=extracts&exintro&explaintext&redirects=1&titles=, being appended to this link sequentially all the articles names that need to have their content extracted. The result will be in **JSON** format, as presented in Fig. 4.9.



```
{
   "batchcomplete":"",
   "query":{
      "pages":{
         "290":{
            "pageid":290,
            "ns":0,
            "title":"A",
            "extract":"A, or a, is the first letter and the first vowel letter of the mc
         }
      }
   }
}
```

Fig. 4.9. *Wikipedia* endpoint response format

In the *dataNLTK.py* and *dataYTP.py*, two data providers are implemented, one for **NLTK** and one for *Youtube*. The data obtained from the first one was only used for testing purposes, not being included in the final version of the application and the other, was used in the *Youtube Transcript* component, which is described in more details in the following sections.

- ***Prediction***

This package contains the implementation of the **TOPICO** algorithm, as described in the pseudocode illustrated in Fig. 4.10. The code combines all the other packages, being the main component of the entire thesis.

```
    def topico(file, topics):
            # Filter Data
            data = parse_data(file)

            # Calculate occurrences and find keywords
            c = CountVectorizer()
            occurrences = c.fit_transform(data)
            key_words = find_key_words(occurrences)

            # Find Main topic using keywords
            main_topic =
find_main_suitable_topic(topics, key_words)

            # Make the prediction
            results =
get_lda_model_and_predict(main_topic, data)
            return results
```

Fig. 4.10. Pseudocode of the **TOPICO** algorithm

- ***Topico_alg***

To have the needed flexibility when it comes to much more accurate predictions, models need to be trained, being the main objective of the package. In *lda.py*, the method *create_lda_model* will offer the necessary environment to create all the needed models. The steps of this algorithm are the following:

- ***Step 1*** - get the data from the file and filter;
- ***Step 2*** - create the vocabulary eliminating the duplicate words;
- ***Step 3*** - compute the count of the vocabulary using *CountVectorizer* model;
- ***Step 4*** – extract the top 30 key words from the given text (needed for word similarity algorithm);
- **Step 5** - create the **LDA** model.

In this state, for every model created a performance factor named *coherence* is calculated, but the entire process is more detailed in the next sections. The results obtained from the computation are: **LDA** model, *CountVectorizer* model, vocabulary, performance scores and the top 30 key words. Another used method is *get_lda_model_and_predict*, which retrieves the **LDA** and *CountVectorizer* models and computes the prediction in order to match the parsed content to the more relevant topic.

The file tf_idf.py refers to subsection 4.1.2 about word similarity, but as presented in the final implementation of the application, it was not included due to performance issues.

- ***Topico_api***

The main feature of the entire project is the **TOPICO** algorithm, but to provide a better understanding of how it works, a web page is provided which includes the algorithm. This package represents the backend of the web page and has the following structure:

- ***build*** - compiled version of the UI;
- ***login*** - login service;
- ***models*** - models used to communicate with the database;
- ***pages*** - controllers for every web page;
- ***repo*** - repositories for tables of the database;
- ***type*** - enum to describe types.

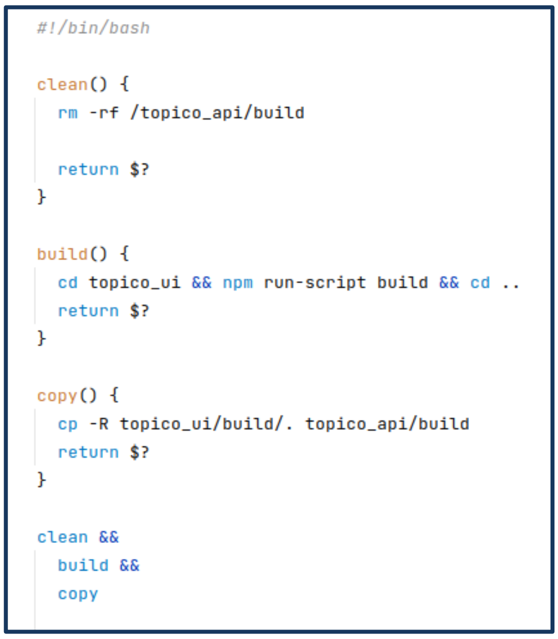
- ***Topico_ui***

The UI of the web page is developed using *ReactJS*, using *hooks functions*. The starting is the *index.html*, the requests on the page being routed to the mapped components. If the requested route does not exist, the returned error code is *404 Not Found*. The structure of the package is the following:

- ***component*** - the main components used in the application (ex: *YouTube Transcript*, *File Upload*, *Statistic and Logs Graph* etc.);
- ***http*** - client which handles all the GET and POST requests from the client side of the application - the requests are made using *XMLHttpRequest* based on *Promises*;
- ***model*** - models for parsing the results from the server in **JSON** format;
- ***static*** - CSS classes and the navbar of the application;
- ***view*** - component for every view of the application.

To persist the data on the client until the moment it is sent to the server or to save variables in various functions, the approach used is *Hooks functions*, which allows defining a state for every function created. The reason behind using this method is to avoid *boilerplate* code and unnecessary high complexity for small business logic.

Taking into consideration that we also serve the frontend and the backend on the port 5000, a *build* of the *UI* is necessary when the application needs to be deployed, because *ReactJS* needs to be translated into **HTML&CSS** and *Javascript*. To acquire this result, before a deployment, the script *local-build.sh* (presented in Fig 4.11) needs to be called.



```
#!/bin/bash

clean() {
  rm -rf /topico_api/build

  return $?
}

build() {
  cd topico_ui && npm run-script build && cd ..
  return $?
}

copy() {
  cp -R topico_ui/build/. topico_api/build
  return $?
}

clean &&
  build &&
  copy
```

Fig. 4.11. The *local-build.sh* script

4.2.2 Login Page / Register Page

If a user does not have an account, it needs to be created. The only credentials needed are an *username* and a *password*. An account is required because **TOPICO** *(Web page)* provides the possibility of training custom **LDA** models that will be used for further predictions.

4.2.3. Main Page

In order to optimize the prediction process, an interface with three types of inputs is provided. In this view, a user can provide different types of data, in order to obtain the best match with its subtopics (as shown in Fig. 4.12) There are three types of input accepted: *Youtube Transcript* (get the transcript from an YouTube video), *Text Input* (raw text) and *Drag and Drop (a file* - file filled with text).

