Analysis of Cognitive and Keyword-based Approaches for Searching Judicial Documents

BY

MATTEO MARZIALI
M.S., Politecnico di Milano, Milan, Italy, 2019

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

ACKNOWLEDGMENTS

This research work is the result of the joint effort of all the people that collaborated with me, both working and standing by my side. Firstly, I want to thank Pier Luca Lanzi, Cornelia Caragea and Andrea Lui, without whose valuable advice and support this work would not have been possible. A thanks also goes to Business Integration Partners and, in particular, to all my colleagues from xTech team for giving me the chance to challenge myself working in a professional environment. I want to thank my friends, those of a lifetime and those met during my academic career, you made these years as rich and exciting as possible. Finally, the most heartfelt thanks goes to my family, without your love, your sacrifices and your support I could never have achieved my prestigious goals.

MM

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

	Chapter I
IR	Information Retrieval
NLP	Natural Language Processing
HCI	Human-Computer Interaction
TR	Text Retrieval
OCR	Optical Character Recognition
CNN	Convolutional Neural Network
LSTM	Long-Short Term Memory
RNN	Recurrent Neural Network
BoW	Bag of Words
LSA	Latent Semantic Analysis
SVD	Single Value Decomposition
TF-IDF	Term Frequency - Inverse Document Frequency
L2R	Learning to Rank
NN	Neural Networks
CBOW	Continuous Bag of Words
DNN	Deep Neural Networks

QE	Query Expansion
SERP	Search Engine Result Page
	Chapter II
AI	Artificial Intelligence
API	Application Programming Interface
KL Divergence	Divergence of Kullback-Leibler
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
	Chapter III
ELMo	Embeddings from Language Models
LDA	Latent Dirichlet Allocation
biLSTM	bi-directional Long Short Term Memory
PDF	Portable Document Format
RAKE	Rapid Automatic Keyword Extraction
PRF	Pseudo-Relevance Feedback
	Conclusions
UI	User Interface

SUMMARY

Searching and retrieving information efficiently represents an urgency in both ordinary and business tasks. With the introduction of high-performance storage systems and cloud technologies, reporting on paper sheets resulted an obsolete practice. Hence, companies aimed at reducing this type of operations to digital processes, employing digital repositories to store files in order to gain in availability and resilience. In this situation, software that provide functionalities to allow direct and valid access to stored data are referred to as search engines.

Aiming at analyzing cognitive and keyword-based searching algorithms, the goal of this dissertation is to develop a domain-specific search engine capable to combine the two cited approaches. The rationale behind adopting these two techniques together has to be found in the necessity to overcome the lack of 'query context' and 'intent understanding', along with the inefficiency of common procedures in handling equal words carrying diverse meanings. To meet the desired objectives, Text Retrieval was carried out on a juridical domain by performing the search on a corpus of authentic legal documents from the Italian Court of Cassation.

We organized our work into two core activities: Document Processing and Text Retrieval, both integrated in the Search Engine pipeline. In particular, Text Retrieval has been performed on top of processing units expressly built for granting proper answers to literal and non-literal queries. During the Document Processing phase, significant effort has been destined to extracting texts from actual judicial documents, initially in the form of images of scanned documents. Text Retrieval, instead, concerned with the realization of a search engine pipeline featuring diverse Deep Learning approaches. Such techniques involved the encoding of text portions into a more furbished representation by capturing both syntactic and semantic word features. Finally, the considered embedding approaches are compared by collecting the answers to specific questionnaires given to a random sample of people with the purpose of validating our approaches in a concrete use-case scenario.

CHAPTER 1

INTRODUCTION

Information Retrieval represents a crucial and cross-domain research field, indeed quickly and efficiently accessing data is a necessity in everyday activities. The underlying software for automatically performing searches against vast number of sources are known as search engines. Although these software are now part of everyone's daily life, they are constantly improved with recent technological breakthroughs and there are ad-hoc search engines for specific business applications still to be developed. Also, information has many shapes, it can consists of text, image, or even audio an video, each to be handled specifically on both the user and the system sides, as not only documents can contain data different from text, but also people can express a necessity with non-textual manifestations. Thus, Information Retrieval (IR) encompasses many computer science research fields such as Speech Recognition, Computer Vision and also Text Retrieval, where users express their need as a textual query and the system returns them a corresponding list of texts. The purpose of this work is to develop a search engine which features keyword-based and cognitive approaches for images of scanned documents. In achieving this, we reduce a Computer Vision problem to a Text Retrieval task by firstly extracting the text from the source files using an Optical Character Recognition (OCR) engine and then implementing a system to search against the collection of the derived text documents. Concerning the search engine development, we firstly identify the backbone of the architecture in five principal components, then we focus on finding the most suitable keyword-based and cognitive techniques and on effectively combining them. The five components are responsible for the following operations: document processing, document representation, document retrieval, document ranking and results visualizing. The rationale behind combining keyword-based and cognitive approaches consists in aiming at effectively answering to both literal and non-literal queries. Literal queries are those intended to find the exact same words in the results, on the other hand non-literal are directed to concepts and contexts similar to those expressed by the query. In pursuing the specified goals, we find a suitable domain in the judicial field. In particular, we identify the optimal scenario for developing and testing our search engine in a collection of about 12000 judgements from the Italian Court of Cassation, as they are verbose, contain many keywords and cover a wide range of topics. The core of the search engine and consequently of this dissertation is the word embedding component, responsible for capturing syntactical and semantic proprieties of text and encoding them into a compact representation, namely numbered vectors. Indeed, during this step morphological and contextual features of text are extracted and will be exploited in downstream task for defining the similarity of the query with each candidate result. The similarity, actually, measures the relevance between the user input and the search engine results, through the information encoded via word embedding model. Thus, we organize this thesis as a comparison among different embedding techniques, with the already stated purpose of finding an approach capable to outperform state of the art encoding models in performing cognitive and keyword-based text searches.

We provide here an overview on the organization of concepts with respect to both the overall structure of this dissertation and each individual chapter.

The first chapter presents a brief introduction to Information Retrieval ambit. Initially, we are going to provide an overview on Text Retrieval and its sub-areas. Then, we focus on ad-hoc retrieval for images of scanned paper files, we analyze some common document processing techniques and we describe the state of the art and the functioning for the components of search engines, software that perform automatic search on a document collection provided with a user query.

The second chapter , recovering the structure of 2, describes the techniques and the methods we have considered in addressing our problem among those present in the vast state of the art literature. For preliminary steps, we cover the approaches evaluated for extracting text from source images and the algorithms involved in processing the extracted text. Then, we present and detail the word embedding models used for encoding texts. In the end, we analyze the methodologies applied in performing the retrieval, namely the task of selecting the sub-set of documents that are candidate to be relevant with respect to the specific query.

The third chapter aims at detailing the characteristics of the addressed problem. Initially, we consider the selected domain and the rationale behind this choice. Furthermore, we explore the data by firstly describing some general aspects of the dataset and secondly analyzing common patterns and features of the documents. Finally, we look at the project specification by explaining goals, requirements and corresponding functions for each of the search engine components.

In the fourth chapter we describe in depth the solution we elaborated for addressing our problem. Still, we follow the structure defined also in the other chapters. Thus, we describe how document have been collected, how texts have been extracted from images and how they have been consequently processed. Then, we detail, in depth, how the embedding models presented in Chapter II have been used and combined together in order to learn text representations. Afterwards, we present how the proposed retrieval techniques have been used in selecting documents relevant to the query and how these documents have been ranked in terms of similarity measure used and required computations. In conclusion, we define the characteristics of the visualization component, responsible for presenting to the user the search results. Finally, we summarize the sequence of operations involved in the search process, detailing also the files required to finalize it. In addition, We present the main aspects of the search engine maintenance.

The fifth chapter is aimed at presenting the results of the evaluation performed for the proposed search engine. Reasonably, firstly we describe the evaluation paradigm defined, then we clarify the expected results and we provide experimental results and we comment the analysis and the comparisons carried on. Lastly, we conclude providing a key for interpreting the obtained results.

In order to provide the instruments to understand the algorithms behind Neural Networks, we endow this contribution with **one appendix**. Here we dwell on basic Neural Networks (NN) concepts, their architecture and functioning, describing also some of the features and parameters involved in the training of such networks.

CHAPTER 2

INFORMATION RETRIEVAL

The goal of this chapter is to define the specific domain of the work as well as to introduce the addressed problem. Firstly, we provide some basic knowledge about Information Retrieval ambit, then we analyze the ad-hoc text retrieval problem and how Search Engines address it. Finally, we present the main Search Engine components and the most relevant approaches adopted, in the past few years, in implementing them.

2.1 Theoretical Overview

Information is crucial in many ambits. Data qualifies people, incorporates the knowledge and represents the most valuable asset for many companies, indeed storing and retrieving documents and information has always been a necessity. With the advent of computers, from the ability to memorize large amounts of data and the need to access it, arose the field of Information Retrieval (IR). In Computer Science field, tasks concerned with interactions between machines and human language, particulary those related to automatic processing of language data, such as problems faced by IR, belong to the Natural Language Processing (NLP) macro area. As pointed out by Singhal [1], Information Retrieval refers to the action of automatically retrieving knowledge relevant to a user need from a collection. Retrieval of information, in terms of Human Computer Interaction (HCI), can occur in different forms. Users express their needs with a query, textual when they type on a keyboard, voice-based when they formulate a vocal request, visual when interrogation involves images, sometimes the query can also be implicit. On the other hand, provided results can be either text or multimedia documents. In particular, we address the problem of Text Retrieval (TR), where the query consists in a string of text and the system returns a ranked list of text passages. Text Retrieval according to Mitra et al. [2] covers two main tasks:

- *Ad-hoc Retrieval*, which refers to the task of retrieving and ranking documents relevant to a user query, based on the correlation between them;
- *Question Answering*, which deals with the task of returning to the user an exact answer to the question submitted through a query;

In this work, we are focusing on the ad-hoc retrieval problem. This task has been widely explored in the past years, although there are many either unsolved or poorly addressed challenges. Among these, we can list the ad-hoc retrieval task from scanned paper documents, which represents the actual topic of this dissertation. Dealing with a collection of images, rather than textual files, leads to a greater complexity, indeed, text has to be extracted via image processing in order to make it accessible for retrieving purposes. The required transformation operation is called Optical Character Recognition (OCR) which, according to [3], consists in "the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text". Definitely, despite the peculiarity of dealing with images of documents, the addressed problem can be reduced to the classical ad-hoc retrieval, which is addressed by a specific class of software, search engines.

In this section we are going to present some fundamental concepts of Information Retrieval, which are widely used in the following pages of this dissertation, as well as the main Search Engine components. In addition, we are going to describe traditional approaches adopted to solve the challenges carried by these processing units.

2.2 Document Processing

Document Processing is a crucial step in any Natural Language Processing (NLP) task. It refers to apply transformations to the source documents so to obtain a clean version, optimized with respect to the following required computations. With this being a determinant step, a keen eye needs to be adopted for document processing. In fact, even if usually touching the source files is necessary, it can lead to unwanted consequences, such as introducing bias and causing information losses. Considering the context of this thesis, dealing with scanned paper documents, firstly requires us to extract the text and secondly to process the obtained content. In this section, we are introducing the historically consolidated techniques for performing OCR from images and processing texts to perform NLP tasks.

2.2.1 OCR

"OCR is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text", as explained in [3]. It aims at digitalizing printed texts in order to make them electronically edited, searched, efficiently stored and used in machine processes such as cognitive computing. According to [3], the core algorithm of OCR engines are based on two different approaches. Both the techniques work with a character-level granularity, or rather, both try to isolate each character from the original document image and in turn recognize them. These approaches are:

- *Pattern matching*, which consists in "comparing an image to a stored glyph on a pixel-by-pixel basis". The performances relies on the quality of the two glyphs, the stored must be similar in font and scale to the other, which has to be correctly isolated from the rest of the image. This approach works well with typewritten text of already seen font.
- *Feature extraction*, which relies on extracting features such as lines and their intersections, closed loops and others, from the input glyph. These features are then compared in similarity to the features of stored glyphs so to get the character whose features are the most similar to the input ones.

Traditionally, state-of-the-art OCR systems involve Machine Learning techniques, mostly in the ambit of Neural Networks (NN) and Deep Learning (DL), such as Convolutional Neural Networks (CNN) and Long-Short Term Memory (LSTM), a type of Recurrent Neural Networks (RNN) which are suitable for processing not only data points but whole sequences of data throught their bi-directional connections. In practice, unlike many of these architectures, which are patented and available as cloud services, the most valuable open source project is represented by 'Tesseract' OCR engine. Originally developed by Helwett-Packard Laboratories, 'Tesseract' is carried on by Google since 2006 and the last stable version relies on a Long-Short Term Memory (LSTM) Neural Network architecture. As explained in [4] by Smith, Tesseract belongs to the OCR engines which perform feature extraction according to the following step-by-step pipeline:

- *Connected component analysis*, which aims at storing the component outlines and gatering them together by nesting, into Blobs.
- *Feature extraction*, which manages to organize "Blobs into text lines, then the lines and regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to the kind of character spacing. Fixed pitch text is chopped immediately by character cells. Proportional text is broken into words using definite spaces and fuzzy spaces".
- *Two-pass Recognition*, which, in the first pass, attempts to "recognize each word in turn. Each word that is satisfactory is passed to an adaptive classifier as training data. The adaptive classifier then gets a chance to more accurately recognize text lower down the page. Since the adaptive classifier may have learned something useful too late to make a contribution near the top of the page, a second pass is run over the page, in which words that were not recognized well enough are recognized again".
- *Final phase*, which consists in resolving the fuzzy spaces and exploring alternative hypoteses for text height to locate small-cap portions.

2.2.2 Text processing

During this step, which is usually referenced as preprocessing, some popular operations, such as data cleansing to correct corrupted records, data editing to review the quality of data and data wrangling to represent raw data into different formats more suitable for downstream tasks, are performed. NLP is of no exception in the need to transform input documents before applying any algorithm on them, however, for processing text the following techniques have been consolidated in years of research.

- *Tokenization*, which refers to the process of segmenting a portion of text into subsets. Tokenization, that is converting a text into a list of representative tokens, can be performed at different granularities, indeed a document can be tokenized into sentences, words or even characters. During tokenization process some words, characters or punctuation can be removed by applying stop-words and punctuation removal, as well as other filtering techniques.
- *Punctuation removal*, which consists in removing from texts punctuation symbols. Punctuation removal can address all reference marks as well as just few ones. For example, aiming at tokenizing text into sentences, you may want to remove all the punctuation but full stops, to exploit them in the process of identifying sentences.
- *Stop-word removal*, which consists in removing from texts those terms that are of low meaning impact. In practice, the stop-words definition is domain dependent, sometimes they are taken from a fixed list made by articles, connectives and other auxiliary terms, sometimes they are composed considering the least frequent words in the collection.
- *Stemming*, which aims at reducing all the words to their root form by chopping their ending letters off according to an heuristic process. There exist many stemming algorithms, however the Porter algorithm, which involves five sequential word reduction steps driven by different rules, whose application depends on the term that is being transformed,

is the most known and used, as it has been recognized very effective in practice. (i.e. 'information' → 'inform', 'retrieval' → 'retriev')

- *Lemmatization*, which pursues the same goal of stemming, reducing inflectional word forms to their base format, known as *lemma.* Unlike stemming, Lemmatization consists in applying transformation according to word morphology analysis and usually involve a specific vocabulary to obtain the dictionary form of terms. (i.e. 'Search engines are coded using Python' → 'Search engine be cod use Python')

Presented techniques consist in general approaches for text processing, however, although these approaches have found a wide usage in many NLP tasks, their application should be considered with respect to the faced problem, which can require ad-hoc solutions.

2.3 Search Engine

Search Engines, according to Tarakeswar et al. [5], are programs which search the database, gather and report the information that contains the specified or related terms. Traditionally, these retrieval systems are built upon the following components:

- *Word Embeddings*, which address the problem of representing words, sentences and whole documents in a context-based and light-weight fashion;
- *Document Indexer*, which refers to the task of storing the collection of documents so to perform the retrieval efficiently;
- *Document Ranker*, which deals with the task of ranking the retrieved documents in order to favor the results that are highly correlated with the query;

- *Result Viewer*, which is responsible of providing the results to the user;

2.3.1 Word Embeddings

An Embedding is defined by Mitra et al. [2] as "a representation of items in a new space such that the properties of, and the relationships between, the items are preserved". Word Embeddings refers to the concept of encoding an alphanumeric set of characters, namely a word, with a numeric representation, usually a vector of real numbers so to exploit its multidimensional structure to embed many information. The use of Word Embeddings relies on two aspects. Firstly, processing raw text is computationally expansive, both in terms of time and storage, secondly, a plain text representation prevents the computer from capturing the meaning and the semantics of words. Indeed, given the raw text, while a machine can check if two words are equal, it is not able to establish whether two words are synonyms, whereas, given a numeric representation and a similarity measure, it can infer relations, such as word equivalence by computing the distance between two word vectors. According to Mitra et al. [2],there are two main word embeddings techniques:

- *Local representation*, which creates one binary vector for each word in a fixed-length vocabulary with as many dimensions as the number of unique words in the collection of documents. These vectors have all zeros apart from one cell that has value '1', the cell related to the word represented by the vector.
- *Distributed representation*, which encodes each word in the vocabulary as a vector of real numbers that "can be either a sparse or a dense vector of hand-crafted features or a learnt representation in which the individual dimensions are not interpretable in isolation".

While the Local, or One-hot, representation is more intuitive, it is not applicable in dealing with a wide corpus, where the size of the vocabulary is huge. Moreover, since each word vector encodes a different space dimension, this representation prevents to exploit the arithmetic between vectors to establish word relations. It also provides no encoding for out-of-vocabulary words. On the other hand, the rationale behind distributed representation models consists in aiming at representing a word by its features, in order to define some notion of similarity among the different terms in the vocabulary according to specific properties. However, it is meaningful to point out that equal words with different semantics are mapped to the same vector in distributed representations, leading to an information loss.

In text retrieval ambit, the concept of word embeddings is crucial for turning into the problem of encoding longer text portions, such as sentences, paragraphs and whole documents. Historically, four main text representation approaches have been widely used to learn document embeddings out of a collection:

- *Bag of Words (BoW)*, which generates a term-document matrix by setting one column per unique word in the collection and one row per document in the corpus. There are three Bag of Words (BoW) implementations. One approach generates binary vectors by setting a cell to '1' if the word is present in the associated document, '0' otherwise. Another approach consists in creating a vector where each cell stores the number of times that the related word appear in the correlated document. Finally, the third approach consists in taking the term-document matrix generated by the second approach and multiplying each cell by a factor that penalizes words that occur in many documents and favors words that are rare in the corpus. The rows in this generated matrix are called Term Frequency - Inverse Document Frequency (TF-IDF) vectors.

- *Latent Semantic Analysis (LSA)*, which aims at generating low-dimensional vectors of features by applying Singular Value Decomposition (SVD) to the term-document matrix built with BoW vectors in order to map sets of words to the concept they express so to have vectors with one cell per topic in place of one cell per word.
- *Learning to Rank (L2R)*, which tries to figure out the feature vectors for each document so to maximize the ranking performances. This approach requires supervised data, that in Text Retrieval consists in having a set of queries and lists of document relevant for each query.
- *Neural embeddings*, which typically consists in the weights of the last layer of a neural network built to deal with a prediction task. These embeddings result from neural network models which are trained to predict words from their attributes. Both these two components, the word and its features are encoded by either one-hot or distributed representations. These vectors are fed in the input layer, the model learns dense representations with a reduced size by minimizing the prediction error and vectors with the same representation of input ones are retrieved out of the output layer.

Recently, Neural Networks are to the fore for encoding text, in particular, a set of models named Word2Vec represents the state-of-the-art for word embeddings. Word2Vec consists in two types of two-layer neural network created by a Google research team, whose work is presented in the paper 'Efficient Estimation of Word Representations in Vector Space' by Mikolov et al.

[6]. ”Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space” In practice, there are two Word2Vec models, namely Continuous Bag-Of-Word (CBOW) and Skip-gram. Continuous Bag of Words (CBOW) neural network learns to predict a word given a window of surrounding words as input, oppositely, Skip-gram model is trained to predict the surrounding words given the current word as input, as shown in Figure 1. These rather simple models, since they are very fast to train, have been shown to be very effective in practice, however they have a main drawback. Considering the idea behind word2vec, as well as its neural networks architectures, it is clear that it carries a concept of semantics which is strongly related to the idea of word position in sentences. In practice, this leads to the generation of similar vectors for words that are used in similar contexts, causing that terms such as ’good’ and ’bad’ are considered very similar when they are totally opposite. However, word2vec has been of inspiration for many other neural network models for text embedding. For example, Joulin et al. in [7] proposed the fastText model, a sort of Word2Vec revisiting, which, instead of taking word vectors as input features, takes n-grams features, where a n-gram is a set of n characters, and predicts the current n-gram given a window of surrounding n-grams. Besides from these rather simple NN models, also more complex structures such as Recurrent Neural Networks and Deep Neural Networks (DNN) have been applied in order to learn more and more effective word embeddings. Kiros et al. in [8] describe the Skip-thought vectors, obtained through an

Recurrent Neural Network (RNN) encoder-decoder neural network architecture whose goal is to predict from an encoded sentence in input, the preceding and the following sentences as output, so to learn similar vectors for sentences that share semantic and syntactic properties. Neural Network architectures for learning word embeddings are multiples, however the adoption of one approach rather than another is strongly related to the problem to be addressed. Surely, factors like the size of the dataset, the training time, the desired embeddings granularity, that can be either at word or sentence or paragrah or even document level, are to be considered in the process of neural network selection.

2.3.1.1 <u>Notions of Similarity</u>

A key concept in Text Retrieval consists in the notion of similarity. In order to retrieve documents related to a user query, it is crucial to define a concept of similarity and a similarity measure. Traditionally, since Text Retrieval deals with word and document vectors, Cosine Similarity is the adopted measure to compute words relatedness. Given two vectors of attributes, A,B, and relative components, A_i,B_i, the Cosine Similarity, $cos(\theta)$, is defined as follows:

$$cos(\theta) = \frac{A \cdot B}{\|(A)\| \cdot \|(B)\|} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.1}$$

Intuitively, such a convention is explained by the usage of numerical vectors itself, in fact, adopting these structures allows to map each word to a point in a multidimensional space, hence word similarity is efficiently captured by the angle between the two term vectors. However, other similarity measures exist, such as Euclidean and Manhattan distances. The similarity measure

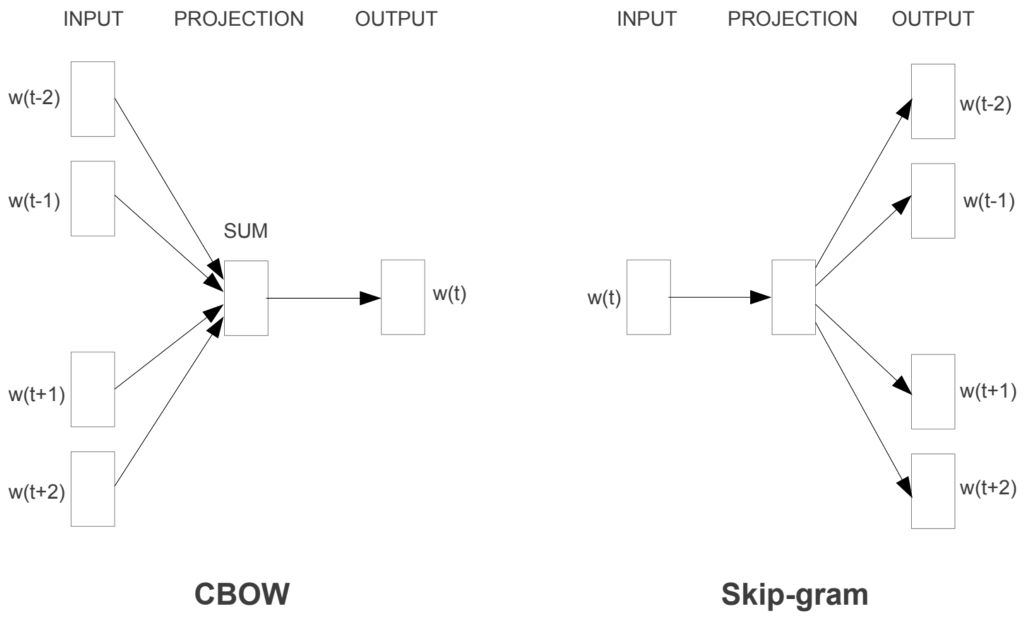


Figure 1: Word2Vec architectures.
www.machineintellegence.com/word-encoding-and-embedding-algorithms/

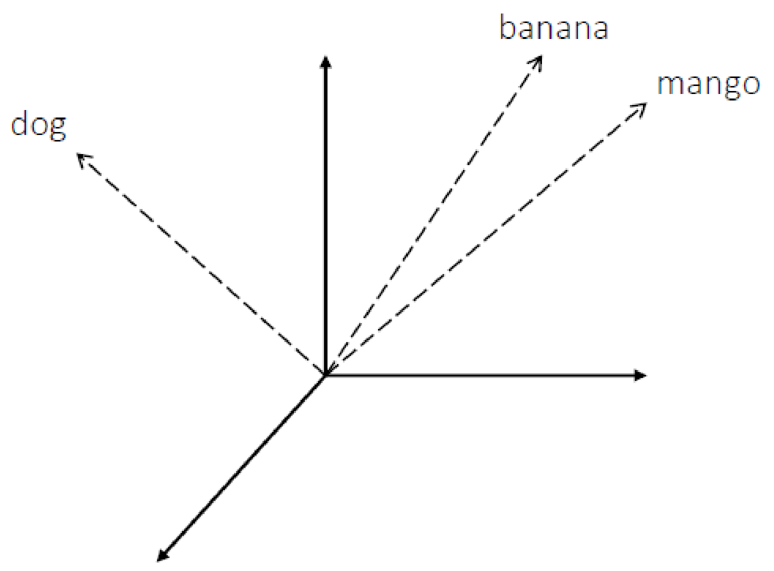


Figure 2: A vector space model that places the word "banana" closer to "mango" than to the term "dog".
https://www.microsoft.com/en-us/research/uploads/prod/2018/04/NeuralIR-Nov2017.pdf

choice depends on the concept of similarity that we want to quantify, that in turn derives from the problem domain.

2.3.2 Document Indexer

One of the main challenges in building Search Engines is to ensure a limited response time to user queries. In pursuing this, document indexing is a key component, indeed, its impact is two-fold. Firstly, indexing speeds up the search for matching documents, those files that are related to the query, thanks to the fact that it allows to retrieve them without checking their content. Secondly, indexing operates as a filter at query time, when the user has entered an input query and is waiting for the results. Since answering to user interrogations requires to compute similarities between the query and the documents in the collection, ideally, on a time saving purpose, we would like to have Search Engines that calculate relatedness scores exclusively for documents that are correlated with the query. In order to achieve this goal, an efficient document indexing is essential to filter out documents that are known to be uncorrelated with respect to the query. According to Fuhr et al.[9], document indexing is the task of assigning terms to documents for retrieval purposes. However, documents include a wide range of information other than text content, such as title, publication date, author and many alternative characteristics. Hence, historically two main indexing approaches have been defined:

- *Full-text indexing*, which consists in creating a structure, namely an Inverted Index, to map to each word in the collection vocabulary all the documents which contain that term. This basic index prevents to search documents where query terms are not present, but may rule synonyms and hyponyms out.
- *Metadata indexing*, which aims at building indexes upon context-based document information such as the topic, a set of keyword and the author.

Full-text inverted indexes allows to search the content of documents, however they require a lot of storage space, proportional to the size of the vocabulary. In order to leverage their impact on memory, some common text processing techniques can be applied in order to filter out from the vocabulary words, such as articles and complex verbal forms, which are of no contribution in computing whether a document is relevant for a query. On the other hand, indexes built on top of metadata aim at handling a sort of categorical query, where the target is a characteristics of the document, such as its author, the number of times it has been read, its publication date and others. Hence, the impact of metadata indexes on memory is limited, in fact, they are proportional to the number of documents, usually less than vocabulary size. By concluding, since full-text and metadata filtering have a two-fold purpose, sometimes they are both present in Search Engines.

2.3.3 Document Ranker

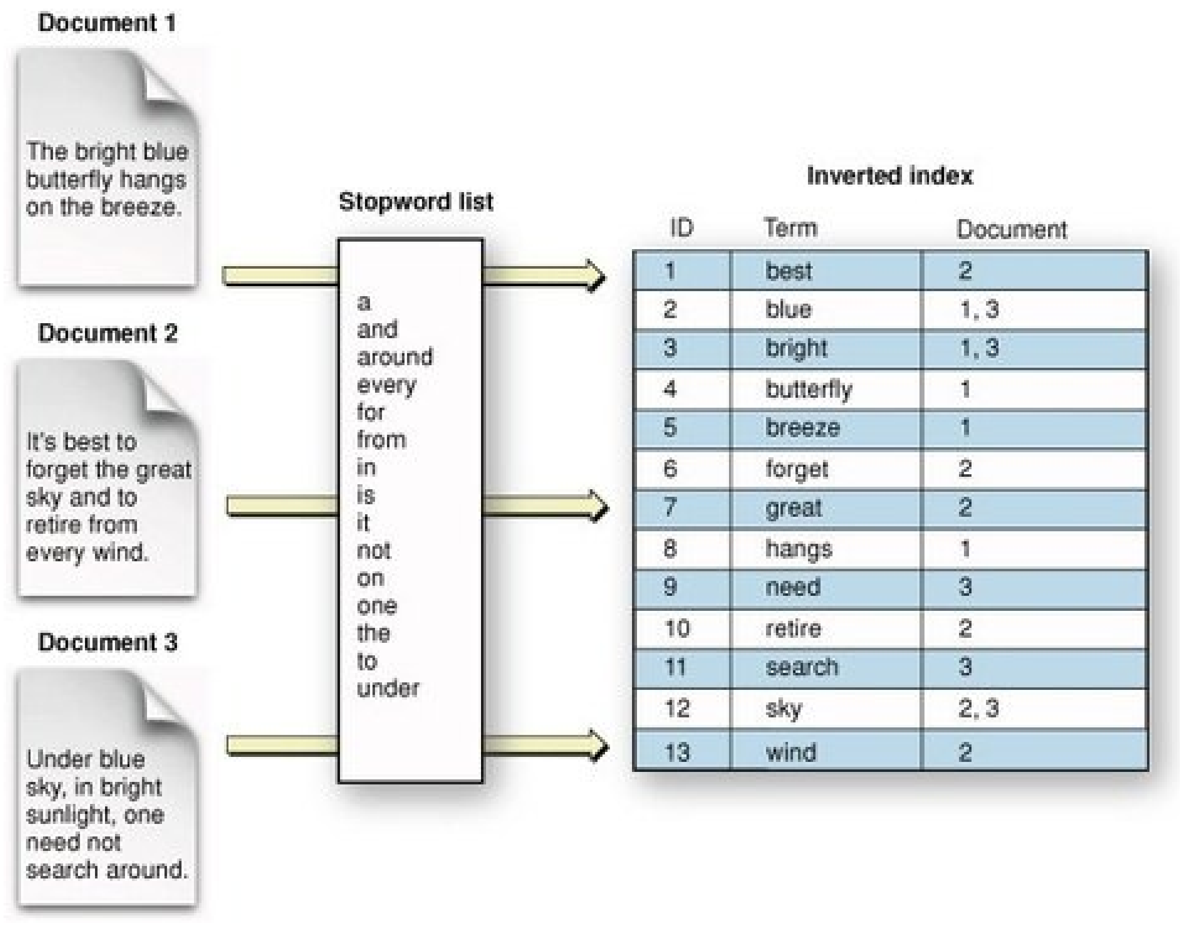
Ranking the matching documents is another key Search Engine component. Favouring results that are strongly correlated with user target is crucial both in evaluating system performances and in satisfying customers. Result ranking takes into consideration two metrics:

- *precision*, that counts the fraction of retrieved documents that are relevant to the query.

$$precision = \frac{RelevantDocuments \cap RetrievedDocuments}{RetrievedDocuments} \tag{2.2}$$

- *recall*, which is the fraction of relevant documents that are successfully retrieved.

$$recall = \frac{RelevantDocuments \cap RetrievedDocuments}{RelevantDocuments} \tag{2.3}$$



Document 1: The bright blue butterfly hangs on the breeze.

Document 2: It's best to forget the great sky and to retire from every wind.

Document 3: Under blue sky, in bright sunlight, one need not search around.

Stopword list: a, and, around, every, for, from, in, is, it, not, on, one, the, to, under

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Figure 3: Inverted index construction.
https://i.pinimg.com/originals/94/8f/52/948f52de7fb6dcef36fc5f699e7b6d58.png

Precision and Recall are real numbers between 0 and 1, that are usually considered together, indeed we talk about precision-recall tradeoff. Briefly, Estimating precision gives no information about recall and viceversa, and usually each measure grows at the expense of the other one. By retrieving all the documents, recall is 1 but precision is low, on the other hand, as much as we reduce the number of retrieved documents, the precision grows and the recall decreases. In practice, by plotting precision and recall we can estimate the area under the precision/recall curve, the bigger the area, that is, the closer the curve to the point (1,1) in the space, the better the IR system performance.

This brief introduction to Precision and Recall aims at presenting how document ranking performances are evaluated in order to better understand the related component, so to keep in mind the rationale behind ranking documents approaches. Document ranking heavily relies on the document indexing phase and the selected similarity measure, which implicitly define the time and the computation required to the ranker to score each retrieved document. In practice, a basic ranker can simply sort the retrieved documents based on their similarity score with respect to the query, in such a scenario, search engine performances depends mostly on the selected word embeddings. However, advanced search engines can either assign many scores to documents or perform the so called 'Query Expansion' (QE), which consists in improving the query in order to target more results, two approaches that require more complex computations in ranking the results. For example, it is well-known that the famous 'Google' search engine, along with many other scoring technique, scores web pages based on the PageRank algorithm, that works by estimating the amount and the values of links pointing to a web page to roughly determine how important the related website is. On one hand, considering different scoring measures implies that search engine needs to combine diverse scores in order to maximize performances. On the other hand, performing Query Expansion usually increases the recall and lowers precision, so it is possible to penalize results obtained through Query Expansion (QE) to leverage this effect, requiring some computation in coupling distinct scores. However, what mostly affects document ranking is the addressed problem, particularly whether we are dealing with supervised or unsupervised data. In text retrieval ambit, we have supervised data if we have a set of queries and a set of documents related to them, otherwise we deal with unsupervised data. Dealing with supervised data, allows to exploit the a priori knowledge on data to improve search engine performances in ranking documents, indeed, it makes possible to build the scoring function that maximizes the precision-recall tradeoff. Oppositely, in unsupervised problems is not possible to drive the search engine development automatically, however human queries evaluation can be used to progressively improve the product according to the canonical software development life-cycle.

2.3.4 Result Viewer

Search engines, as user destined software, heavily rely on user experience and usability, in particular, since accessing information is the main goal of search engine users, the result viewer plays a crucial role for this task. Results are listed in a so called Search Engine Result Page (SERP) sorted in descending order per relevance score with respect to the query. Traditionally, each result displayed in SERP presents a Title, a link to the page of the document that matches the query and eventually a preview of the matching portion of text. In web search engines, two types of results can be distinguished, organic pages, retrieved by the software algorithms, and sponsored pages, that consist in advertising. Usually, a SERP contains a subset of all the retrieved items, so that many SERPs are generated in response to queries, with results that are progressively less relevant. In advanced and commercial search engines, such as Google and Bing, many elements are displayed in SERPs. As replies to queries that aim at locations it is common to see maps and images, single-term queries may have a definition box as first result, misspelled queries are answered with their potential correct formulations, similarly there are many other diverse approaches aiming at speeding-up and enhancing users information access.

2.4 Summary

So far we explored the state of the art for Text Retrieval related tasks, starting from Document processing, declined into text extraction and text elaboration, then we dwelt on search engine components, detailing the traditional approaches consolidated in literature in developing them.

CHAPTER 3

EVALUATED APPROACHES

In this chapter we present, from a formal point of view, the main tools and algorithms that we involved in the core components of the search engine. This chapter has the two-fold meaning of both clarifying the reasons which led us to apply such techniques to our problem and explaining their functioning, so to bring to the reader attention the critical points, as well as the advantages and disadvantages, of these approaches.

3.1 Introduction

Before getting into the main matter of this chapter, it is worth clarifying the rationale behind this part of the thesis. This introductory pages aim at separating the development pipeline presentation from the formal definition and explanation of the adopted approaches. This choice in organizing the work is keen to lighten the dense part dedicated to the pipeline and make the reading easier and more complete. However, it is worth remarking the difference and the distance among this chapter and the following one, about the structure of the conceived search engine. On one hand, here we present algorithms and tools for the core components, a subset of the search engine parts, formally, out of our work context. On the other hand, in the next chapter we define the whole search engine structure, along with how tools and algorithms have been used and combined to experiment and to get to the end product.

3.2 Document Processing

Concerning the processing of documents, one of the most crucial tasks consists in extracting text from document images. In performing this operation we exploited the already seen Tesseract open source OCR Engine by Google. The reason why we decided to go with this tool is simply that we were satisfied by the obtained results, indeed, the quality of the extracted text, along with the fact that Tesseract was free and open source, made up good reasons to adopt it. This powerful tool is constantly updated and we made use of the version 4.0.0, which features a new LSTM neural network focusing on line recognition, however it still features the legacy 'Tesseract3' which works on recognizing character patterns. According to the tests ran by Google, the LSTM architecture makes Tesseract faster and more accurate. In order to understand how LSTMs have been integrated in Tesseract Architecture, firstly we explain LSTM Neural Networks, then we analyze their impact in the OCR engine in question.

3.2.1 LSTM Networks

Recently, a particular model of Recurrent Neural Networks (RNN), namely LSTMs,have gained attention as they resulted very suitable for forecasting, indeed, their structure allows to process sequences of data more effectively than other network architectures. This is due to their characteristic of adding to the input of each iteration some information taken from the recurrent unit, which works as an internal memory of the past observations. Simple RNNs implement a similar functionality of using past information for predictions by adding to the input a sole weighted sequence computed by combining weights of the hidden layers at the previous forward pass, a less powerful way of reusing past information for predictions.

RNNs behaviour is summarized by the following equations:

$$\vec{a}_t = \vec{b} + \vec{W}h_{(t-1)} + \vec{U}x_t \tag{3.1}$$

$$\vec{h}_t = tanh(\vec{a}_t) \tag{3.2}$$

$$\vec{o}_t = \vec{c} + \vec{V}h_t \tag{3.3}$$

$$\hat{\vec{y}_t} = softmax(\vec{o}_t) \tag{3.4}$$

where $\vec{a}$ consists in the input vector to be fed to the activation function $tanh(\cdot)$ of the hidden layer. The output of the previous state, $\vec{h}$, represents the recurrent state to be used during the next iteration; $\vec{o}_t$ is the resulting net of the transformation $softmax(\cdot)$, a kind of logistic function that gives as outcome the end prediction $\hat{\vec{y}}$; $\vec{W}$, $\vec{U}$ and $\vec{V}$ represent the weight matrices; $\vec{b}$ and $\vec{c}$ represent the biases.

There are also RNNs made of more complex units called LSTM cells. These peculiar units are built with a gating system which allows that a neural network evolves more powerful reminiscent techniques. Thanks to this architecture, the RNNs (which in turn take the name of LSTM networks from the cell names) gain the faculty to selectively store data for an arbitrary number of iterations, overwhelming the single-step remembering capacity of basic RNN networks mentioned above.