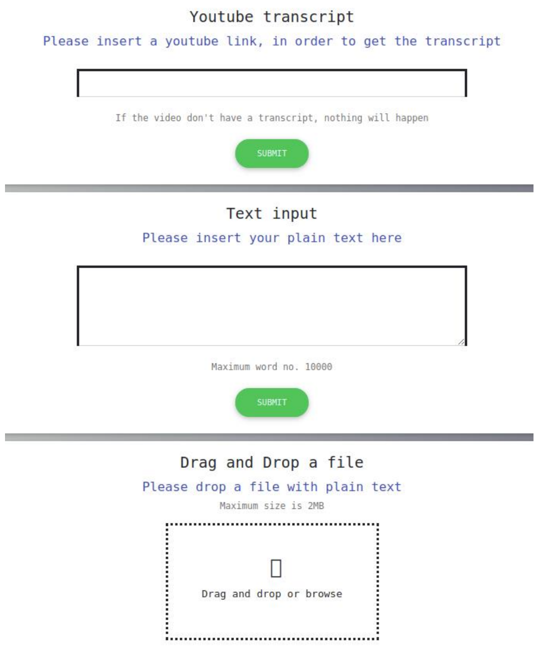

Youtube transcript
Please insert a youtube link, in order to get the transcript
If the video don't have a transcript, nothing will happen
SUBMIT
Text input
Please insert your plain text here
Maximum word no. 10000
SUBMIT
Drag and Drop a file
Please drop a file with plain text
Maximum size is 2MB
Drag and drop or browse

Fig. 4.12. Input components from the *Main Page*

- *YouTube Transcript*

As presented in [39], on YouTube there are over 30 million watchers a day, so the ability to analyze the videos that we consume is very useful, especially when a user needs a short summary about a large one.

YouTube transcript component is a bot created using *Selenium*, that is used to crawl YouTube links and obtain the transcript from it. If a video does not contain such an element, the component will do nothing.

As the UI code of every web page is public, what Selenium does is to search for **HTML** elements using descriptors. These descriptors can be: **CSS** classes, Ids, inner components (first child of a *div*), etc. Taking into consideration that every YouTube video has the same layout, if a transcript is present, it can be extracted using the same selectors, repeatedly, but there is one issue. The **HTML** or **CSS** code can be slightly modified by the developers of an application, affecting the selectors used for extracting the data. This can be resolved by constant maintenance, looking for new valid selectors when needed.

To start extracting the data, a chrome page is started in the background with the provided link. To not affect the user experience, the following options are provided to Selenium's core:

- **options.headless = True** - the chrome page will *not* open a window;
- **mute-audio** - no sound will start, when an audio input is present;
- **disable-dev-shm-usage** - optimization, prevent crashing.

After *Selenium* is started with the desired link and options, the search process starts. A human user would open the link and click some buttons to gather the needed information. Almost the same happens with this *Selenium* bot but at a higher speed, selecting the elements using the provided selectors. To simulate human behavior, *time waits* are inserted between every action done. This is necessary, because some bots can be blocked if the number of requests per second is higher than a human can do.

After the text is extracted and filtered from the YouTube timestamps, the text is ready to be provided to **TOPICO**.

- ***Text Input / Drag and Drop a file***

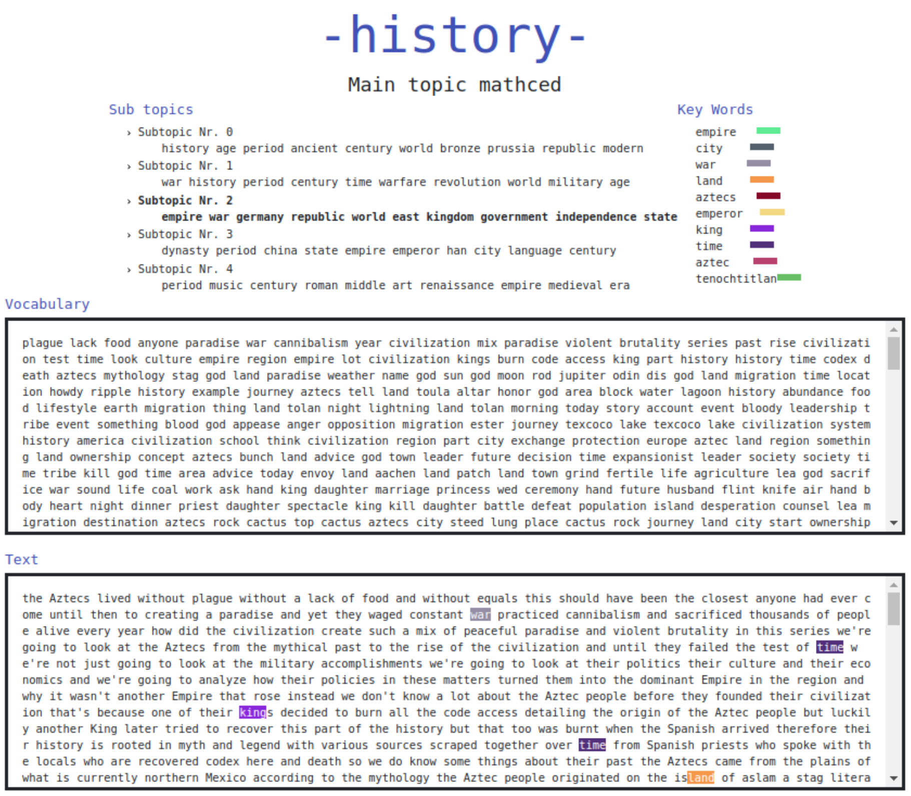
If an user wants to categorize unsupervised data, these two components provide the necessary infrastructure. Due to performance, for the text input component, the maximum suggested number of words is 10000 and for the file component, the maximum suggested size is 2MB.

To obtain better performance on large amounts of data, the input that is going to be sent is saved in a temporary file. What **TOPICO** receive, is a file object which it will be sent next to the filtering process with the option ***fl***.

After **TOPICO** finishes the execution, every component will redirect to the *Result page* (presented in Fig. 4.13), where the metrics from the prediction will be displayed. The headline of this view represents the name of the main topic followed by the subtopics. The bolded one is the most suitable. For an easier interpretation of the final results, key words, vocabulary and content are also displayed. The input needed for this view is a **JSON** formed from the following information:

- **max_score** - max score of matched subtopic;
- **sub_topic_matched** - id of the matched subtopic**;**
- **subtopics** - the subtopics of the main topic;
- **name** - name of the main topic;
- **content** - the text which was predicted;
- **vocabulary** - vocabulary present in the *content;*
- **key_words** - top ten keywords with an associated color.

Furthermore, after every prediction, in the database, logs with the results are saved for future reviews and the counter for the matched topic is incremented. In the next sections, these metrics will be more detailed.