Commonly LSTM architectures feature these three components where information sequentially passes through:

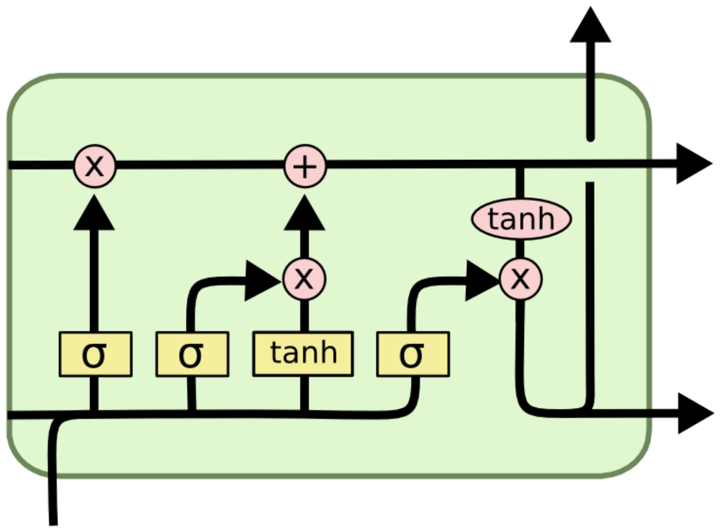


Figure 4: The internal architecture of a LSTM unit cell.
https://colah.github.io/images/post-covers/lstm.png

- a *forget gate*: this component deals with choosing whether to throw away or not the stored data. Actually, the decision is not binary, indeed it can range from 0 (get totally rid of the information) to 1 (totally keep the information), as it is computed as result from a sigmoid function. The forget gate is represented by the following equation:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.5}$$

- an *input gate*: this gate refers to the step when a decision about storing the input data that are entering the cell for the new iteration is taken. This state is updated according to the combination of the selection of the present values with a new candidate vector $\tilde{i}$. the input gate behaviour is explained by these equations:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.6}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.7}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.8}$$

- an *output gate*: this component, as the name suggests, is responsible for the output selection. It is based on the state of the cell, but also filtered further. Similarly to what happens for the input, the choice of the values to output is computed by a sigmoid function, the state, instead, is filtered using the so called hyperbolic function:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.9}$$

$$h_t = o_t * tanh(C_t) \tag{3.10}$$

Clearly, possible applications of LSTMs in any problem where is required to perform sequence learning are obvious. Indeed, these architectures are particularly suitable in a task where the value at the next iteration is strongly related to some values of previous time instances. Obviously, considering words as sequences of characters and thinking about them as non-random strings allows us to expect great result out of LSTMs application.

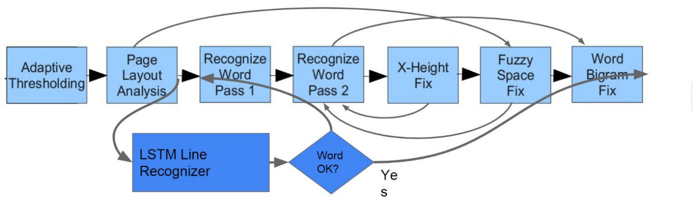


Figure 5: Tesseract architecture. https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/2ArchitectureAndDataStructures.pdf

3.2.2 Tesseract Architecture

Having introduced the LSTM networks, we can now analyze the Tesseract architecture. According to Figure 5, Tesseract v4.0.0, here analyzed, features a sort of LSTM path, in blue, which consists in an alternative to the legacy 'Tesseract3' path, in light blue. Indeed,by passing through the darker blue path, which consists in exploiting LSTMs, causes that, if the word identification ends positively, the classical word recognition approach is totally skipped. As we can see in the cited image, LSTMs are used in order to identify lines rather than directly words, even if, clearly, this two tasks are related, indeed lines are nothing else than sequences of words, which in turn are sequences of characters. As the picture shows, after identifying lines, firstly the network aims at recognizing words using LSTMs and then the legacy approach is performed only whether the former fails. Figure Figure 6 shows how Tesseract exploits LSTMs. First of all, it takes in input pixel sequences, indeed, the network is fed with one pixel of fixed height per time instant along the horizontal axis. The input passes through a two-layer

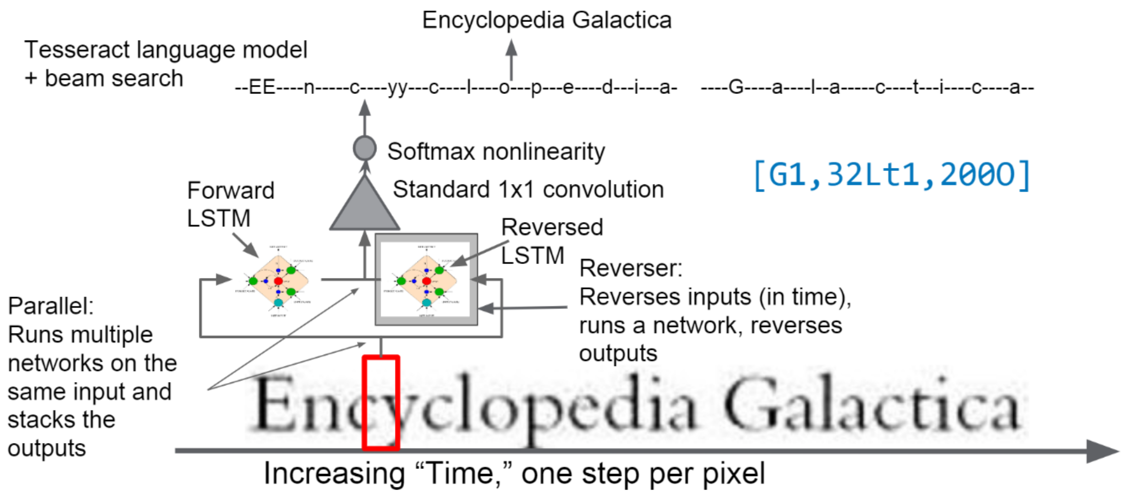


Figure 6: Tesseract LSTM word recognition approach.
hub.com/tesseract-ocr/docs/blob/master/das_tutorial2016/6ModernizationEfforts.pdf

architecture consisting of two LSTMs, a classical and a reversed one. Such combination is called bi-directional LSTM (biLSTM) and allows, in a way, to exploit both the past and the future information in the sequence, with respect to the current iteration. Indeed, bi-directional Long Short Term Memory (biLSTM) allows to process the input sequence from left to right through the classical forward LSTM and from the end to the beginning via the Reversed LSTM. At the very end of the architecture, a typical softmax layer outputs a prediction for the character in question by scoring 200 different alternatives. The alternative that receive the highest score represents the actual prediction for the character in question. Exploiting bi-directional LSTMs allows to add more context information to the current time input value, increasing the predictive power of standard LSTMs. However, it is worth clarifying that the adoption of this peculiar Deep Recurrent Neural Network is constrained to the faculty to have access the whole sequence, otherwise scanning the input backward is clearly unfeasible.

3.3 Word Embedding

Word Embedding refers to the task of encoding text, either words or larger portions, into numbered vectors in order to build a model that captures relations among textual elements, related both to the semantics and the syntax. Clearly, encoding text is a core step in search engines, as the effectiveness of the search heavily relies on this aspect. In practice, choosing an adequate technique for accomplishing this task is complex due to its delicacy. Hence, when approaching new problems, which means either developing ad-hoc software or adapting an existing program to new data, multiple possible solutions are explored in parallel and then compared to come up with the best solution. In doing so, we encoded text according to three different algorithms, the well-known fastText developed by Facebook, Skip-thought, presented in [8], and Embeddings from Language Models (ELMo), conceived by a group of researchers from the Allen Institute for Artificial Intelligence (AI2).

3.3.1 fastText

FastText is a free and open source model, presented in [10] by a Facebook Artificial Intelligence (AI) Research team, which revisits the word2vec model described in [6]. According to [10], fastText exploits the same neural network architecture adopted in word2vec for learning word representations, namely the continuous skipgram model shown in Figure 1.

However, the fastText approach follows a slightly different concept with respect to the one implemented by word2vec. Briefly, word2vec model aims at learning a vector representation per each word in a dictionary specifically built out of the collection of documents upon which the model has to be trained. In doing so, the word vectors, which consist in the weights of the network architecture hidden layer, are learnt by predicting the probability of observing an arbitrary number of surrounding words. This peculiar way of learning word representations, implies that in the resulting models, different words, which are often used in similar context get close representation regardless to their meaning. For example, adjectives, such as 'good' and 'bad' are seen very similar when their meaning is totally opposite. Coming back to fastText, it basically applies the same concept of word2vec but at lower granularity. Indeed, the aim of this model is to learn vector representation for each n-gram encountered in the documents of the considered collection. A n-gram simply is a portion of a word made of n letters, so word n-grams consist in all the sets of n characters obtained by sliding a window of size n over the word in question. A visual explanation of this concept is represented by Figure 7.
Considering this example, pretending to feed the word 'going' to the fastText skipgram architecture, it will be represented by the following bag of character n-gram:

'<go','goi','oin','ing','ng>','<going>'

As we can see, the whole word itself is included in the input. The special characters '<' and '>' are the boundaries which respectively define the beginning and the end of words, so to differentiate n-grams to complete words. For example, in this representation, the word '<her>' is different from the n-gram 'her' that we can find within the word '<where>'. Formally,

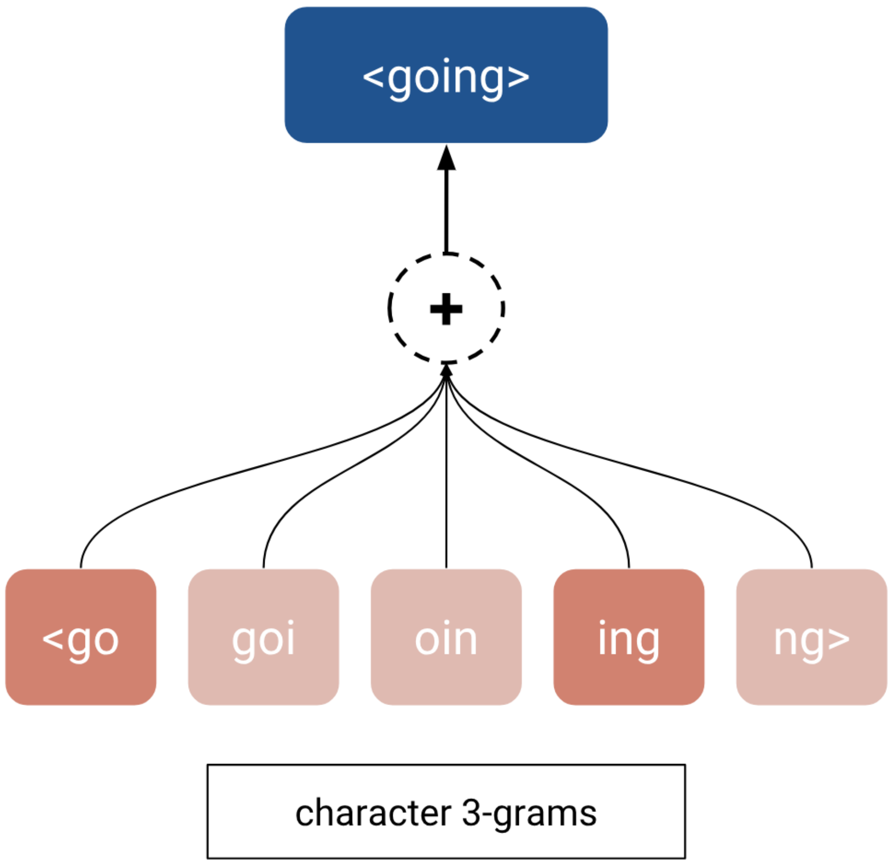


Figure 7: Example of 3-grams word representation.
https://medium.com/data-from-the-trenches/arithmetic-properties-of-word-embeddings-e918e3fda2ac

assuming to have a dictionary of n-grams of size G, given a word w and a related set of n-grams $G_w \subset \{1, ..., \text{G}\}$, we identify a vector z_g to encode each n-gram g, and we denote a word as the sum of the vector representations of its n-grams.

Regarding this peculiar representation, it is worth making some considerations. Firstly, in this model the 'positional' issue of word2vec that causes 'good' and 'bad' to be similar is leveraged, as word similarity depends also on the presence of equal n-grams. Furthermore, fastText is more accurate when it gets to encode out-of-dictionary words, those terms that are not present in the training collection. Since each word can be broken up into n-grams, it can be encoded by knowing the vectors of its n-grams, or at least some of them, rarely all the n-grams of a meaningful term are unseen. Finally, it has to be said that even if fastText takes into account contextual information as it encodes words considering surrounding terms, it is also evident that this information is scarce and sometimes harmful as we have shown with the 'good' and 'bad' example. On the other hand, the way fastText aims at representing words allows to build a strong keyword based model, as words with similar syntax are likely to be encoded by close vectors in the n-dimensional space, where n is equal to the word vectors size.

3.3.2 Skip-thought

Differently from word2vec and fastText, this model is suitable for learning sentence representations and in particular, taking up the words used in [8], the related paper, it is an approach for unsupervised learning of a generic, distributed sentence encoder. This characteristic of working at sentence level, has been the main factor that lead us to use this model, aiming at complying with project requirements Table III and Table V directly, in order not to

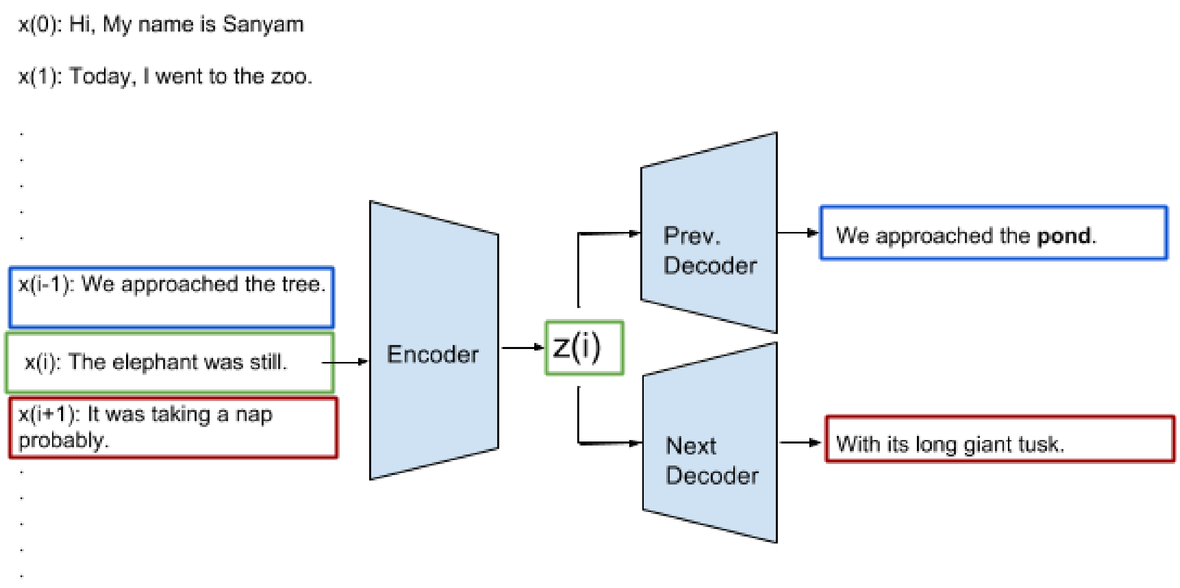

x(0): Hi, My name is Sanyam
x(1): Today, I went to the zoo.
x(i-1): We approached the tree.
x(i): The elephant was still.
x(i+1): It was taking a nap probably.
Encoder
z(i)
Prev. Decoder
Next Decoder
We approached the **pond**.
With its long giant tusk.

Figure 8: Skip-thought model architecture.
https://medium.com/@sanyamagarwal/my-thoughts-on-skip-thoughts-a3e773605efa

perform further computations. Skip-thought vectors for sentence encoding, as well as fastText, relies on a Encoder-Decoder Neural Network architecture.

As shown in Figure 8, the rationale behind the network structure is to predict the previous and the following sentences, given the current one.

The three components of Skip-thought network are the followings:

- **Encoder**: This network, usually a RNN such as LSTMs, takes as input the sentence x_i and computes a fixed length representation z_i.

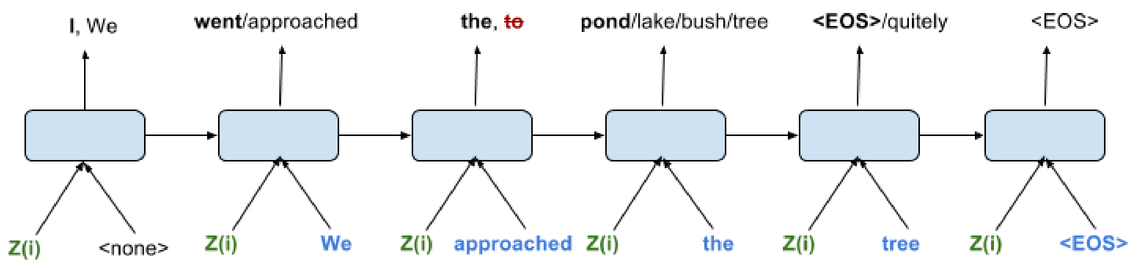


Figure 9: Example of sentence prediction performed by decoders.
https://medium.com/@sanyamagarwal/my-thoughts-on-skip-thoughts-a3e773605efa

- **Previous Decoder**: This network, still a RNN as LSTMs, takes as input the sentence embedding z_i and tries to reconstruct the previous phrase $x_{(i-1)}$.
- **Next Decoder**: This networks is analogous to the Previous Decoder but it has an opposite goal. By taking the same input z_i, it tries to reconstruct the following period $x_{(i+1)}$.

However, the decoders task is extremely challenging. Predicting a sentence given both the previous and the next one is an ambitious problem even for humans. The way this operation is performed by decoders consist in incrementally predicting sentence words, by knowing the actual sequence till the time instant that precedes the current guess, as shown in Figure 9. The learning technique, namely the training process, for this kind of network consists in minimizing the reconstruction error of the previous and next sentences, given the embedding z_i. Indeed, this error is back-propagated in order to make the network learn a better encoding z_i at the following iteration. The underlying assumption is that whatever leads to a better

reconstruction of the neighbouring sentences is also the essence of the current sentence.
Similarly to what happens with fastText, as the embeddings basically consist in the hidden layer, once that the training is terminated, we get rid of the decoding to keep the weight vectors z_i.
Such vectors, generate a space where semantic similar sentences are close to each other. It is worth noting that, differently from fastText, this model favors semantic information over syntactic in building encoding vectors. In fact, this sentence processing technique consists in finding the most significant words in each sentence for a given context of three periods. Clearly, this method leverages the impact of syntactical information stored in embedding vectors. Indeed, not only sentences with similar words can lead to deeply diverse representations based on their context, but also, since each word contributes to the sentence embedding, the resulting vector is less dependent on specific words.

3.3.3 Embeddings from Language Models

Embeddings from Language Models, ELMo in short, according to [11] consists in a model to learn "deep contextualized word representation that models both complex characteristics of word usage (e.g., syntax and semantics), and how these usages vary across linguistic contexts". Differently from fastText, which takes a words dictionary and aims at generating corresponding word vectors, ELMo analyzes each term within the context it has been used. Indeed, ELMo creates vectors on the fly, by passing the input through a deep learning model. In doing so, clearly, equal words correspond to different encodings, based on their surrounding context.