-history-

Main topic mathced

Sub topics

› Subtopic Nr. 0
history age period ancient century world bronze prussia republic modern

› Subtopic Nr. 1
war history period century time warfare revolution world military age

› **Subtopic Nr. 2**
empire war germany republic world east kingdom government independence state

› Subtopic Nr. 3
dynasty period china state empire emperor han city language century

› Subtopic Nr. 4
period music century roman middle art renaissance empire medieval era

Key Words

empire	🟩
city	⬛
war	⬜
land	🟧
aztecs	🟥
emperor	🟨
king	🟪
time	🟪
aztec	🟥
tenochtitlan	🟩

Vocabulary

```
plague lack food anyone paradise war cannibalism year civilization mix paradise violent brutality series past rise civilizati
on test time look culture empire region empire lot civilization kings burn code access king part history history time codex d
eath aztecs mythology stag god land paradise weather name god sun god moon rod jupiter odin dis god land migration time locat
ion howdy ripple history example journey aztecs tell land toula altar honor god area block water lagoon history abundance foo
d lifestyle earth migration thing land tolan night lightning land tolan morning today story account event bloody leadership t
ribe event something blood god appease anger opposition migration ester journey texcoco lake texcoco lake civilization system
history america civilization school think civilization region part city exchange protection europe aztec land region somethin
g land ownership concept aztecs bunch land advice god town leader future decision time expansionist leader society society ti
me tribe kill god time area advice today envoy land aachen land patch land town grind fertile life agriculture lea god sacrif
ice war sound life coal work ask hand king daughter marriage princess wed ceremony hand future husband flint knife air hand b
ody heart night dinner priest daughter spectacle king kill daughter battle defeat population island desperation counsel lea m
igration destination aztecs rock cactus top cactus aztecs city steed lung place cactus rock journey land city start ownership
```

Text

```
the Aztecs lived without plague without a lack of food and without equals this should have been the closest anyone had ever c
ome until then to creating a paradise and yet they waged constant war practiced cannibalism and sacrificed thousands of peopl
e alive every year how did the civilization create such a mix of peaceful paradise and violent brutality in this series we're
going to look at the Aztecs from the mythical past to the rise of the civilization and until they failed the test of time w
e're not just going to look at the military accomplishments we're going to look at their politics their culture and their eco
nomics and we're going to analyze how their policies in these matters turned them into the dominant Empire in the region and
why it wasn't another Empire that rose instead we don't know a lot about the Aztec people before they founded their civilizat
ion that's because one of their kings decided to burn all the code access detailing the origin of the Aztec people but luckil
y another King later tried to recover this part of the history but that too was burnt when the Spanish arrived therefore thei
r history is rooted in myth and legend with various sources scraped together over time from Spanish priests who spoke with th
e locals who are recovered codex here and death so we do know some things about their past the Aztecs came from the plains of
what is currently northern Mexico according to the mythology the Aztec people originated on the island of aslam a stag litera
```

Fig. 4.13. Example of the *Result page* after a prediction

4.2.4. Topics Page

As presented in the previous section, the main feature of **TOPICO** is the fact that every user can create their personal **LDA** models with their custom data. In this view, every user can visualize the models with their subtopics, create another model providing the necessary data (as shown in Fig 4.14) and view statistics about the topic/prediction, as can observe in Fig. 4.15.

There are three types of models: *MOST_GENERIC*, *GENERIC* and *USER_CREATED*. The *GENERIC* models are created at the first run of the server and are visible to all the users while the *USER_CREATED* models are custom and visible only to the user that created them. The *MOST_GENERIC* model is created also at the first run of the server but is not taken into consideration for prediction. The data provided for this model is a combination of the data provided for all the *GENERIC* models. It was created due to performance comparison, to make it easier for the user to see the advantages of using multiple models with data more centered around only one subject, instead of only using one model trained with data from multiple subjects.

A major issue in creating a model is the database chosen for training. If the database is not properly chosen, the results will not be as good as expected. At first, the database chosen was a bunch of old literature books, because it was considered that all the important words are present. Unexpectedly, the results were poor, because the content of every book was not centered enough around multiple subjects. Afterwards, for training a dataset containing amazon reviews about sport articles, books, music and art was also used. Again, the performance was low, because the reviews were not so concluded to the subjects presented previously. Most of the reviews were texts like: “I enjoyed this product”, “This is a bad product”, “The best”, etc, which describes the products purchased using mostly adjectives, instead of providing relevant information to a subject.

For this implementation, the GENERIC models created contain information about: *Music*, *Painting*, *Sport* and *History*. The dataset used for these **LDA** models are articles from *Wikipedia* centered around the subjects suggested by their names. For training, this is one of the best options, because most of the posts from *Wikipedia* are objective and descriptive most of the time. The articles were obtained via calling a *Wikipedia* endpoint (as described in the *Data_gen* section). The input for this bot is a list with all the articles names that need to be extracted and the output will be a directory with files containing the content. If a name is invalid, the execution of the bot will not be stopped, but no file will be created.

If a user wants to create a custom model, the data necessary should be carefully chosen. Even though **TOPICO** offers more flexibility when it comes to **LDA** to obtain the best results there should be created as many models as possible. For example, if a user has data about *tennis*, *basketball*, *skiing* and *hockey*, a better approach will be to create two models, one named *winter-sports* and the other named *sports-using-ball*, then to create one *GENERIC* named *sport*. The creation is done by providing either a file or a .zip file containing multiple documents and a name.

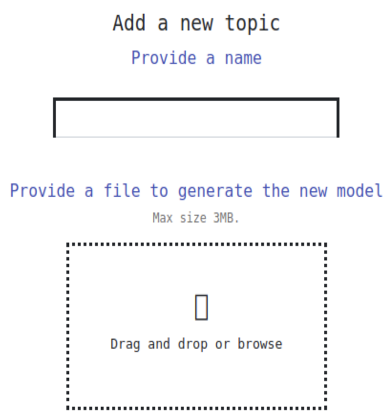


Fig. 4.14. Add a new topic

To monitor if a model is stable and relevant, in the topic view a graph is presented which shows how many times a model was chosen for prediction. (as shown in Fig. 4.15). There are some cases when the key words for a specific model are chosen poorly, so it will affect the matches made in the word similarity phase. If this happens, the concerned model can be easily identified, if there are more matches than expected.

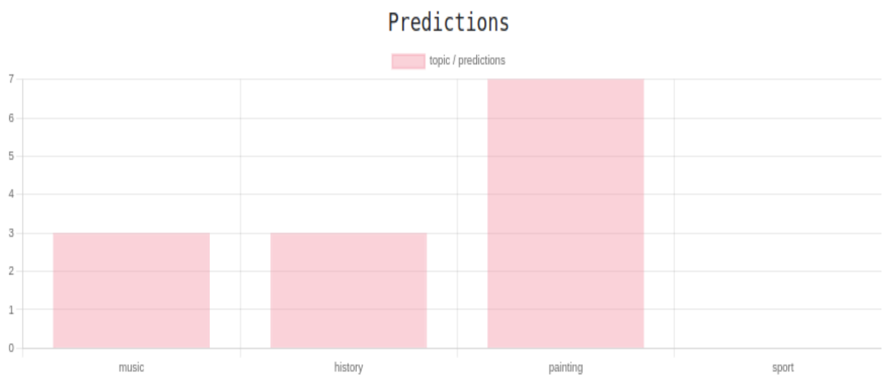


Fig. 4.15. Topic/Prediction graph

4.2.5 Statistics & Logs Page

The best way to create better models is to analyze the already created ones and decide which improvements are needed. In this view, a user can see the *coherence factor* calculated for every model created by himself and for the *GENERIC* ones, revise every prediction made in the past and see the *MOST_GENERIC* model created using all the data used for the *GENERIC* models.