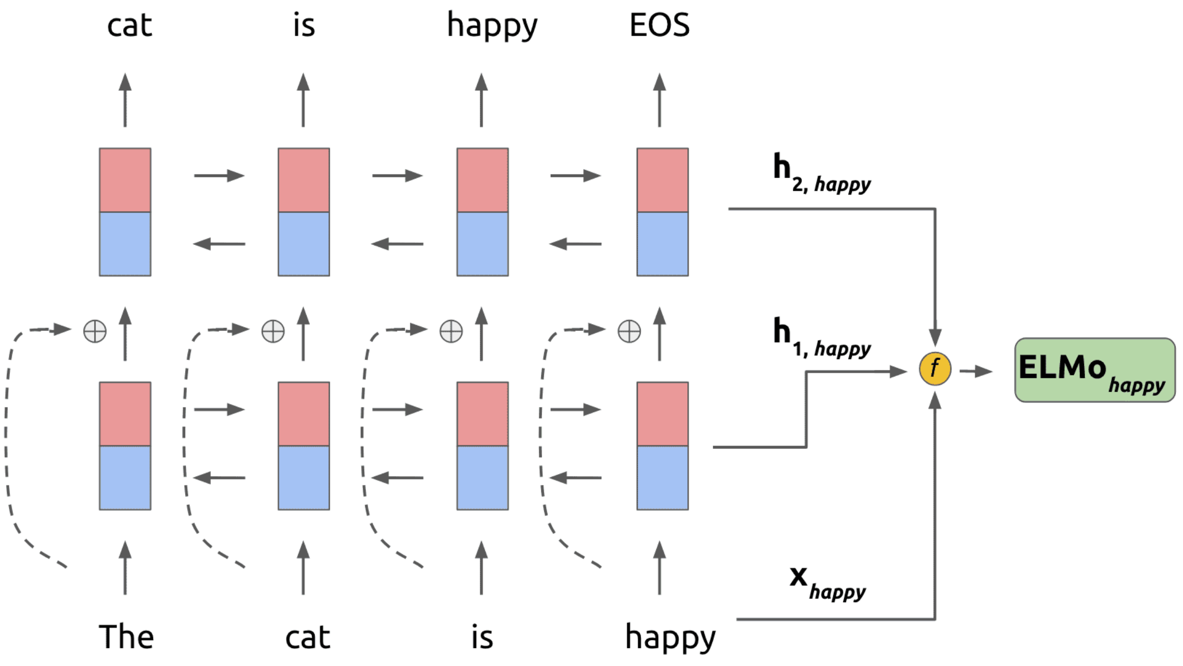


Figure 10: Architecture of Neural Network used to train ELMo embeddings.
https://www.mihaileric.com/posts/deep-contextualized-word-representations-elmo/

In Figure 10 we can see the Deep Neural Networks (DNN) architecture used for training ELMo model. It is essentially composed of an input layer, two biLSTM layers, the red box represents the forward LSTM and the blue models the backward one, and an output layer. Immediately we notice that the output layer, in this image, consists of a linear combination of the three layers before, namely x, h_1 and h_2, however, in [11], it is clearly stated that an alternative approach consists in taking the weights of the last hidden layer as embedding vector. The dashed lines that go from the input layer to the second hidden layer model the so called residual connections, a method for improving the training success of the network. Talking

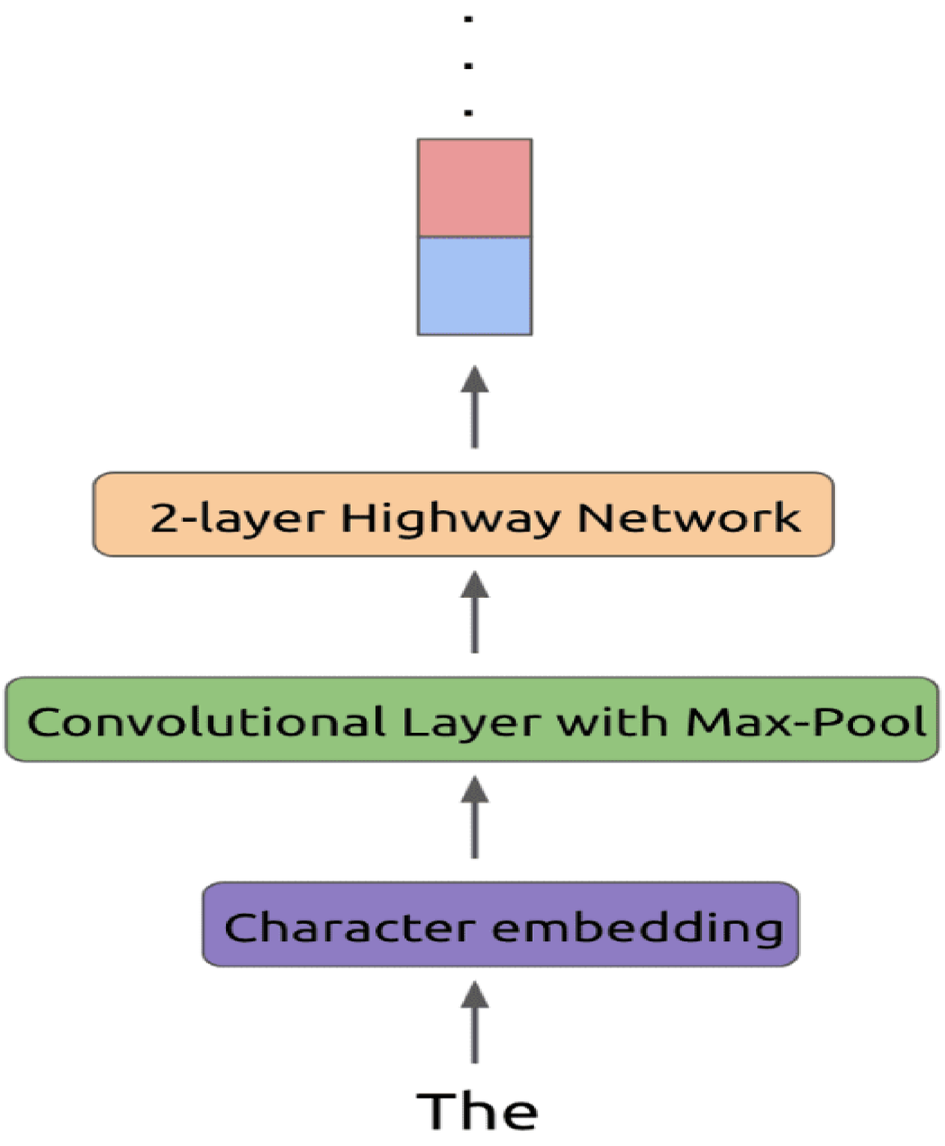


Figure 11: ELMo input layer architecture.
https://www.mihaileric.com/posts/deep-contextualized-word-representations-elmo/

about hidden biLSTM layers, according to [11] the number of such layers can be arbitrarily increased, augmenting the depth of the network. For the sake of simplicity we analyze the cleaner structure shown in Figure 10. Formally, for each token t_k is computed the following representation R_k:

$$\vec{R}_k = \{\vec{x}_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} \mid j = 1, 2\} \tag{3.11}$$

$$= \{\vec{h}_{k,j}^{LM} \mid j = 0, 1, 2\} \tag{3.12}$$

where $\vec{h}_{k,0}^{LM}$ is the token layer and $\vec{h}_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}]$ for each bi-directional LSTM layer. As of the input layer, we can see the actual unfolded structure in Figure 11. This image, indeed, shows how the input word is firstly converted into character embeddings, in turn fed to a typical convolutional layer, aiming at extracting morphological features by applying filters to the input. The new representation after passing through a two-layer highway network, that has a 'highway' path to convey the input directly to the output, is then fed to the first biLSTM layer. The usage of character embedding, in addition to be more suitable than the whole word in learning syntactical information, it is also suitable for encoding out-of-vocabulary words. Looking at the whole ELMo architecture, it is evident how, on one hand, the processing of the input shown in Figure 11 aims at capturing syntactical word proprieties, indeed, equal words are transformed into the same embeddings. On the other hand, the computation carried on by the hidden bi-directional LSTM layers aims at learning context-based information. Clearly, the purpose of combining both a context-based approach and a morphological one is to learn representations that feature both these aspects.

3.4 Retrieval

The Retrieval component is responsible of selecting those documents in the collection that are somehow related to the query, so to reduce the number of texts where to look for matching sentences. Obviously, this is a core task. Search engine performances heavily rely on this step, as wrong retrieval makes the system return results non-relevant to the user input, not because there are no related documents in the collection, but as they are not considered!
Besides, searching on the whole collection regardless to the query is not feasible, due to the computation time and effort needed in such a case. Recalling the overall goal of this thesis, developing a search engine which features both keyword-based and cognitive-based approach, we considered two filtering techniques to engage both these approaches. Thus, aiming at selecting documents based on their context we analyze the Latent Dirichlet Allocation (LDA) algorithm. Concurrently, focusing on filtering texts based on their keywords, we explore the Google Cloud Natural Language Application Programming Interface (API)s, in particular the Analyzing Entities tool, to extract entities from our source files.

3.4.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation, from now on referred to as Latent Dirichlet Allocation (LDA), consists in an algorithm to compute a topic model, given a collection of documents. The goal of LDA is to map each document to a set of topics so to define, based on words occurrence, a specific group of subjects for each text, those that mostly capture its context among all

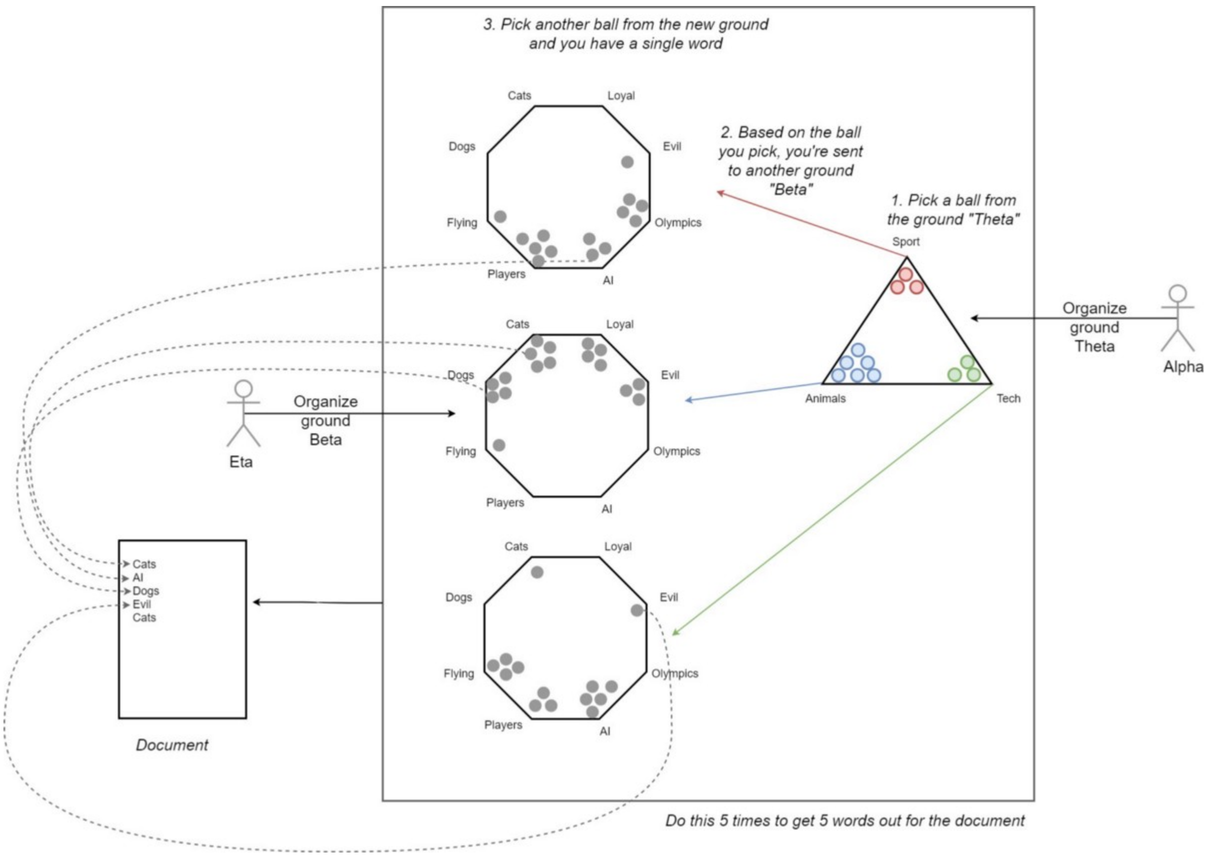

3. Pick another ball from the new ground and you have a single word
2. Based on the ball you pick, you're sent to another ground "Beta"
1. Pick a ball from the ground "Theta"
Sport
Animals
Tech
Organize ground Theta
Alpha
Organize ground Beta
Eta
Cats
Loyal
Dogs
Evil
Flying
Olympics
Players
AI
Cats
AI
Dogs
Evil
Cats
Document
Do this 5 times to get 5 words out for the document

Figure 12: Graphic visualization of an already-learnt LDA model. https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158

the topics that cover collection arguments. Due to the fact that these topics emerge during the modelling process, since they are inferred by running the algorithm, they are considered 'latent'. This algorithm assumes that each text can be related to a distribution of topics and each topic, in turn, is connoted by a distribution of words. According to this assumption, what we are interested in is finding a distribution of topics over documents and a distribution of words over topics. In order to understand how LDA algorithm works, we have to define the model parameters and how to learn them. Considering the final goal, building a topic modelling system, and having a look to Figure 12, we can see that the model has four parameters:

- Alpha (α): parameter that governs the distribution of topics on top of the collection.
- Theta (θ): Random matrix where θ(i,j) represents the probability of the i-th document to containing the j-th topic.
- Eta (η): parameter that governs the distribution of words with respect to the topics.
- Beta (β): Random matrix where β(i,j) represents the probability of the i-th topic to containing the j-th word.

The name Latent Dirichlet Allocation, comes out from the fact that both θ and β are modeled as Dirichlet distributions. In statistics, a Dirichlet distribution is a continuous probability distribution, reliant on a real integer numbered vector, which generalizes the Beta distribution to the multivariate case.

Formalizing what we are interested in finding, we get:

$$P(\theta_M, z_M, \beta_k \,|\, D; \alpha_M, \eta_k) \tag{3.13}$$

where the newly introduced z, M, k and D respectively represent the number of topics per document, the number of documents, the total number of topics and the collection of data.

In words, we are looking for the posterior probability of a distribution of topics per document

(θ_M), a number of topics per document (z_M) and a distribution of words per topic (β_k) given our collection (D). Assuming also that our collection is defined by a parameter vector of topics for each document (α_M) and a parameter vector of words for each topic (η_k). It is worth noting that we are assuming the number of topics k as given, we will analyze later how to define it. As long as computing the probability above is prohibitive, we estimate them by approximating it with a close known distribution, according to variational inference idea. In doing this, one way is to minimize the Divergence of Kullback-Leibler (KL Divergence), a measure of how close are two distributions, among our approximation and the given probability. This translates in the following optimization problem:

$$\gamma^*, \phi^*, \lambda^* = argmin_{(\gamma,\phi,\lambda)} D(q(\theta, z, \beta \,|\, \gamma, \phi, \lambda) \,||\, p(\theta, z, \beta \,|\, D; \alpha, \eta)) \qquad (3.14)$$

where γ, ϕ and λ respectively represent the three parameters we are approximating θ, z, β with. In the equation above, D(q——p) represents the KL Divergence between q and p and our goal is to find the γ^* , ϕ^* and λ^* that minimize the KL Divergence between the approximation q and the true posterior p. The computation is then carried on by iteratively solve the above equation until γ, ϕ and λ converge. Having build a LDA topic model means having a model that maps a set of topics to each document, along with their probability to cover it, and maps to each topic a set of words that contributes to that argument.

Coming back to the task of defining the number of topics in the collection, a valuable framework for learning this number is represented by Topic Coherence. Actually, topic coherence is a

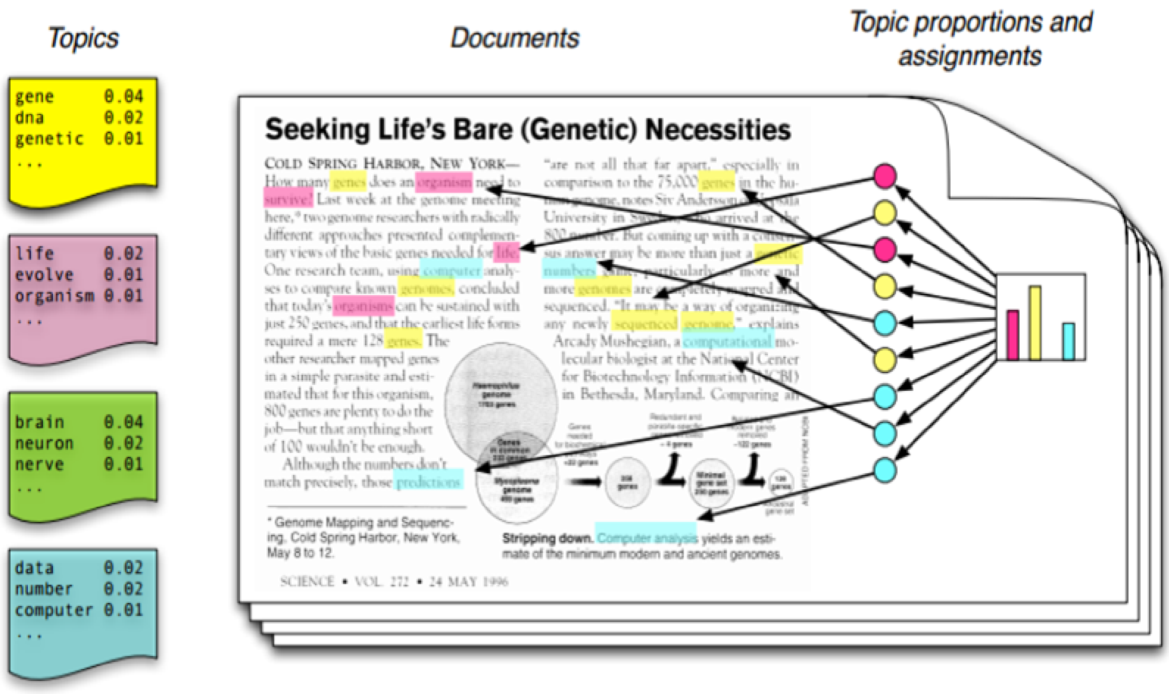


Figure 13: Document topic model visualization. https://www.semanticscholar.org/paper/Introduction-to-Probabilistic-Topic-Models-Blei/5f1038ad42ed8a4428e395c96d57f83d201ef3b3

framework for evaluating the effectiveness of a specific number of topic value, by analyzing four aspects:

- Segmentation: How much topics cover different content
- Probability estimation: Quantitative measurement of topics quality
- Confirmation measure: Determine quality as per some predefined standard and assign some number to qualify
- Aggregation: Combine all the quality numbers and compute the overall model quality

Hence, Topic coherence itself has no faculty of determining the most suitable number of topic for a given collection. Instead, it can be used as evaluation technique to compare several number of topics so to choose the best one among those that have been tested, in a hyper-parameter tuning scenario.

3.4.2 Entity extraction

Among the commercial solutions available, we considered Google Cloud services, in particular the Natural Language APIs. Among all the functions provided by those libraries, we were mostly interested in the Analyzing Entities functionality, to automatically extract entities and keywords from texts. The purpose of using this this tool would be categorizing documents based on the relevant information they contain, by mapping to each entity a set of documents where it can be found. Indeed, having such an indexing structure, would allow to quickly retrieve those documents that contain the same entities present in the query. As long as we are talking about a proprietary software, the processing approach is not publicly accessible, however, it is worth analyzing the characteristics and the shape of entity extraction results. After submitting text to the service, a short waiting of few seconds, varying with respect to text length, a JavaScript Object Notation (JSON) format file is returned to the user. This document, contains all the found entities along with some related classifying information such as its type, salience and metadata. Entity types are the following:

- UNKNOWN, when the entity does not belong to any other type
- PERSON, as 'judge' and 'Matteo Marziali'
- LOCATION, like 'dining room' and 'Milan'
- ORGANIZATION, such as 'Tribunal of Rome'
- EVENT, like 'bankruptcy'
- WORK_OF_ART
- CONSUMER_GOOD
- OTHER
- PHONE_NUMBER
- ADDRESS
- DATE
- NUMBER
- PRICE

The salience, consists in a floating point score number assigned to entities to estimate their centrality and importance in the provided text. It ranges from 0, less salient, to 1, highly salient. Finally, metadata represents qualifying information for the extracted entities. In practice this information mostly consists in Wikipedia Uniform Resource Locator (URL)s, except for specific entity types. For example prices come with the currency, phone numbers with the area code and others.

3.5 Summary

In this chapter, we presented the techniques, in the vast Text Retrieval literature, that we evaluated for achieving the goal of this thesis. In doing this, we described the algorithms, models and external tools we considered for processing documents, embedding text and performing document retrieval, the core search engine operations.

CHAPTER 4

JUDICIAL DOCUMENTS SEARCHING PROBLEM

In this chapter, we discuss how we selected the use case and we describe its aspects of relevance in a general scenario. Further on, we move to the description of the selected public dataset as well as its applicability in the context of the addressed problem.

4.1 The Scenario

As pointed out in Chapter I, search engines represent a wide class of software which groups programs that aim at performing ad-hoc retrieval tasks. The term ad-hoc lead us to two different considerations. On one hand, it suggests that the related IR challenge is applicable in many areas, on the other hand, it implies that such problems expect solutions that are strongly fitted to the application domain. In practice, these characteristics fall on search engines, whose components are subject to the peculiarities of the addressing task. In fact, documents can have many formats, different lengths, diverse languages and contents, as well as many other distinctive features. Surely, preprocessing aims at handling some of the most remarkable particularities of data, but their structure and the implementation requirements also have an impact on word embeddings selection, similarity measure adoption, indexing structure and ranking algorithms design.

The goal of this thesis is to develop a domain-specific search engine combining cognitive and keyword based approaches in order to maximize search effectiveness and minimize the effort required to train and configure it. On one hand, pure cognitive search techniques suffer from handling same words with different possible meanings. On the other hand, mere keyword based methods provide limited performance because of the lack of query context and intent understanding.

4.2 The Data

Moving in the provided scenario, aiming at developing a software of wide applicability range, we dedicated a great attention to the choice of the dataset by looking for specific text characteristics and by pursuing availability and accessibility criteria. In this selection process, the key decision factor has been the generality of data. By looking for documents with characteristics that can be commonly found writings from many ambits, in terms of information provided, length and structure of texts, allow us to develop a system of general scope, not limited to our specific scenario. Eventually, we decided to work on legal documents, precisely on judgements from the Italian Court of Cassation, freely available online on [12]. Precisely, we collected about 12000 reports, written in Italian, among Sentences, Ordinances and Decrees, from Civil Division, Penal Division, Labor Division and Joint Divisions dating from 2011 to 2017. This decision was driven by many considerations. First of all, according to project requirements, we were required to work with document images or scanned paper texts, file formats that would have required us to extract the content from the raw data in order to exploit it in the development process. Secondly, we were looking for texts with specific characteristics, both in terms of length and provided information, indeed, judicial documents encompass many topics in the judicial ambit and consist of several pages, in the order of tenths or even hundredths. Dealing

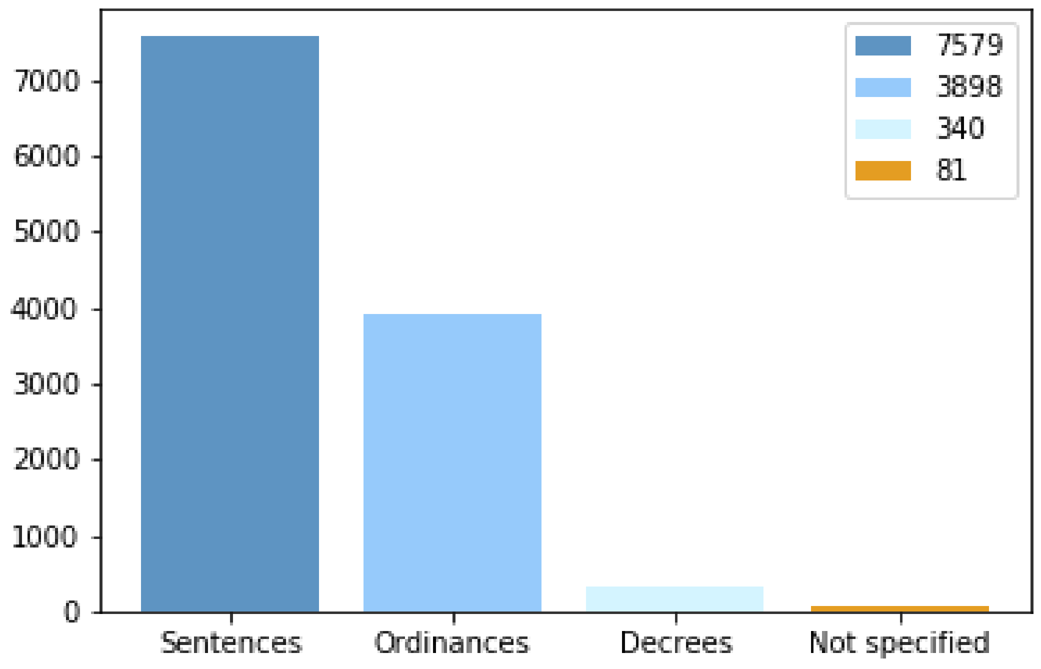


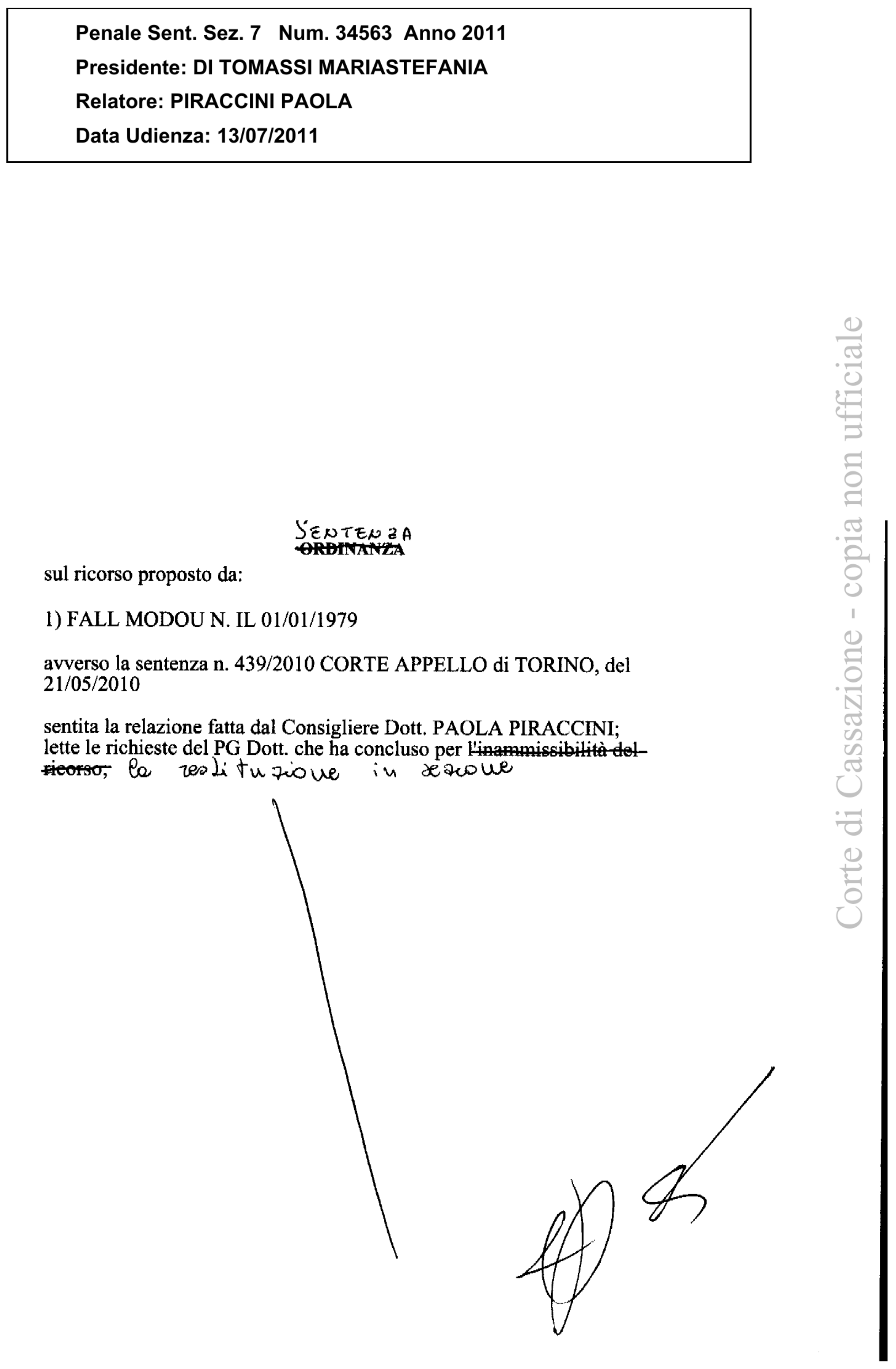
Figure 14: Distribution of documents in the collection.

with such verbose documents, indeed, makes possible to look for contextual information in addition to keyword-based search, aspect that validates the use of cognitive approaches. These factors allow us to validate our research work in the perspective of achieving similar results when we export the conceived model to a different domain, composed by a document collection that shares such general aspects of file length and arguments variety.

4.2.1 Data Exploration

Long way from inappropriately using the term Data Exploration, which is a well-known Data Analysis approach featuring specific tools and visualization algorithms, the purpose of this section is to point out some characteristics of our source documents that are commonly shared across the whole collection.

The gathered judgements can be analyzed from two different perspectives, as they present some shared aspects directly related to their judicial nature as well as others that are peculiar of the data source. Usually, judicial papers present a quite fixed structure with respect to the order and the organization of the content. Judgements, regularly, starts with a box which describes the category of the legal procedure, qualifies the commission president and speakers and showing the date of the process. The structure then proceeds with a keyword to distinguish whether we are dealing with either a Sentence, a Decree or an Ordinance and by presenting the involved parties. After this introductory part, there is the core of the report which is divided into two sections, the former explains the main aspects of the question that is under investigation, the latter provides an interpretation of the facts under inspection in judicial key, by analyzing the laws and the elements that lead to the final decision. Finally, there is the verdict, the punishments, place and date of the Process.



Penale Sent. Sez. 7 Num. 34563 Anno 2011
Presidente: DI TOMASSI MARIASTEFANIA
Relatore: PIRACCINI PAOLA
Data Udienza: 13/07/2011

SENTENZA
~~ORDINANZA~~

sul ricorso proposto da:

1) FALL MODOU N. IL 01/01/1979

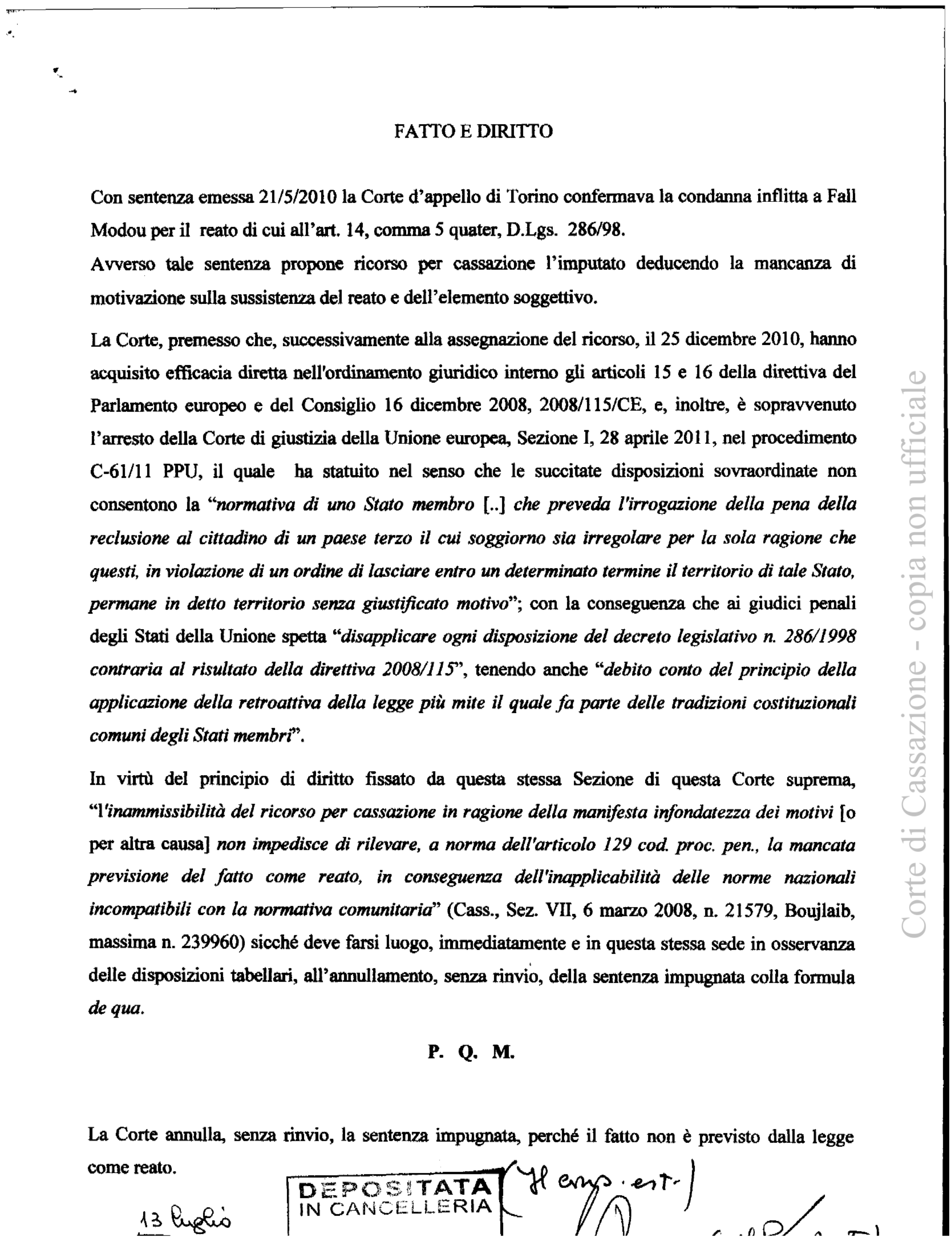
avverso la sentenza n. 439/2010 CORTE APPELLO di TORINO, del 21/05/2010

sentita la relazione fatta dal Consigliere Dott. PAOLA PIRACCINI;
lette le richieste del PG Dott. che ha concluso per l~~'inammissibilità del ricorso;~~ la restituzione in sezione

[illegible]

Figure 15: Sample Sentence pg.1
http://juriswiki.it/

Aside from the document structure, there are also fixed sentences and well-defined recurring forms to express concepts such as whether the accused are guilty or not and law references. These structural characteristics lead us to some considerations, indeed, having so many similar sentences in the collection can suggest us to adopt a frequency-based approach, like Term Frequency - Inverse Document Frequency (TF-IDF), so to favor rare words and sentences. On the other hand, recognizing these fixed-form phrases can ensure to produce an effective sentence-level tokenization. Another factor to consider is that, dealing with scanned paper documents, whose text has to be extracted using an OCR, some characteristics of judicial documents, such as specific layout and spacing, presence of stamps and signatures, as well as other noisy elements, can heavily affect the way OCR recognizes text, possibly leading to wrong characters and words understanding. Analyzing more in depth our source files content, we can also notice an intensive use of abbreviations and acronyms, a typical practice in judicial documents. This peculiarity of our documents, lead us to two considerations. Firstly, abbreviations and acronyms can cause to produce a wrong word graph when we generate text embeddings. For example, encoding vector for the abbreviation 'art.' meaning 'article', could be very similar to the embedding for 'art', intended as those human activities that produce visual, auditory or performing artifacts. Another potential issue of shortened words consists in their impact on the tokenization of text into sentences. Indeed, being usually ended by dots, such phrase elements could lead to misunderstand the actual sentence termination by interpreting the abbreviation delimiter as an ending mark. Moreover, sometimes acronyms can hide relevant information. For example, the term 'c.p.p.' meaning 'Penal Code' contributes to reduce the amount of information carried



FATTO E DIRITTO

Con sentenza emessa 21/5/2010 la Corte d'appello di Torino confermava la condanna inflitta a Fall Modou per il reato di cui all'art. 14, comma 5 quater, D.Lgs. 286/98.
Avverso tale sentenza propone ricorso per cassazione l'imputato deducendo la mancanza di motivazione sulla sussistenza del reato e dell'elemento soggettivo.

La Corte, premesso che, successivamente alla assegnazione del ricorso, il 25 dicembre 2010, hanno acquisito efficacia diretta nell'ordinamento giuridico interno gli articoli 15 e 16 della direttiva del Parlamento europeo e del Consiglio 16 dicembre 2008, 2008/115/CE, e, inoltre, è sopravvenuto l'arresto della Corte di giustizia della Unione europea, Sezione I, 28 aprile 2011, nel procedimento C-61/11 PPU, il quale ha statuito nel senso che le succitate disposizioni sovraordinate non consentono la "*normativa di uno Stato membro* [..] *che preveda l'irrogazione della pena della reclusione al cittadino di un paese terzo il cui soggiorno sia irregolare per la sola ragione che questi, in violazione di un ordine di lasciare entro un determinato termine il territorio di tale Stato, permane in detto territorio senza giustificato motivo*"; con la conseguenza che ai giudici penali degli Stati della Unione spetta "*disapplicare ogni disposizione del decreto legislativo n. 286/1998 contraria al risultato della direttiva 2008/115*", tenendo anche "*debito conto del principio della applicazione della retroattiva della legge più mite il quale fa parte delle tradizioni costituzionali comuni degli Stati membri*".

In virtù del principio di diritto fissato da questa stessa Sezione di questa Corte suprema, "*l'inammissibilità del ricorso per cassazione in ragione della manifesta infondatezza dei motivi* [o per altra causa] *non impedisce di rilevare, a norma dell'articolo 129 cod. proc. pen., la mancata previsione del fatto come reato, in conseguenza dell'inapplicabilità delle norme nazionali incompatibili con la normativa comunitaria*" (Cass., Sez. VII, 6 marzo 2008, n. 21579, Boujlaib, massima n. 239960) sicché deve farsi luogo, immediatamente e in questa stessa sede in osservanza delle disposizioni tabellari, all'annullamento, senza rinvio, della sentenza impugnata colla formula *de qua*.

P. Q. M.

La Corte annulla, senza rinvio, la sentenza impugnata, perché il fatto non è previsto dalla legge come reato.

13 luglio

DEPOSITATA
IN CANCELLERIA

(Il cons. est.)