When a new model is created, every subtopic generated needs to be analyzed and determine how relevant it is from a human perception. For topics, the *score* determines whether a topic is relevant. There are many factors that can be calculated to establish the performance and the score of a model, but the one that brings out how closer a topic can be to the human meaning of the words is the *coherence factor*. What coherence does is to check the *semantic similarity* between all the keywords from a topic. As a higher score is obtained, the topic is more relevant to the model. The coherence factor is used to find the optimal number of topics inside a model. There are multiple measures for the coherence factor, but the one chosen for **TOPICO** is *u_mass***.** The *u_mass* measure takes into consideration all the occurrences of the words in document using the following formula:

$$score_{U_{mass}}(w_i, w_j) = log(\frac{D(w_i,w_j) + 1}{D(w_i)}) \tag{1}$$

where (w_i, w_j) represents words from a topic that need to be analyze and $D(w_i, w_j)$represents all the occurences of word w_i and w_j together in a document.

This result is divided to the number of occurrences of word w_i, because two words that appear together multiple times can be more relevant to a subject than only one word. As the numerator is smaller than the denominator, the $score_{U_{mass}}$function returns negative numbers. So, the closer the obtained value is to zero, the better the topic model is. Although the coherence factor is a useful metric to determine how good a model is, there is one issue. **LDA** models are probabilistic, which means that every time the coherence factor is calculated, we will obtain another value. In **TOPICO**, when a model is created, the *coherence factor* is calculated multiple times with different numbers of topics. In this way, an evolution of the performance can be observed. From multiple examples, the conclusion is that the lower the number of subtopics is, the more relevant they will be. The number chosen by the author in the application is five.

As mentioned previously, there are five iterations when a model is created (as shown in Fig. 4.16). Every iteration computes the coherence factor for a model with subtopics ranging from 2 to 20 topics. Using *sklearn* for the creation of the models, *tmtoolkit* was used to compute the factor. The inputs needed for the framework are the *CountVectorizer* model pre-trained with the corpus, vocabulary, corpus and the **LDA** model components resulting from the training.

The Statistics & Logs Page has two other components that will help every user to determine whether his models or the *GENERICS* are well trained. Every prediction made can be revised and compared with the *MOST_GENERIC* model.

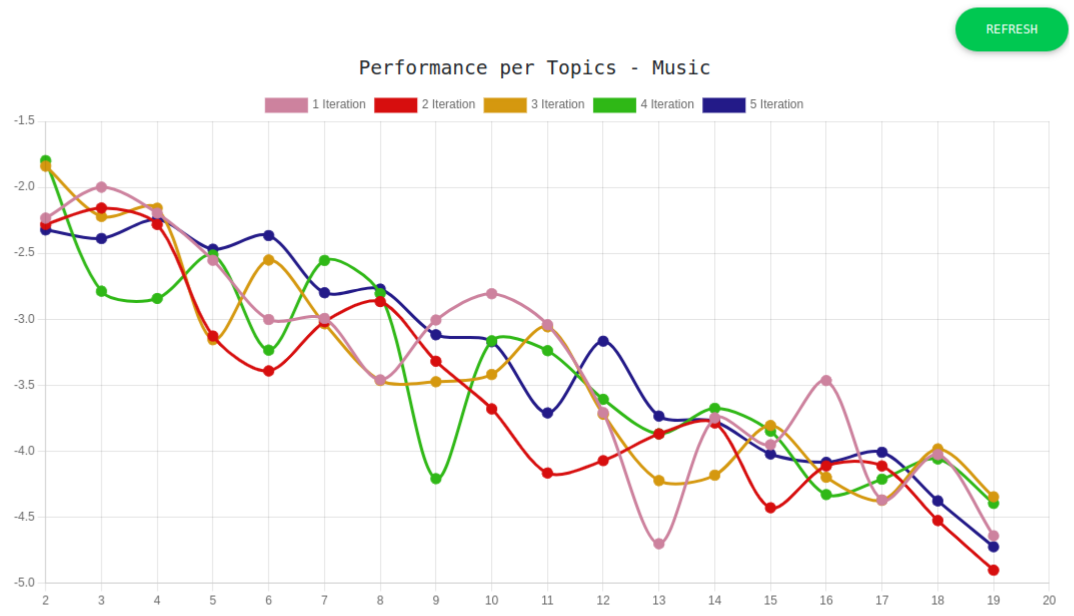


Fig. 4.16. Performance graph for Model about Music

5. Experimental Results

One of the major issues encountered was finding a good database to create the *GENERIC* models. On the internet, most of the datasets are centered around supervised learning, more than around unsupervised learning. At first a lot of datasets from www.kagle.com, which is a very reliable source, were taken into consideration, but were not useful. Some examples of datasets that are found on this website are: *Heart Attack Analysis* & *Prediction Datasetset*, *Reddit Vaccine Myths*, *Water Quality* etc.

For experimental proposes, there were taken into consideration two datasets, to illustrate the differences. One is constructed from almost 40,000 reviews from amazon and the other one contains five categories of articles extracted from *Wikipedia*. The results outline the differences between these datasets through the relevance each one of them provide.

As presented in the previous section, as training data was used Amazon Reviews for products from different categories. In the following figures is presented the subtopics obtained from this data set and the actual one used (*Wikipedia* articles).

The main idea of the project is to obtain generic models well trained on a specific subject, being as general as possible. For example, a good model about *music* would be a model which has the following subtopics:

- **subtopic about instruments**;
- **subtopic about rock music**;
- **subtopic about bands**.

Of course, the previous example is relative from user to user, but the objective is achieved when it is covered as much as possible from a specific subject. Comparing Fig 5.1.a with Fig.5.1.b, there is a big difference between the models trained with different databases. Even though the dataset was large, the generated subtopics of the *Amazon model* are not concludent, and a user can't review and center them around a subject.