[illegible]

Figure 16: Sample Sentence pg.2
http://juriswiki.it/

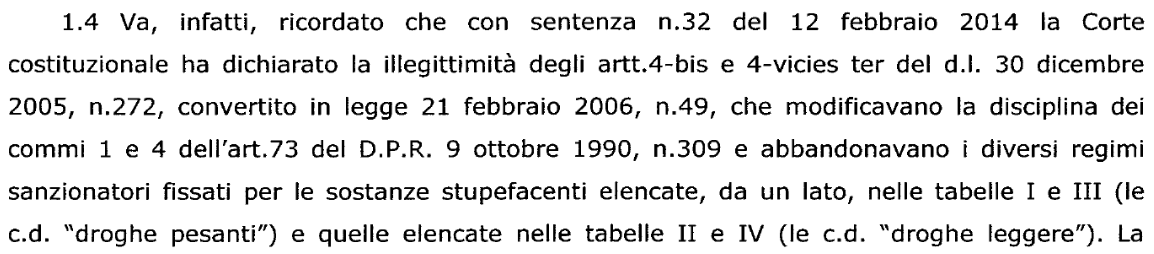
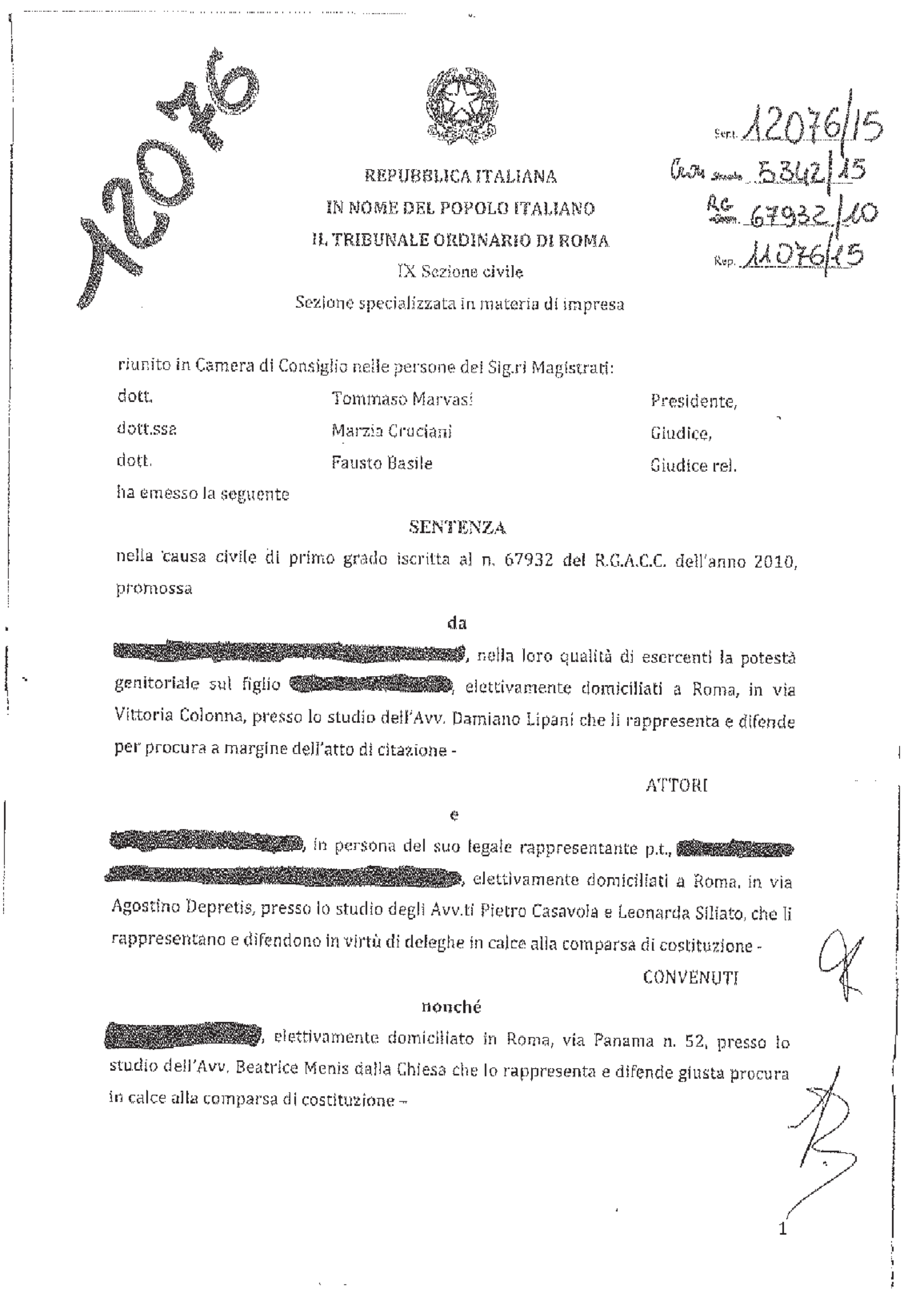

1.4 Va, infatti, ricordato che con sentenza n.32 del 12 febbraio 2014 la Corte costituzionale ha dichiarato la illegittimità degli artt.4-bis e 4-vicies ter del d.l. 30 dicembre 2005, n.272, convertito in legge 21 febbraio 2006, n.49, che modificavano la disciplina dei commi 1 e 4 dell'art.73 del D.P.R. 9 ottobre 1990, n.309 e abbandonavano i diversi regimi sanzionatori fissati per le sostanze stupefacenti elencate, da un lato, nelle tabelle I e III (le c.d. "droghe pesanti") e quelle elencate nelle tabelle II e IV (le c.d. "droghe leggere"). La

Figure 17: Sentence portion with many acronyms and abbreviations
http://juriswiki.it/

by the text, or at least reduces the number of people which are able to access that notion, by assuming that who is reading knows how to expand the provided opening letters. Issues related to acronyms and abbreviations are amplified by the fact that across our collection there are diverse shortened forms for the same words, characteristics that, along with misspelled errors due to the OCR, widely increase the number of cases to be handled.

Moving forward with the analysis, it is worth noting that few documents suffered a phenomenon known as name dimming. This handle refers to the practice of blacking out names, either of people or companies, from judgments so to ensure the privacy of the subject. However, from such erased images, text extraction by means of OCR is very difficult. Usually, manual erasures produce an empty area with a color that is different from the page background, causing the OCR to capture meaningless text when passing over these opaque regions.



12076

Sent. 12076/15
Cron. 5342/15
RG 67932/10
Rep. 11076/15

REPUBBLICA ITALIANA
IN NOME DEL POPOLO ITALIANO
IL TRIBUNALE ORDINARIO DI ROMA
IX Sezione civile
Sezione specializzata in materia di impresa

riunito in Camera di Consiglio nelle persone dei Sig.ri Magistrati:

dott.	Tommaso Marvasi	Presidente,
dott.ssa	Marzia Cruciani	Giudice,
dott.	Fausto Basile	Giudice rel.

ha emesso la seguente

SENTENZA

nella causa civile di primo grado iscritta al n. 67932 del R.G.A.C.C. dell'anno 2010, promossa

da

[redacted], nella loro qualità di esercenti la potestà genitoriale sul figlio [redacted], elettivamente domiciliati a Roma, in via Vittoria Colonna, presso lo studio dell'Avv. Damiano Lipani che li rappresenta e difende per procura a margine dell'atto di citazione -

ATTORI

e

[redacted], in persona del suo legale rappresentante p.t., [redacted], elettivamente domiciliati a Roma, in via Agostino Depretis, presso lo studio degli Avv.ti Pietro Casavola e Leonarda Siliato, che li rappresentano e difendono in virtù di deleghe in calce alla comparsa di costituzione -

CONVENUTI

nonché

[redacted], elettivamente domiciliato in Roma, via Panama n. 52, presso lo studio dell'Avv. Beatrice Menis dalla Chiesa che lo rappresenta e difende giusta procura in calce alla comparsa di costituzione -

[illegible]

[illegible]

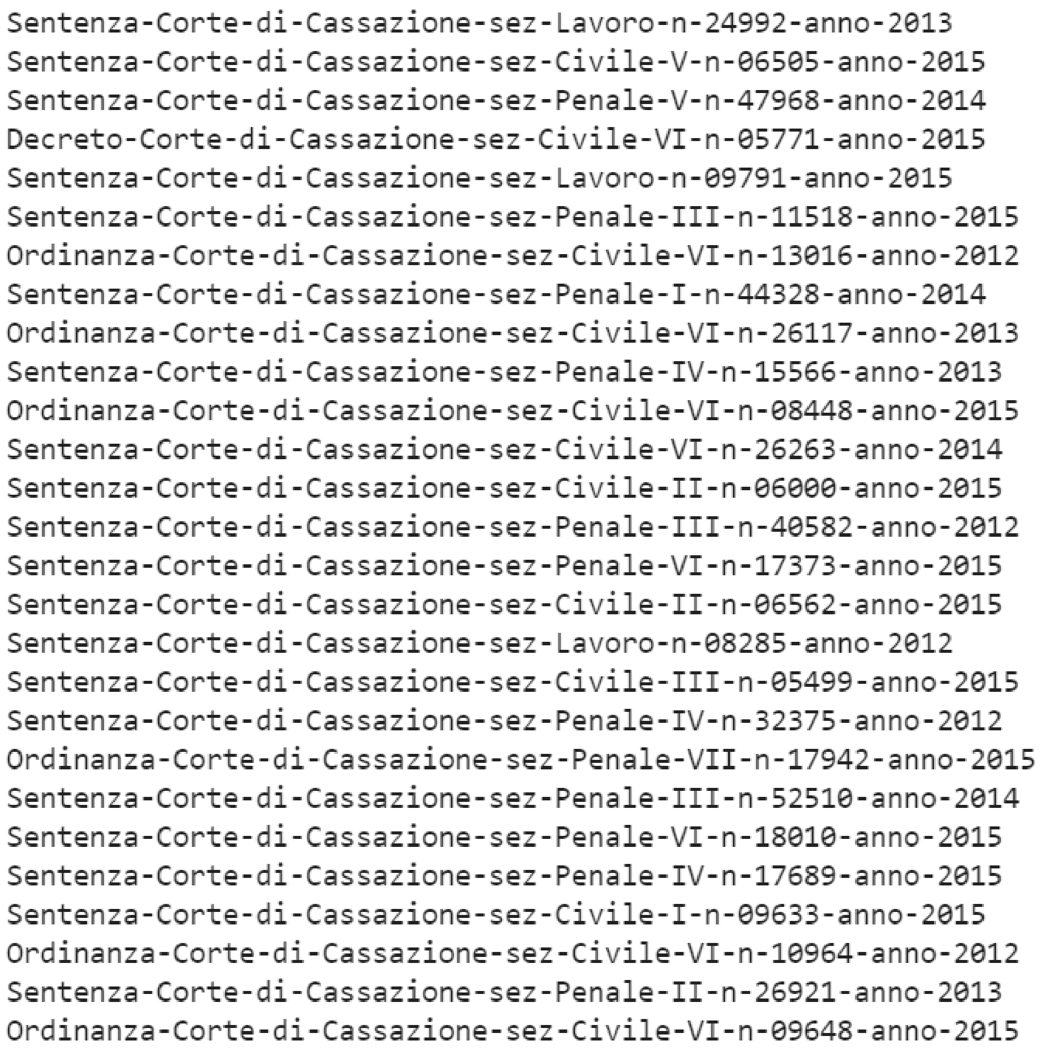
1

Figure 18: Example of jammed Sentence
http://juriswiki.it/

Still looking at the content of our source files, stands out that dates are registered with different formats even in the same document. Remembering that Italian dates want the day, the month and the year in this order, in the dataset there can be identified four configurations: GG/MM/AAAA, GG\MM\AA, GG-MM-AA and GG.MM.AAAA. This variety in time arrangements, surely, adds complexity in text processing and in particular the dotted pattern is of trouble in splitting the documents in sub-entities. Finally, it is worth analyzing the file names in our collection, as they are very descriptive. Apart from few documents related to the more recent processes which have peculiar titles, each file name is divided in four parts and each one gives relevant contextual information. At the very beginning of the string we can find the type of the report, distinguishing among Sentences, Decrees and Ordinances. Then, there is a reference to the responsible judicial body, which is always the Court of Cassation in our case, along with the Division of the process. Towards the end of the name string we are given two more fundamental information, firstly the unique identifier of the process within the provided Division, secondly the year in which it was conducted. This brief overview on the characteristics of the collected documents is crucial to understand the challenges and the issues that we encountered in this work, as well as to figure out many of the solutions adopted, that are being described in the next chapter, by relating the data with the faced problem, to clearly define the complete scenario we are moving within.

4.3 Domain Requirements

We already summarized the purpose of this research work talking about the scenario, however it is essential to have a closer look to the faced problem and its challenges, so to precisely



```
Sentenza-Corte-di-Cassazione-sez-Lavoro-n-24992-anno-2013
Sentenza-Corte-di-Cassazione-sez-Civile-V-n-06505-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-V-n-47968-anno-2014
Decreto-Corte-di-Cassazione-sez-Civile-VI-n-05771-anno-2015
Sentenza-Corte-di-Cassazione-sez-Lavoro-n-09791-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-III-n-11518-anno-2015
Ordinanza-Corte-di-Cassazione-sez-Civile-VI-n-13016-anno-2012
Sentenza-Corte-di-Cassazione-sez-Penale-I-n-44328-anno-2014
Ordinanza-Corte-di-Cassazione-sez-Civile-VI-n-26117-anno-2013
Sentenza-Corte-di-Cassazione-sez-Penale-IV-n-15566-anno-2013
Ordinanza-Corte-di-Cassazione-sez-Civile-VI-n-08448-anno-2015
Sentenza-Corte-di-Cassazione-sez-Civile-VI-n-26263-anno-2014
Sentenza-Corte-di-Cassazione-sez-Civile-II-n-06000-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-III-n-40582-anno-2012
Sentenza-Corte-di-Cassazione-sez-Penale-VI-n-17373-anno-2015
Sentenza-Corte-di-Cassazione-sez-Civile-II-n-06562-anno-2015
Sentenza-Corte-di-Cassazione-sez-Lavoro-n-08285-anno-2012
Sentenza-Corte-di-Cassazione-sez-Civile-III-n-05499-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-IV-n-32375-anno-2012
Ordinanza-Corte-di-Cassazione-sez-Penale-VII-n-17942-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-III-n-52510-anno-2014
Sentenza-Corte-di-Cassazione-sez-Penale-VI-n-18010-anno-2015
Sentenza-Corte-di-Cassazione-sez-Penale-IV-n-17689-anno-2015
Sentenza-Corte-di-Cassazione-sez-Civile-I-n-09633-anno-2015
Ordinanza-Corte-di-Cassazione-sez-Civile-VI-n-10964-anno-2012
Sentenza-Corte-di-Cassazione-sez-Penale-II-n-26921-anno-2013
Ordinanza-Corte-di-Cassazione-sez-Civile-VI-n-09648-anno-2015
```

Figure 19: Sample of document filename strings

TABLE I: Goal, Requirements And Functions For Input Documents.

	Goal
G1	Developing a Search Engine that works on top of scanned documents.
	Requirements
R1	User can feed the search Engine with a collection of images
R2	User can add files to the collection after software deploy
	Functions
F1	Uploading function for a document batch for search engine boot.
F2	Uploading function for a single document without affecting search engine status and performances.

define the addressed task in terms of detailing all the goals we were required to achieve and the related requirements and functions. The final purpose of this thesis is the development of a Search Engine, however, the end product has to be aligned with some project requirements. In this section, we are exploring those project specifications by considering each search engine component, as well as the input documents, separately.

4.3.1 Input Documents

In the context of this work, we talk about input documents referring to the collection of files on top of which we are developing the search engine. Project requirements concerning input documents are mainly related to the source file format. Indeed, the search engine to be will have to handle scanned texts, commonly stored as either images or Portable Document Format (PDF)s. In practice, dealing with this constraint means including, in the software to be developed, functions to upload data images and to extract their content.

TABLE II: Goal, Requirements And Functions For Document Processing.

Goal	
G2	Extracting text from source images.
Requirements	
R3	The system is able to process source images to produce correspondent textual versions.
Functions	
F3	Processing function to perform OCR text extraction from a set of input images.

4.3.2 Document processing

Considering the task of processing documents, we have to comply with some requirements concerning both the shape of the files to be processed by the software and the expected output of the search. As we saw in the previous section, input files consist of images, however in order to achieve the proposed goal, in short, building a search engine that combines cognitive and keyword based approaches, we need to extract the text from the pictures. Indeed, in order to exploit word characteristics, such as its, its meaning, its role and position within a sentence, its frequency in the document and in the collection and others, we need to have file contents in their native string type.

4.3.3 Word Embedding

As of word embedding, there were no specific functional requirements. The implicit constraint for word embeddings, is strongly related to the requirements for visualizing the search results, which consists in asking for presenting the user the most relevant sentences with re-

TABLE III: Goal, Requirements And Functions For Word Embedding.

Goal	
G3	Producing sentence-level embedding vectors.
Requirements	
R4	The system is able to generate sentence encoding vector given a document text.
Functions	
F4	Encoding function that either takes in input a sentence string and outputs a correspondent vector or takes in input a set of word vectors and outputs a single vector, combination of the inputs.

spect to the query. Indirectly, this expectation affects word embedding in what concerns the granularity of the text to encode. As long as the system is required to provide the user with sentences relevant to his/her input string, we are implicitly required to weight this relevance concept at phrase level, thus we need to find a way to produce encoding vectors for sentences either directly or indirectly. Directly, by building a model that, fed with string phrases, outputs their embedding vector, indirectly, by using a model that encodes single words and then combining the embedding of the terms composing a sentence, so to have, still, a vector for such group of words.

4.3.4 Retrieval and Ranking

Concerning the retrieval component, responsible of filtering out documents from the collection, in order to reduce the number of candidate sentences to retrieve, we can identify two functional requirements. The trivial one consists in having this component in the search engine, indeed, theoretically these software can work without, in practice the impact of this component

TABLE IV: Goal, Requirements And Functions For Retrieval And Ranking.

Goal	
G4	Performing retrieval on a subset of the collection documents.
Requirements	
R5	The system is able to filter out documents, retrieving a limited set containing those relevant to the input query.
Functions	
F5	Filtering functions to select the documents of the collection that most likely contain results relevant with respect to the query.

is strongly related to the collection size and the computations needed to establish the relevance of documents with respect to the query. In fact, one of the most relevant non functional requirements for search programs refers to the response time, users would love to have immediate answers to their query. This said, in business search engines this component is no more matter of requirements, as it is trivially granted.

4.3.5 Visualization

In the task of presenting search results to the user, we have a relevant constraint to comply with. Indeed, the system is expected to return a set of sentences, selected among the retrieved documents as the ones that are most relevant to the query, together with the names of the files where these phrases can be found. This requirements is the result of wanting the user to get straight to the point of his/her search, in fact, returning the sole file can be meaningless in case it is composed of many pages. On the other hand, the single most similar period is not enough. In case of generic, i.e. non-literal, queries, such as 'judgments of 2014', the concept of

TABLE V: Goal, Requirements And Functions For Result Visualization.

Goal	
G5	Returning to the user the most similar document sentences with respect to his/her query.
G6	Including in search results the name of retrieved files, as well as their links.
Requirements	
R6	The system is able to present the user the sentences that are most similar to the query and return them as search results, along with the file name of the containing document.
Functions	
F6	Post processing functions which take search results in input and compute and output the most similar sentences with respect to the query and the related files.

providing the user with the most relevant sentences is pointless, or rather detrimental. Clearly, presenting file names to the user is more significant in similar circumstances.

4.4 Summary

In this chapter, we discussed the characteristics of the scenario within we intend to apply our solution. In particular, we described the rationale behind the choice of the domain and correspoding dataset, we deeply analyzed the adopted data and, finally, we formally defined the project requirements we were subject to.

CHAPTER 5

PROPOSED SEARCH ENGINE PIPELINE

In this chapter, we describe in depth our proposed solution, retracing the outline of Chapter 2. We start with a brief overview to recall the main search engine components and clarify how they integrate in the system. Then, we present in detail our approach for each of these components. Finally, we analyze our product from the end-user perspective, by illustrating the search process.

5.1 Overview

Search engines, regardless to their domain, commonly feature typical components. Considering the most authoritative example, the popular search Engine created by Google founders, Sergey Brin and Lawrence Page, and described in [13], we basically find the same components presented in Chapter I. Surely these units evolved and improved since 1998, publication year of the cited paper, other components have been added to recent search engines, however, surprisingly, the backbone of this software family never changed. Parts in question are:

- **Crawler**, responsible of collecting documents on top of which the search engine works and extracting their text. We can separate its functions in:
  - Document collecting
  - Document processing - text extraction

- **Indexer and sorter**, which, according to [13], perform the operations that we declined in three phases:
  - Document processing - text elaboration
  - Word embedding
  - Retrieval
- **Ranker**, responsible of ranking search results

In addition, we considered also the **Visualization** step, during which the ranked results are formalized to comply with presentation requirements. The proposed schema refers to the software development, indeed, we analyze the processes and the algorithms involved in creating the resources that run the user-destined end product. In the end, we explain how the components we dwelt on in this chapter work together in performing the search and how system maintenance is carried on.