Subtopic Nr. 0	Subtopic Nr. 0
chord music seventh opera bass root note dominant form triad	guitars strap guitar capo use work fine time price product
Subtopic Nr. 1	Subtopic Nr. 1
music rhythm drum notation sound folk rock percussion time kit	tuner guitar sound use case pick string bass tune love
Subtopic Nr. 2	Subtopic Nr. 2
music century piano term movement instrument baroque bass violin oboe	mic use power need distortion clip gain thing say turn
Subtopic Nr. 3	Subtopic Nr. 3
music cue rock chant rest include song English century songs	guitar price quality set want sound time cable use try
Subtopic Nr. 4	Subtopic Nr. 4
minor scale music note tone interval octave mode pitch tonic	pedal amp tone sound guitar stand acoustic electric volume
(a)	(b)

Fig. 5.1. Subtopics generated **Music mode**: a - *Wikipedia* data, b - *Amazon Reviews.*

As it's presented, most of the subtopics are centered around the same words (*guitar*, *tone, price* etc.). Having all the subtopics that close one to another, will not lead to a good categorization. Although, the model obtained from the *Wikipedia* articles has much better results. For example, subtopic 2 contains keywords like *piano*, *century*, *baroque* which will lead a user to *classical music*. Also, subtopic 1 contains keywords like *rhythm*, *drum*, *sound percussion*, *time* which are centered around *music theory.* Furthermore, the same situation can be easily outlined in the case of the sports models. In Fig. 5.2.b the keywords relevant for the subtopics are words like *Christmas*, *knife*, *son*, *scooter*, which does not give so much information. The reviews are mostly about sport articles bought on amazon, so the results were even worse than the music model trained with the amazon reviews. On the other hand, the sports model obtained from the *Wikipedia* articles resulted in relevant topics. In Fig 5.2.a the subtopic 1 contains words like *hockey, ball, team, lacrosse, ice, polo* which indicates different types of sports. Of course, the models are not perfect and there is always room for improvement. There are two key factors to obtain good models: big databases and relevant information about the desired topic.

Subtopic Nr. 0
sport rifle rally world cross country track raid ice association
Subtopic Nr. 1
sport game world hockey ball team lacrosse use ice polo
Subtopic Nr. 2
game ball sport team ski drag hockey field boat world
Subtopic Nr. 3
hockey ice dance sport roller inline ball game speed figure
Subtopic Nr. 4
sport game ball competition target thumb range course rock type

(a)

Subtopic Nr. 0
use Christmas handle perfect plastic item time easy store ring
Subtopic Nr. 1
set knife buy frisbee sheath wheel blade hit carry deal
Subtopic Nr. 2
price son quality nice gift water think excellent know birthday
Subtopic Nr. 3
scooter love year daughter product fun ride time bike grand
Subtopic Nr. 4
easy use pretty play razor product box light awesome

(b)

Fig. 5.2. Subtopics generated **Sport mode**: a - *Wikipedia* data, b - *Amazon Reviews.*

The dataset issue persists also when it comes to the training process. As presented in the previous section, for a better understanding of differences between models trained with data centered around one subject and more general data, a model with all the data used for *GENERIC* models was also created.

Fig 5.3.b illustrates the model that was created using all the articles from the following topics: *music*, *sports, history* and *painting*. As expected, the subtopics are relevant to the data provided for training but are too general to obtain accurate results. For example, subtopic 2 contains keywords like: *art, history, war*, words that refer to history and a little bit art, but it is not known for sure which branch is of interest. In subtopic 3, there are keywords like *guitar*, *harmonic*, and *sound* which refers to music, but again the branch of interest is unclear. As presented previously, the music model obtained using *Wikipedia* articles is better divided in different branches from the music (*classical music* and *music theory*).

Subtopic Nr. 0
chord music seventh opera bass root note dominant form triad
Subtopic Nr. 1
music rhythm drum notation sound folk rock percussion time kit
Subtopic Nr. 2
music century piano term movement instrument baroque bass violin oboe
Subtopic Nr. 3
music cue rock chant rest include song english century songs
Subtopic Nr. 4
minor scale music note tone interval octave mode pitch tonic

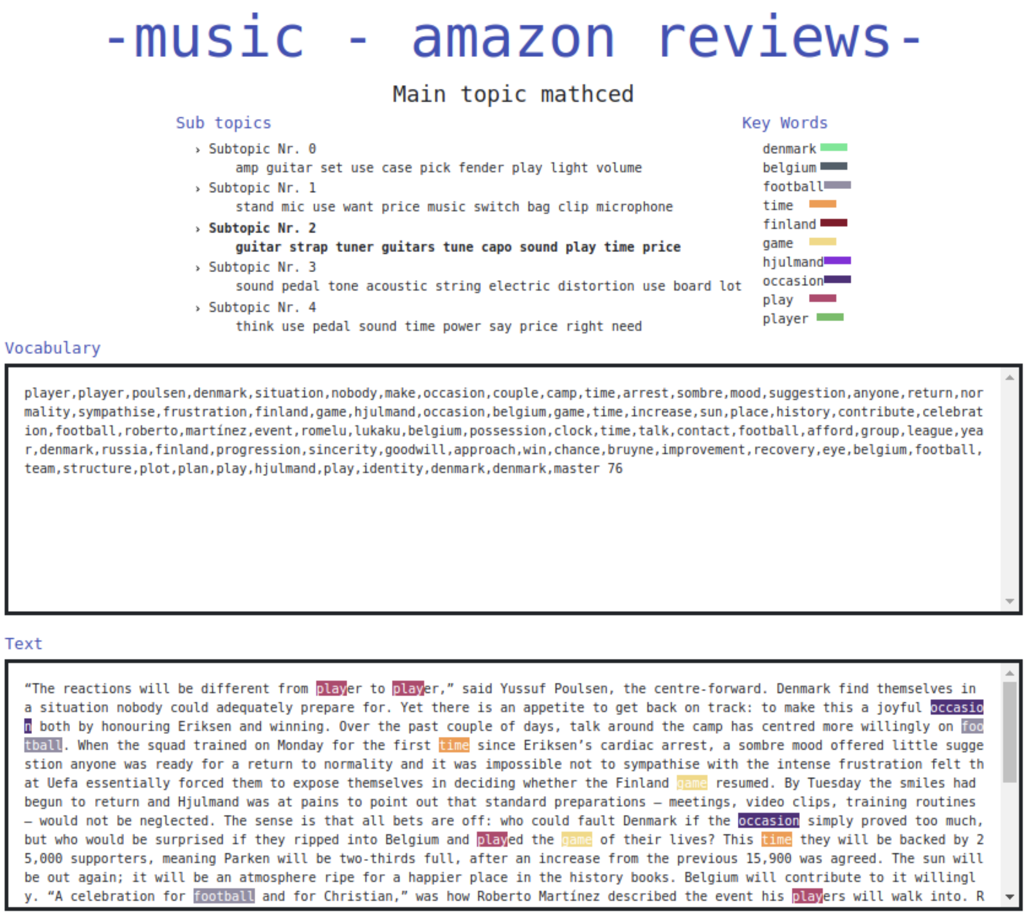
(a)

Subtopic Nr. 0
music time century style greek notation rhythm modern form
Subtopic Nr. 1
work form music style melody movement technique paint
Subtopic Nr. 2
art century history period empire world war europe germany
Subtopic Nr. 3
guitar price quality set want sound time cable use harmonic
Subtopic Nr. 4
music century instrument opera term sound piano bass string

(b)

Fig. 5.3. Subtopics generated: a – Music model (*Wikipedia* data), b – *MOST_GENERIC* model.

Choosing the best dataset for prediction, affects not only how relevant the subtopics will be, but also the prediction itself. Taking into consideration that the model is chosen based on the keywords selected from the training data, it is crucial that the dataset needs to be as descriptive as possible. The confirmation can be observed in Fig. 5.4, which represents the following experiment: for the same text, a prediction was made using models created from amazon reviews and models created from *Wikipedia* articles. The models created in both cases were about *Music* and *Sport.* The text chosen for the experiment was a short article about Christian Eriksen and the situation of Denmark at Euro 2020. In Fig 5.4.a the algorithm chose the *Music* model for the final prediction instead of the *Sport* model, because the keywords selected at the training phase were not relevant to sports. On the other hand, in Fig 5.4.b the chosen model was the right one, the model being trained with articles about sports and the keywords being relevant to the subject.



-music - amazon reviews-

Main topic mathced

Sub topics

› Subtopic Nr. 0
  amp guitar set use case pick fender play light volume
› Subtopic Nr. 1
  stand mic use want price music switch bag clip microphone
› **Subtopic Nr. 2**
  guitar strap tuner guitars tune capo sound play time price
› Subtopic Nr. 3
  sound pedal tone acoustic string electric distortion use board lot
› Subtopic Nr. 4
  think use pedal sound time power say price right need

Key Words

denmark	🟩
belgium	⬛
football	⬜
time	🟧
finland	🟥
game	🟨
hjulmand	🟪
occasion	🟪
play	🟥
player	🟩

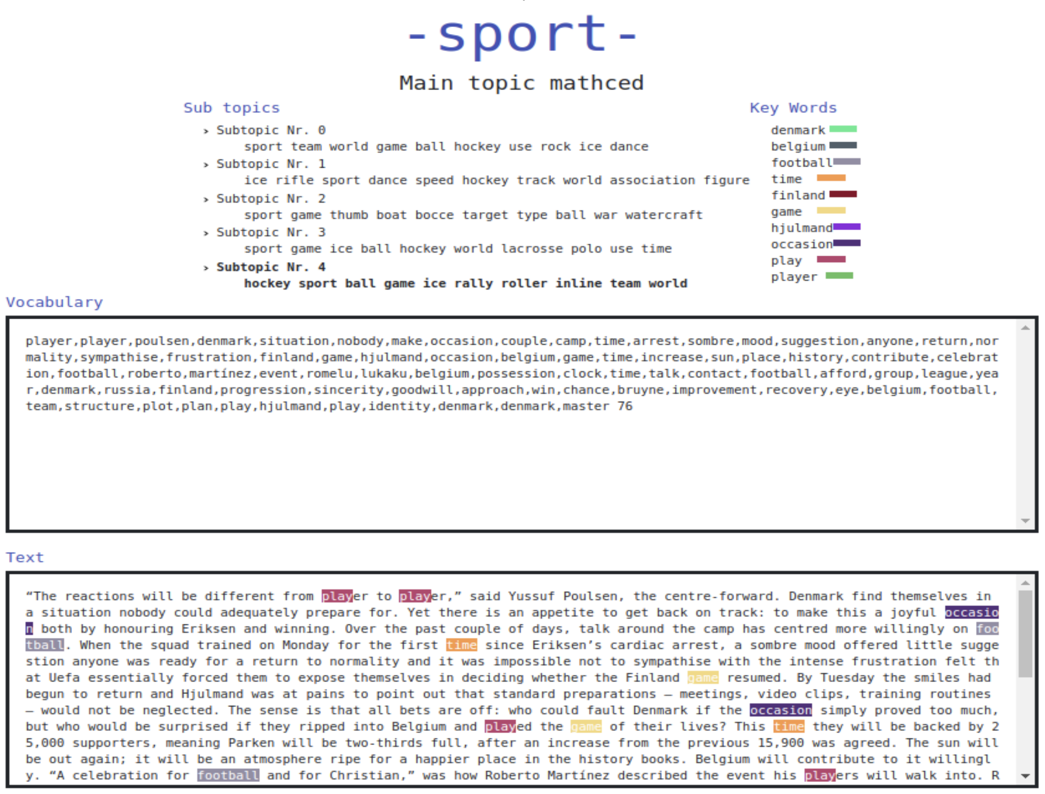
Vocabulary

```
player,player,poulsen,denmark,situation,nobody,make,occasion,couple,camp,time,arrest,sombre,mood,suggestion,anyone,return,nor
mality,sympathise,frustration,finland,game,hjulmand,occasion,belgium,game,time,increase,sun,place,history,contribute,celebrat
ion,football,roberto,martínez,event,romelu,lukaku,belgium,possession,clock,time,talk,contact,football,afford,group,league,yea
r,denmark,russia,finland,progression,sincerity,goodwill,approach,win,chance,bruyne,improvement,recovery,eye,belgium,football,
team,structure,plot,plan,play,hjulmand,play,identity,denmark,denmark,master 76
```

Text

```
“The reactions will be different from player to player,” said Yussuf Poulsen, the centre-forward. Denmark find themselves in
a situation nobody could adequately prepare for. Yet there is an appetite to get back on track: to make this a joyful occasio
n both by honouring Eriksen and winning. Over the past couple of days, talk around the camp has centred more willingly on foo
tball. When the squad trained on Monday for the first time since Eriksen’s cardiac arrest, a sombre mood offered little sugge
stion anyone was ready for a return to normality and it was impossible not to sympathise with the intense frustration felt th
at Uefa essentially forced them to expose themselves in deciding whether the Finland game resumed. By Tuesday the smiles had
begun to return and Hjulmand was at pains to point out that standard preparations – meetings, video clips, training routines
– would not be neglected. The sense is that all bets are off: who could fault Denmark if the occasion simply proved too much,
but who would be surprised if they ripped into Belgium and played the game of their lives? This time they will be backed by 2
5,000 supporters, meaning Parken will be two-thirds full, after an increase from the previous 15,900 was agreed. The sun will
be out again; it will be an atmosphere ripe for a happier place in the history books. Belgium will contribute to it willingl
y. “A celebration for football and for Christian,” was how Roberto Martínez described the event his players will walk into. R
```

a)



-sport-

Main topic mathced

Sub topics

› Subtopic Nr. 0
  sport team world game ball hockey use rock ice dance
› Subtopic Nr. 1
  ice rifle sport dance speed hockey track world association figure
› Subtopic Nr. 2
  sport game thumb boat bocce target type ball war watercraft
› Subtopic Nr. 3
  sport game ice ball hockey world lacrosse polo use time
› **Subtopic Nr. 4**
  hockey sport ball game ice rally roller inline team world

Key Words

denmark	🟩
belgium	⬛
football	⬜
time	🟧
finland	🟥
game	🟨
hjulmand	🟪
occasion	🟪
play	🟥
player	🟩