5.2 Document collecting

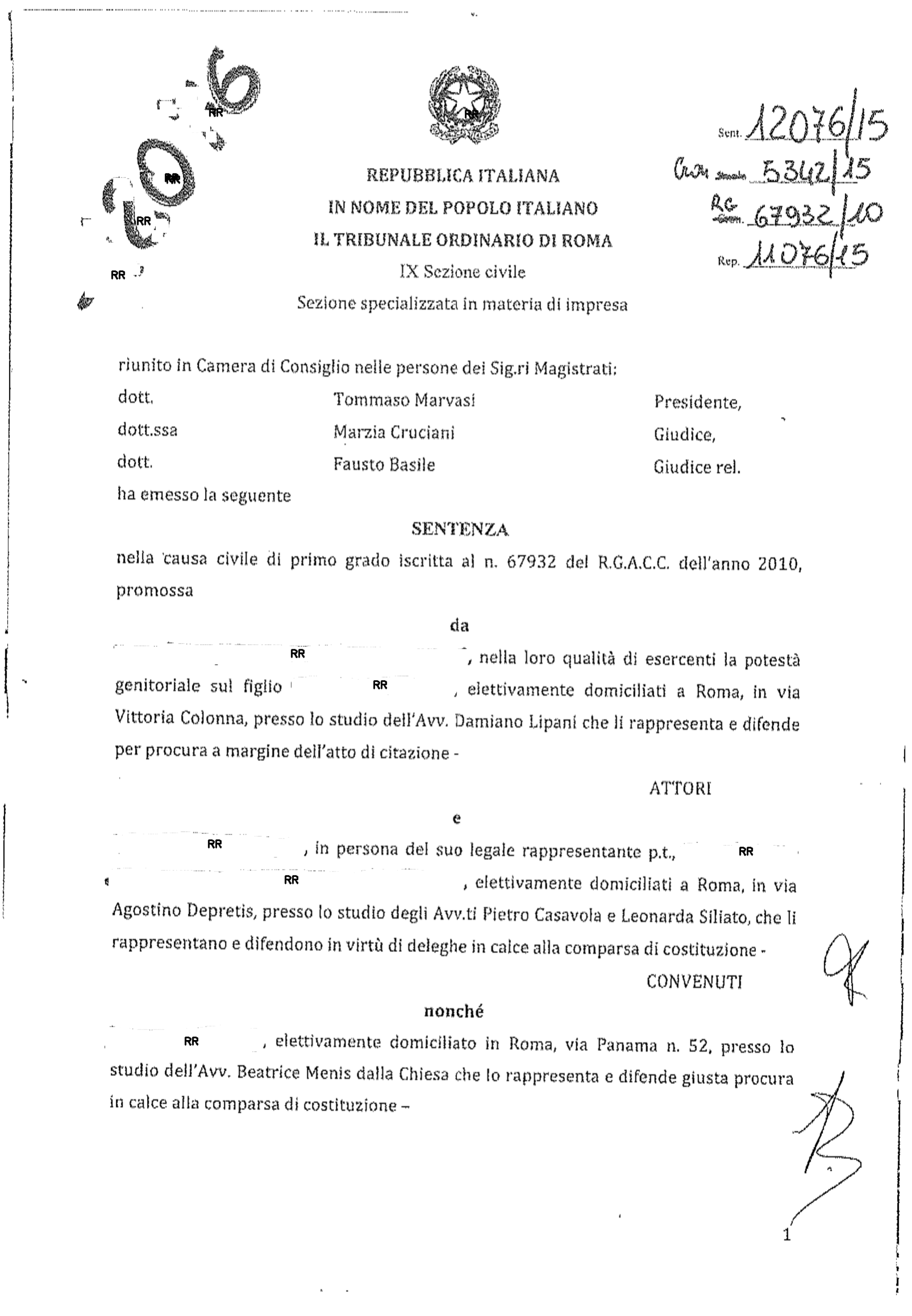
Document collecting refers to the task of gathering texts upon which performing the search. In [13], this component is called Crawler, a script to automatically find documents, extract text and store it. This name, is frequently used in web domain, as it commonly refers to download web pages and extract their content useful for the application in question. As of our context, we briefly described document collecting process in Chapter III, here we analyze it in depth. For crawling files, after having found a source website, [12], we have built a multithreading script to download PDFs of scanned documents, avoiding duplicates. Downloaded data are stored in apposite archives where they can be accessed for further operations.

5.3 Document processing

Document processing refers to those activities necessary to prepare source files for downstream tasks, in particular, during this step we aim at elaborating input data in order to make them suitable for achieving the overall goal. Sometimes raw collected data are already usable, however, generally, preprocessing of the input leads to better final results. Actually, we grouped into document processing two diverse jobs, text extraction and text elaboration. The former addresses the problem of converting collected images into string-format texts, the latter manages to either enhance the characteristics or fix the issues emerged from data exploration. According to this distinction, we analyze the applied methodologies individually.

5.3.1 Text Extraction

Extracting text from images represents a complex operation. As we already pointed out in Chapter I, among the open source software, Tesseract has the best performances. However, as shown in [14], the quality of the extracted text is heavily affected by image processing. Fortunately, the plain 'Tesseract' v4.0.0. gave us excellent results on most of our source documents, except for those affected by name dimming. In Chapter III, we already mentioned that image files subject to name dimming would have been hardly 'understood' by Tesseract. Indeed, what happened was that these blobs generated for blacking names out were converted by the OCR engine into random sequences of characters.



Sent. 12076/15
Cron. 5342/15
R.G. 67932/10
Rep. 11076/15

REPUBBLICA ITALIANA
IN NOME DEL POPOLO ITALIANO
IL TRIBUNALE ORDINARIO DI ROMA
IX Sezione civile
Sezione specializzata in materia di impresa

riunito in Camera di Consiglio nelle persone dei Sig.ri Magistrati:

dott.	Tommaso Marvasi	Presidente,
dott.ssa	Marzia Cruciani	Giudice,
dott.	Fausto Basile	Giudice rel.

ha emesso la seguente

SENTENZA

nella causa civile di primo grado iscritta al n. 67932 del R.G.A.C.C. dell'anno 2010, promossa

da

RR, nella loro qualità di esercenti la potestà genitoriale sul figlio RR, elettivamente domiciliati a Roma, in via Vittoria Colonna, presso lo studio dell'Avv. Damiano Lipani che li rappresenta e difende per procura a margine dell'atto di citazione -

ATTORI

e

RR, in persona del suo legale rappresentante p.t., RR RR, elettivamente domiciliati a Roma, in via Agostino Depretis, presso lo studio degli Avv.ti Pietro Casavola e Leonarda Siliato, che li rappresentano e difendono in virtù di deleghe in calce alla comparsa di costituzione -

CONVENUTI

nonché

RR, elettivamente domiciliato in Roma, via Panama n. 52, presso lo studio dell'Avv. Beatrice Menis dalla Chiesa che lo rappresenta e difende giusta procura in calce alla comparsa di costituzione –

[illegible]

1

Figure 20: Processed version of Figure 18 image.

In order to leverage the impact of name dimming on text extraction performances, we studied an image processing method to obtain better performances with Tesseract. Basically, we created a script that, given an image with blacked out text areas, is able to identify these blobs and to lighten them, aiming at aligning their color to the white background of the page. Under these circumstances, the OCR engine gives good performances, but the extracted text quality decreases, as words originally interleaved by names result adjacent. Thus, we replaced the whitened shapes with a special marker, the RR character, clearly visible in Figure 20, so to allow further text elaboration. According to the text extraction described above, the result for the page shown in Figure 20 is the following:

en nen N

REPUBBLICA ITALIANA nds) Ea A I ì Ae IN NOME DEL POPOLO ITALIANO o

OLGA, Alè 4] ALE ORDINARIO DI ROMA IL TRIBUN N e AMOTE 5 RR 2 IX

Sezione civile

Sezione specializzata în materia di impresa

riunito în Camera di Consiglio nelle persone dei Sig.ri Magistrati:

dott, Tommaso Marvasi Presidente, dott.ssa Marzia Cruciani Giudice, dott. Fausto Basile

Giudice rel.

ha emesso la seguente SENTENZA, nella causa civile di primo grado iscritta al n. 67932

del R.G.A.C.C. dell'anno 2010, promossa da

RR ", nella loro qualità di esercenti la potestà

genitoriale sul figlio ' RR . elettivamente domiciliati a Roma, in via Vittoria Colonna,

presso lo studio dell'Avv. Damiano Lipani che li rappresenta e difende

per procura a margine dell'atto di citazione -

ATTORI e RR , in persona del suo legale rappresentante p.t., RR ' RR : elettivamente

domiciliati a Roma, in via

Agostino Depretis, presso lo studio degli Avv.ti Pietro Casavola e Leonarda Siliato, che li

rappresentano e difendono in virtù di deleghe in calce alla comparsa di costituzione -

CONVENUTI nonché i RR . elettivamente domiciliato in Roma, via Panama n. 52, presso

lo studio dell'Avv. Beatrice Menis dalla Chiesa che lo rappresenta e difende giusta procura

```
in calce alla comparsa di costituzione —
```

These lines above consist in the actual output of our text extraction processing technique for documents. The unusual layout, in terms of spacing and newlines, is strongly related to the page layout of Figure 20. Despite an overall satisfying extraction result, the processed text presents some meaningless words, or better, sequences of characters, directly attributable to the presence of stamps and handwritten signs and numbers in the document.

5.3.2 Text Elaboration

Text elaboration concerns the task of modifying documents content in order to enhance their performances in downstream activities. It is worth noting that text can be altered in many ways, generally the applied techniques are strongly correlated to the operations to come, especially word embedding. Also, aside from these approaches, cited in Chapter I, such as stop-words removal and stemming, well-known in literature, ad-hoc techniques can be applied to deal with text characteristics. As long as we experiment different embedding methods, we also perform diverse text processing for each encoding approach. However, we present here those practices that have been applied regardless to the word embedding procedure. We, then, integrate these methods when we consider each specific encoding approach in the next section. The rationale behind each text elaboration has to be found either in the data exploration phase or in the requirements. In constraints Table III and Table V, we can find the reasons that led us to tokenize text into sentences. This reasonable idea, however, is of complex implementation. Dealing with OCR, indeed, makes texts particularly noisy, in practice, sometimes punctuation and characters are misread and whitespaces are missed, making difficult to determine actual period terminations. Moreover, working on judicial documents involves other issues. Massive presence of acronyms and abbreviations, as we saw in the Data Exploration section, introduces misleading dots that can be confused with full stop marks. In order to leverage this issue, we studied a way to substitute acronyms and abbreviations with their corresponding extended form. In doing so, we built a dictionary to map such shortened words to the equivalent unfolded terms and, through a look-up and replace technique, we produced a new collection of documents. Coming back to the task of dividing texts into sentences, as long as results obtained using authoritative Python libraries were unsatisfactory, we decide to integrate them with handwritten algorithms. In practice, we handle problematic cases such as ordered lists and jurisprudence articles identifiers, like '633/41'. Particularly the latter, when appear as last word of the period, prevented to recognize the adjacent dot as full-stop due to the presence of the slash mark. Another adjustment made refers to the name dimming issue. As we saw earlier, we replaced erasures in text with the 'RR' mark (Figure 20). Then, during text elaboration phase, aiming at preserving the train of thought, we changed the 'RR' with the '[OMISSIS]' symbol, commonly used in place of actual names in judicial documents. Here we presented the general algorithms applied for preparing text for downstream tasks, however it is worth recalling that further processing operations can be performed based on the specific embedding technique directions for use.

5.4 Word Embedding

Embedding text into a numerical representation is crucial to compute similarity among sentences. Indeed, string-format shape prevents to infer textual properties such as semantic and syntactic distance. It is worth highlighting that word embedding represents the core search engine component with respect to the purpose of our thesis, as long as during this process it is possible to encode keyword-based and context-based information. Considering the critical role of embeddings, we decided to try diverse approaches and then evaluate the performances of each one. In particular, since we are interested in proving that our approaches provide better results than state-of-the-art technologies, we decided to consider fastText the term of reference. Other approaches consist in combinations of the evaluated embedding methods, skip-thoughts and ELMo, as well as the well-known TF-IDF paradigm and fastText itself. It is worth noting that, regardless to the embedding method, the problem addressed consists in learning unsupervised vector representations, which means generating text encoding without having known reference embeddings. Also, in order to exhaustively fit our collection and effectively capture domain specific information, we train each of the cited model on our corpus. Here we analyze each approach in depth, defining model parameters and detailing its input and output characteristics for the following techniques:

- fastText
- fastText & TF-IDF
- fastText & ELMo

- fastText & ELMo & TF-IDF
- Skip-thoughts
- fastText & Skip-thoughts

5.4.1 fastText

Representing words by vectors has become a popular approach in modern machine learning tasks involving text. This technique allows to capture and encode non-trivial features of languages such as semantics and syntactic word proprieties. In this scenario, fastText represents a solid baseline. As we saw in Chapter II, it works by generating a n-gram vocabulary to map each set of character encountered in the collection to a corresponding vector of numbers. In order to reduce vocabulary size, we decide to preprocess documents using the already presented stemming technique, aiming at reducing terms to their root; we also remove stop-words and punctuation except for full-stops, useful for sentence tokenization. As of model parameters, there are three constants to be defined, the size of the n-gram vectors which also determine the dimensional space, the window size for the n-grams and the minimum word occurrence to be considered in the model. In theory, the larger the vector size the more the information captured, actually, the dimension suggested in the official fastText tutorial [15] for unsupervised approach ranges from 100 to 300, as lager values would slow down training phase and would require more data. Under these circumstances we decided to build 300-dimensional vectors for n-grams of 3 characters and considering each word regardless to its occurrence count. This said, we can now explain how we encoded sentences into vectors using fastText. Since fastText model, given a word, returns its 300-dimensional numbered representation, we decided to encode sentences by simply averaging the vectors corresponding to each word in the period in question. Then we stored for each document in the collection the vectors corresponding to its sentences, for fast document-query similarity evaluation. This model is very fast to train and features a vocabulary of about 60000 n-grams, capable to generate embeddings for words unseen in the collection, so that every query can be effectively encoded.

5.4.2 fastText & TF-IDF

In retrieval tasks, results are biased, in general, by words with high occurrence across the corpus. Surely, texts contains several irrelevant words, necessary for achieving a complete meaning, but of no interest in qualifying and discriminate document content. These terms of general use like pronouns, adverbs and other grammar items, are recurrent in the whole collection, as they are widely used in writings. The purpose of Term Frequency - Inverse Document Frequency (TF-IDF) consists in assigning a score proportional to the number of times that the word appear in the document in question, and inversely proportional to its frequency in the whole corpus. The idea behind this algorithm is to favor words that occur many times within a single document and have a low frequency with respect to the collection. We make use of this concept by creating a TF-IDF model and applying this in our context. Encoding words with plain TF-IDF is a relatively old approach which have been overcame by newer techniques like fastText. However, TF-IDF coupled with fastText can positively affect the vector representation, by leveraging the impact of irrelevant words and increasing that of rare terms within the averaged sentence vector. Indeed, the difference of this combined approach with respect to the sole fastText technique, consists in weighting each word vector with respect to that word TF-IDF score. In practice, this is done by multiplying each term for the TF-IDF score related to that expression before to compute the averaged sentence representation. This is performed by computing a word-document matrix, where each cell contains the TF-IDF score for the corresponding term-document pair, storing it and accessing it to weight the word vectors. Surely, as long as estimating the TF-IDF score is not possible for words not present in the collection, in such cases the term vector is not weighted, but the sole fastText encoding is considered.

5.4.3 fastText & ELMo

The idea of creating an embedding technique which combines both fastText, a powerful model in learning syntactic word features, and ELMo, an effective technique to capture term semantics, arise from a previous failed attempt. Indeed, firstly we tried to use the plain ELMo model, then, disappointed by the performances, we decided to combine it with the promising fastText. Initially, from a casual evaluation, the newly created method showed performances comparable with the sole fastText, so we decided to bring this approach forward for further analysis. As long as we already explored core aspects of fastText, we are broadly examining ELMo. We defined in chapter II ELMo architecture and functioning, however, we still have many considerations to do. First of all, following the official ELMo manual on github, the well-known code sharing platform, differently from fastText, we do not perform any other text elaboration except for those operations explained in the section 5.3.2. The required input file formats for this algorithm is a set of files containing random sampled sentences across all the collection documents, so, we decide to organize our corpus of about 300'000 phrases into 100 files. ELMo model is quite complex and involves many parameters, thus, we train it using Google Compute Engine, the Google platform to buy and use computational power, in the form of virtual machines. Thanks to cloud resources, we reduce the training time from several days to few hours. Once we have a trained ELMo model, we store it and we use it in order to encode sentences into vectors of 1024 dimensions, using the same parameters discussed in [11], the original paper. Under these circumstances, we combine the two models by concatenating the fastText and the ELMo vectors, obtaining a new vector of 1024 + 300 dimensions for each sentence.

5.4.4 fastText & TF-IDF & ELMo

The fourth model we evaluate features both the ability to learn morphological word characteristics, coupled with TF-IDF score to enhance rare words, guaranteed by fastText, and the capability to capture context-based word properties provided by ELMo. The rationale behind the combination of this two approaches, three if we consider TF-IDF separately, has to be found in the few lines above: joining the power of keyword-based fastText approach and context-based ELMo technique. Also, as long as fastText gives better performances when combined with TF-IDF, we decide to test this configuration together with ELMo. We already analyzed the three techniques, however, it is worth considering how they are exploited together. Firstly, we compute 300-dimensional vectors using the technique explained in 5.4.2, then we apply the same method described in 5.4.3, using the fastText & TF-IDF vectors in place of the plain fastText. Thus, we generate vectors of 1024 + 300 dimensions. However, the impact of TF-IDF on the combined vector is problematic, as it makes the 300 components attributable to fastText of several order of magnitude less than the values in the ELMo portion. In practice, this difference of values, causes that fastText influence was weightless, making this combination totally useless. Hence, aiming at solving this issue, we scale the fastText components to the range of ELMo ones, obtaining a uniform vector that guarantees very promising performances.

5.4.5 Skip-thoughts

Skip-thought vectors consist in an alternative approach for learning context-based sentence representations. Differently from previous techniques, however, the purpose of this model is to directly compute sentence embeddings, a diverse approach in addressing our task. Also, in contrast to ELMo, in order to train Skip-thoughts model we perform stemming as well as stop-words and punctuation removal on collection documents. Considering the Neural Network Architecture and the functioning of skip-thoughts, explained in 3.3.2, it is clear that we need to organize all the sentences in our collection into triplets of adjacent phrases. Thus, we generate files containing several sets of three items, the current, the previous and the next sentences, in fact, this was the format required to feed files to the NN. Skip-thoughts, as well as ELMo, are models based on complex network architectures, which, thus, require high-performing computational resources. Hence, similarly to the ELMo training process, skip-thoughts is trained using Microsoft Azure cloud services. Once we have a trained model, we can encode sentences into 2048-dimensional vectors, in accordance with the model parameters defined in [8]. Roughly analyzing the obtained model, we can infer that it works quite well if the query contains word that are present in the created dictionary, otherwise performances are terrible. This aspect, lead us to consider to apply skip-thoughts combined to a complementary keyword-based approch, such as fastText.