Vocabulary

```
player,player,poulsen,denmark,situation,nobody,make,occasion,couple,camp,time,arrest,sombre,mood,suggestion,anyone,return,nor
mality,sympathise,frustration,finland,game,hjulmand,occasion,belgium,game,time,increase,sun,place,history,contribute,celebrat
ion,football,roberto,martínez,event,romelu,lukaku,belgium,possession,clock,time,talk,contact,football,afford,group,league,yea
r,denmark,russia,finland,progression,sincerity,goodwill,approach,win,chance,bruyne,improvement,recovery,eye,belgium,football,
team,structure,plot,plan,play,hjulmand,play,identity,denmark,denmark,master 76
```

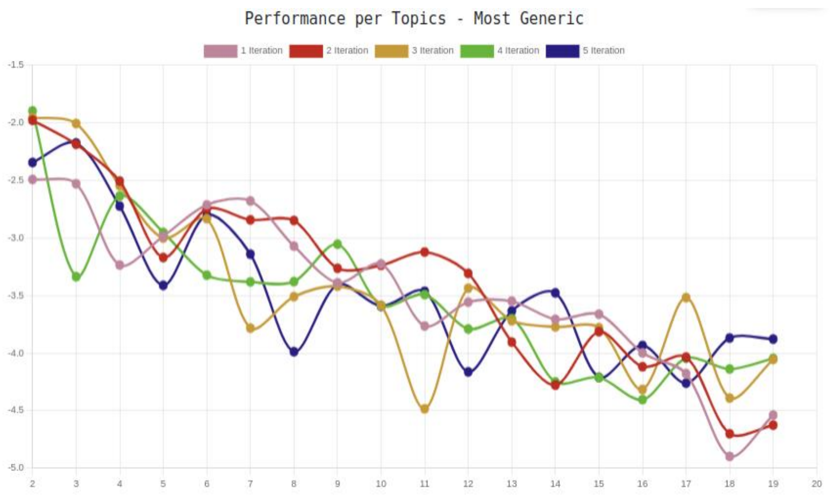
Text

```
“The reactions will be different from player to player,” said Yussuf Poulsen, the centre-forward. Denmark find themselves in
a situation nobody could adequately prepare for. Yet there is an appetite to get back on track: to make this a joyful occasio
n both by honouring Eriksen and winning. Over the past couple of days, talk around the camp has centred more willingly on foo
tball. When the squad trained on Monday for the first time since Eriksen’s cardiac arrest, a sombre mood offered little sugge
stion anyone was ready for a return to normality and it was impossible not to sympathise with the intense frustration felt th
at Uefa essentially forced them to expose themselves in deciding whether the Finland game resumed. By Tuesday the smiles had
begun to return and Hjulmand was at pains to point out that standard preparations – meetings, video clips, training routines
– would not be neglected. The sense is that all bets are off: who could fault Denmark if the occasion simply proved too much,
but who would be surprised if they ripped into Belgium and played the game of their lives? This time they will be backed by 2
5,000 supporters, meaning Parken will be two-thirds full, after an increase from the previous 15,900 was agreed. The sun will
be out again; it will be an atmosphere ripe for a happier place in the history books. Belgium will contribute to it willingl
y. “A celebration for football and for Christian,” was how Roberto Martínez described the event his players will walk into. R
```

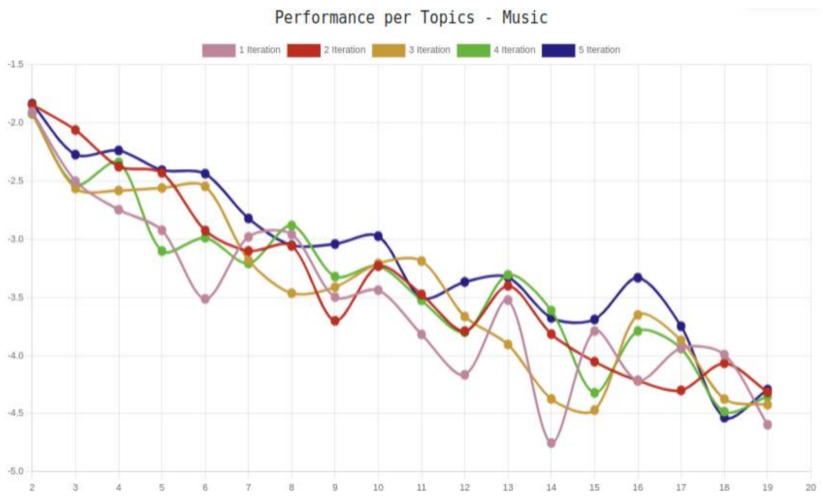
b)

Fig. 5.4. Prediction results comparison between *Wikipedia dataset* vs *amazon reviews dataset*

A more centered model around a subject will have better performance than a model trained with data from multiple domains, from the point of view of the generated topics. In Fig 5.5 can be observed again why a centered model around a subject is better, but this time from the point of view of the *coherence factor*.



a)



b)

Fig 5.5 Coherence factor comparison between *MOST_GENERIC model (a)* and *music model (b)*

A model has good performance, when the coherence factor calculated with *u_mass* measure, returns a negative number as close as possible to zero. For this experiment, like it was presented in the previous sections, multiple iterations were performed, because the coherence factor is a probabilistic measure which means that every time another score is obtained. An iteration represents calculating the coherence factor for a model with topics ranging from 2 to 20. In Fig. 5.5.a is presented a graph of scores calculated for the *MOST_GENERIC* model. In most of the iterations, a lot of spikes that are getting further from zero, can be observed. A higher negative number indicates that the subtopics generated are not relevant from a human point of view. On the other hand, in Fig. 5.5.b, there are not so many spikes, so the *music model* represents a more stable one, with better performance. For smaller number of topics, the *coherence factor* obtained for the *music model* is better than the one obtained for the *most_generic* one*,* as shown in fig 5.5.

7. Conclusion / Further Work

Even though **TOPICO** can be considered a good optimization when it comes to categorizing unsupervised data, there are still improvements that can be done to make the most of it. A great example would be the system of choosing the keywords. Now, when a model is created, the relevant words chosen for identifying which model need to be chosen when a prediction is made are selected automatically from the *CountVectorizer* model. A better approach would be to give the possibility to every user to choose his own words that would be further included in the word similarity algorithm for identification.

At the moment, when the models are created, the parameters used for *fine tuning* are basic, because this project was more of an experiment than a real product. There are multiple possibilities to find the best combination of parameters that will give the model a better performance. A way to obtain this result, is to create a range of possible parameters that will be fed to a *GridSearchCv* model from *sklearn framework* which will run all the combinations of them. In this way, the final model will have much better accuracy in terms of topics.

Overall, the project main purpose was obtained and in some situations it is a better alternative to the classical *topic modelling* methods. Even though there are moments when predictions were not as accurate as expected, this experiment can be constantly improved to obtain better and better predictions. There are a lot of factors that influence the way the entire ecosystem works, but it is not so hard to handle all of them.

BIBLIOGRAPHY

[1] Li, Xin & Lei, Lei. (2019). *A bibliometric analysis of topic modelling studies (2000–2017)*, Journal of Information Science. 47. 016555151987704. 10.1177/0165551519877049 ;

[2] C.B. Asmussen, C. Møller, *Smart literature review: a practical topic modelling approach to exploratory literature review*. J Big Data 6, 93 (2019). https://doi.org/10.1186/s40537-019-0255-7;

[3] M. Allahyari and K. Kochut, *Automatic Topic Labeling Using Ontology-Based Topic Models*, 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 259-264, doi: 10.1109/ICMLA.2015.88;

[4] M. Morsey, J. Lehmann, S. Auer, AC. Ngonga (2011) *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*. In: Aroyo L. et al. (eds) The Semantic Web – ISWC 2011. ISWC 2011. Lecture Notes in Computer Science, vol 7031. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25073-6_29;

[5] K. Burke, *How Many Texts Do People Send Every Day*, [White Paper], https://www.textrequest.com/blog/how-many-texts-people-send-per-day/#:~:text=That%20gives%20us%20about%20277,33%2C834%20text%20messages%20per%20year ;

[6] J. Yeh, C. Lee, Y. Tan and L. Yu, *Topic model allocation of conversational dialogue records by Latent Dirichlet Allocation*, Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific, 2014, pp. 1-4, doi: 10.1109/APSIPA.2014.7041546;

[7] S. Tsuchiya, E. Yoshimura and H. Watabe, *An information arrangement technique for a text classification and summarization based on a summarization frame*, 2009 International Conference on Natural Language Processing and Knowledge Engineering, 2009, pp. 1-5, doi: 10.1109/NLPKE.2009.5313816;

[8] ***, *Neuro-linguistic programming*, [Wikipedia], https://en.wikipedia.org/wiki/Neuro-linguistic_programming ;

[9] ***, *Chinese room*, (2021, June 21 - *last modified*), [Wikipedia], https://en.wikipedia.org/wiki/Chinese_room;

[10] Elvis, *Deep Learning for NLP: An Overview of Recent Trends*, Aug 24, 2018, [White Paper], https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d ;

[11] Anna K., *The 12 most spoken languages in the world*, [White Paper], https://blog.busuu.com/most-spoken-languages-in-the-world/#:~:text=What%20are%20the%20world's%20most,a%20diverse%20and%20beautiful%20place ;

[12] D. Lopez Yse, *Your Guide to Natural Language Processing (NLP)*, [White Paper], https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1 ;

[13] H. Jabeen, *Stemming and Lemmatization in Python*, Oct. 23rd, 2018, [White Paper], https://www.datacamp.com/community/tutorials/stemming-lemmatization-python;

[14] ***, *Stemming*, [Wikipedia], https://en.wikipedia.org/wiki/Stemming ;

[15] M.F.Porter, An algorithm for suffix stripping, 1980, Originally published in \Program\, \14\ no. 3, pp 130-137, July 1980. (A few typos have been corrected.)

[16] Martin Porter, *The Porter Stemming Algorithm*, Jan 2006, https://tartarus.org/martin/PorterStemmer/;

[17] ***, *Lemmatization*, (2021, June 21 - *last modified*), [Wikipedia], https://en.wikipedia.org/wiki/Lemmatisation;

[18] Yassine Hamdaoui, *TF(Term Frequency)-IDF(Inverse Document Frequency) from scratch in python*, https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

[19] ***, *tf–idf*¸[Wikipedia], https://en.wikipedia.org/wiki/Tf%E2%80%93idf;

[20] ***, *Bag-of-words_model*,[Wikipedia], https://en.wikipedia.org/wiki/Bag-of-words_model#CBOW ;

[21] Hussain Mujtaba, *An Introduction to Bag of Words (BoW) | What is Bag of Words?*, [White Paper], https://www.mygreatlearning.com/blog/bag-of-words/ ;

[22] Al. Perrier, *Introduction to Natural Language Processing*, [White Paper], https://openclassrooms.com/en/courses/6532301-introduction-to-natural-language-processing/6980811-apply-a-simple-bag-of-words-approach ;

[23] ***, *Word2vec*, [White Paper], https://www.tensorflow.org/tutorials/text/word2vec,

[24] J. Boyd-Graber, *Understanding Word2Vec*, [YouTube], https://www.youtube.com/watch?v=QyrUentbkvw;

[25] ***, *Word2vec*, [Wikipedia], https://en.wikipedia.org/wiki/Word2vec#CBOW_and_skip_grams

[26] Swatimeena, *Training Word2vec using genism*, [White Paper], https://swatimeena989.medium.com/training-word2vec-using-gensim-14433890e8e4

[27] H. Gautam, *Word Embedding: Basic*, [White Paper], https://medium.com/@hari4om/word-embedding-d816f643140 ;

[28] ****, Two/Too Simple Adaptations of Word2Vec for Syntax Problems* - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-the-Skip-gram-and-Continuous-Bag-of-Word-CBOW-models_fig1_281812760 [accessed 26 Jun, 2021];

[29] The Semicolon, *Word2Vec - Skipgram and CBOW*, [Youtube], https://www.youtube.com/watch?v=UqRCEmrv1gQ ;

[30] B. MADHUKAR, The Continuous Bag Of Words (CBOW) Model in NLP – Hands-On Implementation With Codes, [White Paper], https://analyticsindiamag.com/the-continuous-bag-of-words-cbow-model-in-nlp-hands-on-implementation-with-codes/;

[31] D.M. Blei, A.Y. Ng, M.I. Jordan, *Latent Dirichlet Allocation,* Journal of Machine Learning Research 3 (2003) 993-1022, pp 993-1022, https://ai.stanford.edu/~ang/papers/jair03-lda.pdf;

[32] J. Moller-Mara, *Experiments with Latent Dirichlet Allocation*, [White Paper] https://mollermara.com/blog/lda/;

[33] L. Serrano, *Latent Dirichlet Allocation (Part 1 of 2)*, [YouTube], https://www.youtube.com/watch?v=T05t-SqKArY&t=427s&ab_channel=LuisSerrano;

[34] ***, WordNet, [Wikipedia], https://en.wikipedia.org/wiki/WordNet;

[35] Francis Bond, Lars Nygaard, Adam Pease, John McRae, Luís Morgado da Costa, Open Multilingual Wordnet, [White Paper] , http://compling.hss.ntu.edu.sg/omw/;

[36] ***, *WordNet Interface*, https://www.nltk.org/howto/wordnet.html;

[37] J. Ramos, *Introduction to Natural Language Processing: NLP Tools For Python*, [White Paper], https://itnext.io/introduction-to-natural-language-processing-nlp-tools-for-python-cf39af3cfc64;

[38] E. Kidd, Visualizing WordNet relationships as graphs [White Paper], http://www.randomhacks.net/2009/12/29/visualizing-wordnet-relationships-as-graphs/ ;

[39] Danny Donchev, [White Paper], https://www.fortunelords.com/youtube-statistics/;