5.4.6 fastText & Skip-thoughts

The last encoding method we apply features both the pure syntactical fastText model and the typical semantic approach of Skip-thoughts. In particular, given that Skip-thoughts model suffers out-of-vocabulary words and fastText is able to effectively encode unseen terms, thanks to its ability of capturing subword information, joining them seems a valuable idea. The way the two models are joined still involves the concatenation of their output embeddings, obtained with the fastText and Skip-thoughts configurations described above. As a result we build a model that generates vectors of 2048 + 300 dimensions, leading to an oversized representations with respect to the plain fastText approach. Dealing with vectors of this size has an impact on search response time, a relevant factor to consider in evaluating search engines performances.

5.5 Retrieval

Retrieval unit, in charge of automatically select the documents candidate to be relevant for the query in question, is another decisive search engine module. Considering worst case scenario, having documents in the collection which are strongly relevant to the query without retrieving them, we can understand the criticality of performing retrieval tasks properly. Aiming at performing both cognitive and keyword-based search, we need to select documents that either contain query terms or whose content is kindred to the query semantics. Under these circumstances, our approach consists of two diverse techniques, one to capture the text topics and the other to obtain text keywords. The former method involves the LDA algorithm explained in 3.4.1, the latter consists in comparing the query entities with those found in the corpus. However, the results of these two approaches are then joined to obtain a final list of retrieved files. Aside from directly selecting the documents which are potentially relevant to the user query, there are also query expansion methods. These approaches aim at expanding the set of retrieved documents by enhancing the search input string in order to drive the search by either correcting or improving the user request. In this scenario, we evaluated techniques for either finding and using synonyms of the words in the query or using the keywords in a first generation of results, namely results from a search iteration, to increase the number of selected documents, for a further iteration, that are likely to match the target of the user search. It is worth clarifying that, as long as query expansion can either improve or worsen search results, the defined approaches are performed optionally.

5.5.1 LDA topic filtering

Latent Dirichlet Allocation, as we explained in 3.4.1, consists in an algorithm for identifying 'latent' topics in a document corpus. The underlying model, as it maps specific words to topics, can be used to estimate the membership of text portions to the discovered topics. These memberships are not in the form of binary expressions, they are computed as percentages of belonging to a given topic, so, as they sum up to one, each text can be mapped to diverse topics. Coming to the filtering approach, firstly we compute a topic model and for each topic we store the corresponding documents, then, while the search was being performed, we retrieve these documents whose topic is among the ones that the query is member of. By adopting this method, and in particular by considering all the topics rather than the mere most prominent one, we are targeting a larger number of documents, aiming at reducing the probability to disregard any file relevant to the query. This over-estimating approach comes at the price of slowing down the search process, as more retrieved documents imply more computations, a cost which is worth paying as it allows to reduce the risk of leave texts relevant to the query out. Concerning the LDA topic model parameters, we evaluated the coherence score for models with different number of topics (ranging from 5 to 300) and we found out that, as shown in Figure 21, our corpus is well covered by 30 topics, whose document distribution is displayed by Figure 22.

5.5.2 Entity-based filtering

Entity-based filtering consists in a method for selecting documents based on the presence of specific keywords in their content. The adopted technique for entity-based filtering, described in 3.4.2, features Google Natural Language libraries to extract entities from texts. In practice, similarly to the LDA filtering approach, at first we use external Google APIs to extract entities from each document in the collection, then, we create an inverted index using this data to map to each keyword the documents where it can be found. At query time, we call the same Google Natural Language function previously applied to the corpus files to extract the entities in the user input string. Analogously to LDA topic structure, where each topic is mapped to a list of document members, the entity inverted index allows to perform a very fast look-up to retrieve the files that contain the same entities present in the query. Actually, considering the list of entity types described in 3.4.2, we were interested in extracting only keywords belonging to 'locations', 'names', 'dates', 'organizations', 'events', 'consumer goods' and 'others' categories.

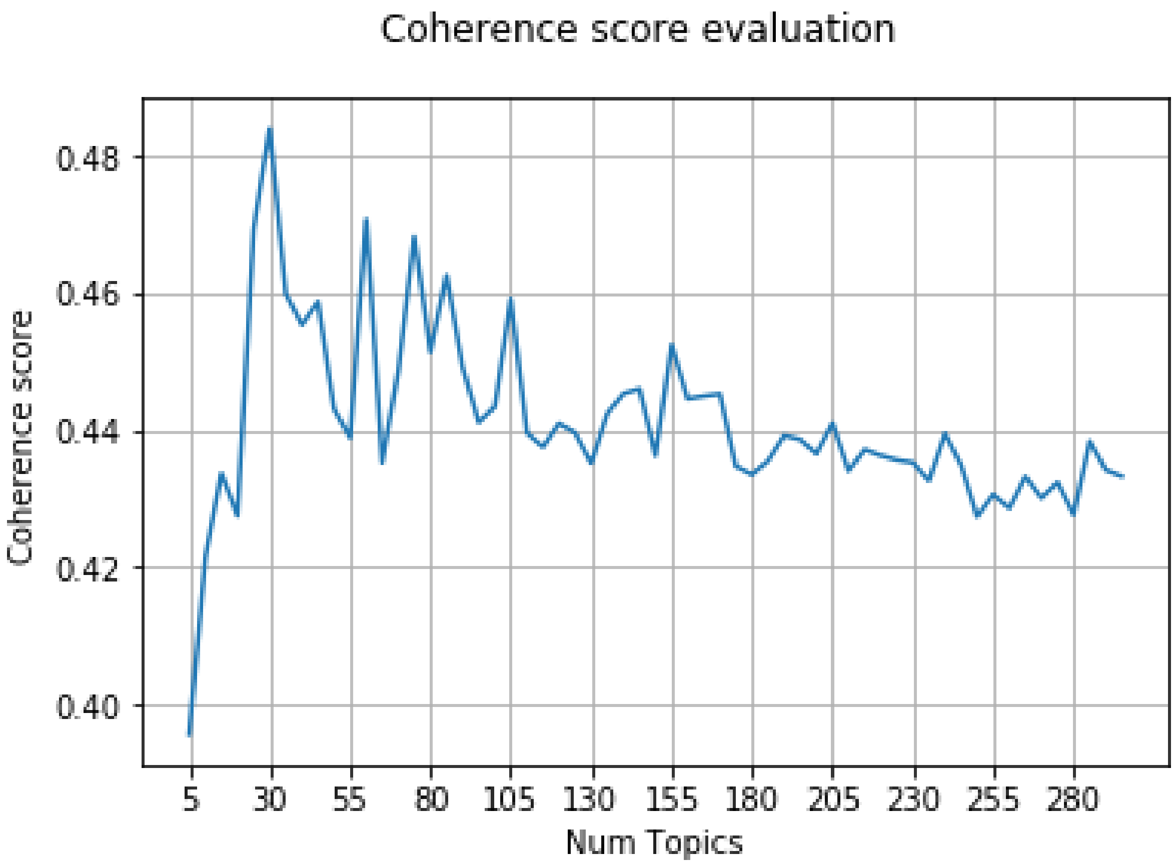


Figure 21: Coherence score plot from models with number of topic ranging from 5 to 300.

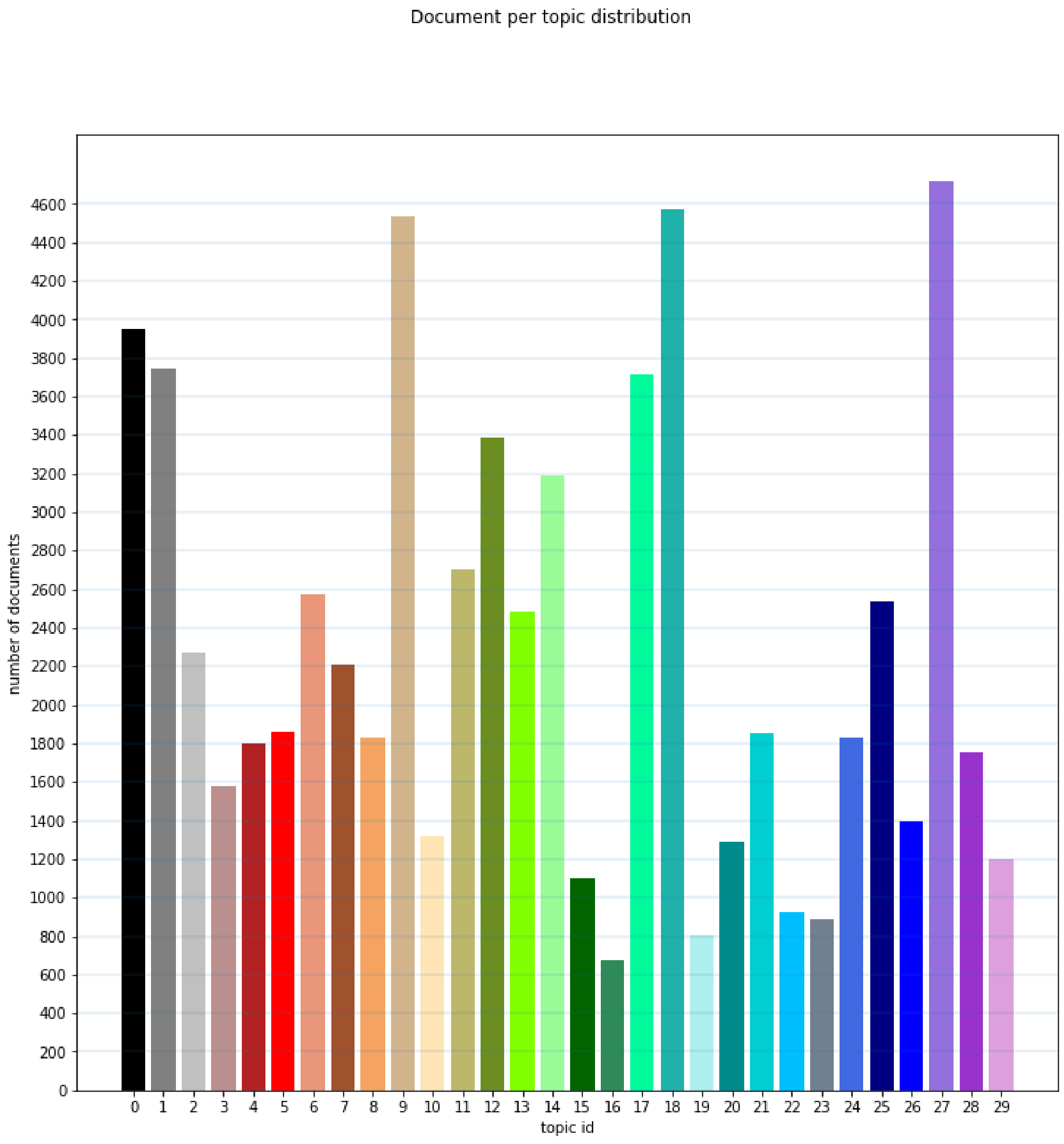


Figure 22: Document distribution across the 30 identified topics.

5.6 Ranking

The ranker is the component which deals with the task of sorting results so to rank them with respect to their relevance to the query. Ranking documents, as we pointed out in Chapter I, is strongly related to the embedding technique, as this sorting operation relies on computing and scoring similarities between encoding vectors. In the context of our problem, the learning representation techniques considered imply a ranking approach based on the cosine similarity, defined in 2.3.1.1. The reason is that this measure of closeness allows to exploit the wide range of information encoded in each of the multiple dimensions of the learnt vectors. Indeed, cosine similarity consists in comparing the angles formed by word representations in a space of as many dimensions as corresponding vectors length. Despite the ease of concepts applied during ranking, this component is critically bounded to excellent performances in terms of computational time, since it represents the pipeline bottleneck for search response time with respect to other units. Embeddings size, number of retrieved documents and code optimization are the main factors that affect ranking performances. In order to leverage the impact of such factors, we performed the multiplications required by cosine similarity as matrix products and we also made use of multi-threading to further improve the response time.

5.7 Visualization

Visualization concerns the task of interacting with the user. Despite this component is not burden by heavy computations, its criticality consists in ensuring that the backend functions and the user ”communicate” properly. Indeed, it is responsible for getting the input query and forward it to the core units, as well as for presenting search results to the user, via SERP. As

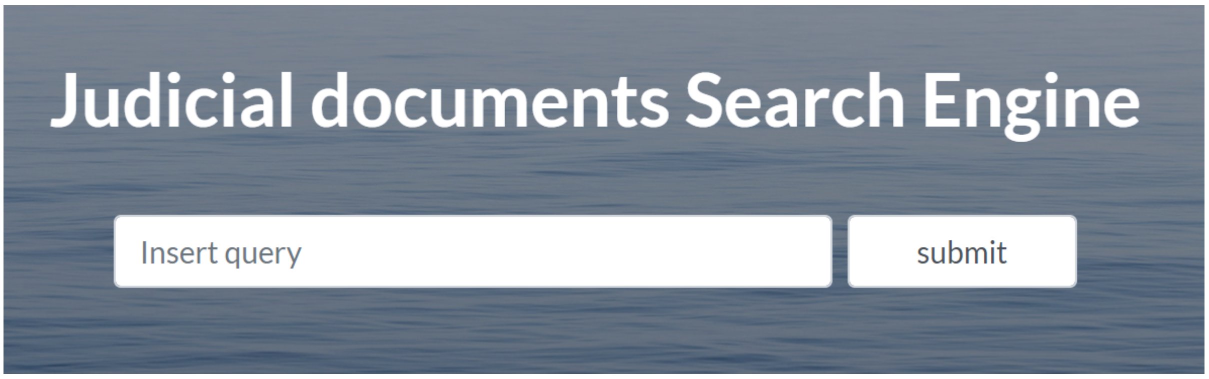
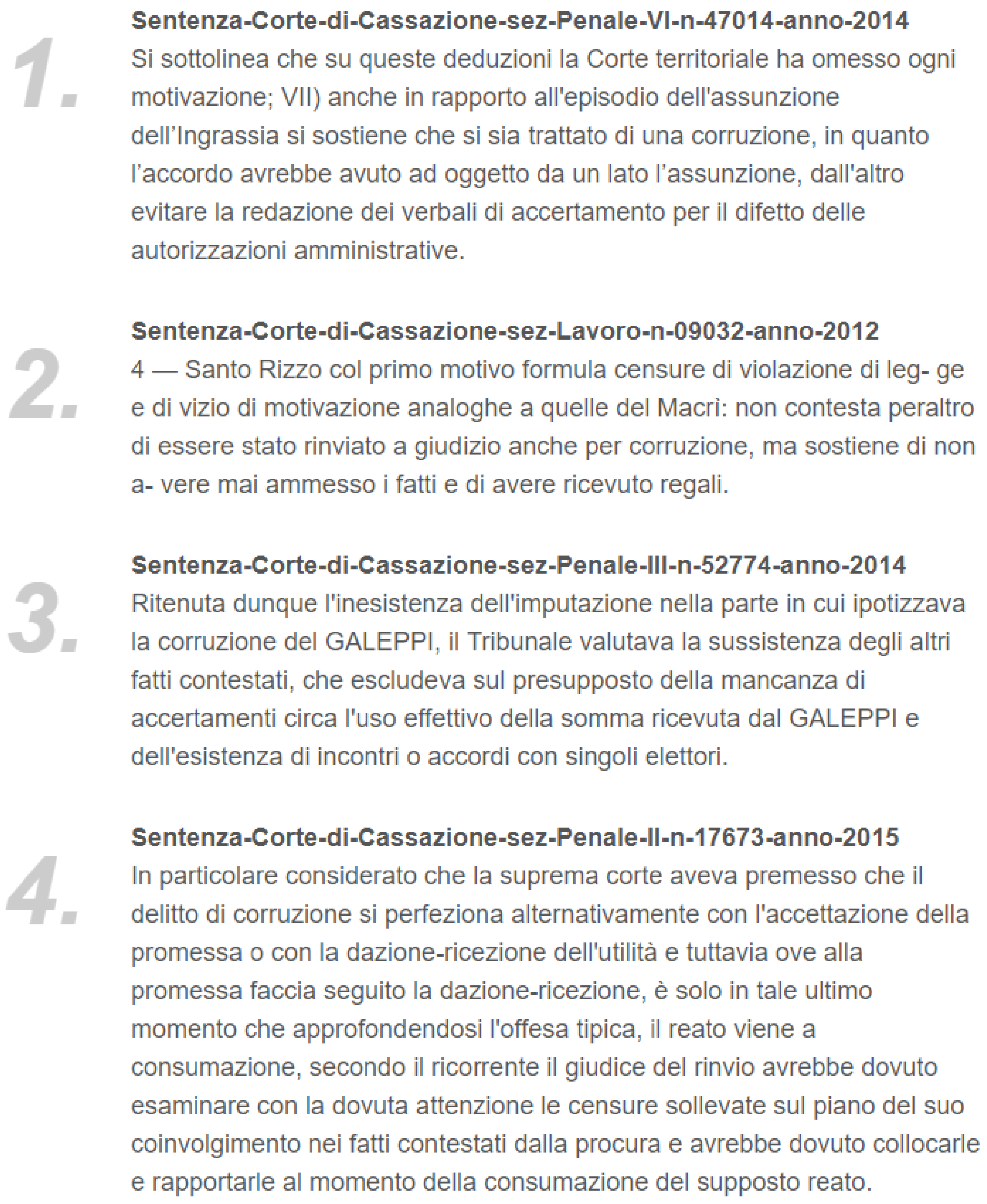


Figure 23: Mockup of the search page.

of our visualization approach, we reasonably included in SERPs both the matching sentence and a link to the corresponding document for each result. This choice is further justified by the purpose of handling both cognitive and keyword-based searches. Indeed, in answering general queries, such as 'Judgements of 2014', providing the most relevant sentences is of as much substantial as showing a link to the most relevant judgements. Oppositely, in case of literal queries, aimed at finding specific keywords, returning the most relevan period within the document, in order to directly take the user to its search target, constitutes an effective answering technique. In addition, since retrieved document can be potentially thousands, we create SERP pages that show the first ten sentences per relevance with the query (top-10 of results rank), allowing the user to iteratively add ten more results. Mockups of the search page and the SERP for a web application are shown in Figure 23 and Figure 24.



1. **Sentenza-Corte-di-Cassazione-sez-Penale-VI-n-47014-anno-2014**
Si sottolinea che su queste deduzioni la Corte territoriale ha omesso ogni motivazione; VII) anche in rapporto all'episodio dell'assunzione dell'Ingrassia si sostiene che si sia trattato di una corruzione, in quanto l'accordo avrebbe avuto ad oggetto da un lato l'assunzione, dall'altro evitare la redazione dei verbali di accertamento per il difetto delle autorizzazioni amministrative.

2. **Sentenza-Corte-di-Cassazione-sez-Lavoro-n-09032-anno-2012**
4 — Santo Rizzo col primo motivo formula censure di violazione di leg- ge e di vizio di motivazione analoghe a quelle del Macri: non contesta peraltro di essere stato rinviato a giudizio anche per corruzione, ma sostiene di non a- vere mai ammesso i fatti e di avere ricevuto regali.

3. **Sentenza-Corte-di-Cassazione-sez-Penale-III-n-52774-anno-2014**
Ritenuta dunque l'inesistenza dell'imputazione nella parte in cui ipotizzava la corruzione del GALEPPI, il Tribunale valutava la sussistenza degli altri fatti contestati, che escludeva sul presupposto della mancanza di accertamenti circa l'uso effettivo della somma ricevuta dal GALEPPI e dell'esistenza di incontri o accordi con singoli elettori.

4. **Sentenza-Corte-di-Cassazione-sez-Penale-II-n-17673-anno-2015**
In particolare considerato che la suprema corte aveva premesso che il delitto di corruzione si perfeziona alternativamente con l'accettazione della promessa o con la dazione-ricezione dell'utilità e tuttavia ove alla promessa faccia seguito la dazione-ricezione, è solo in tale ultimo momento che approfondendosi l'offesa tipica, il reato viene a consumazione, secondo il ricorrente il giudice del rinvio avrebbe dovuto esaminare con la dovuta attenzione le censure sollevate sul piano del suo coinvolgimento nei fatti contestati dalla procura e avrebbe dovuto collocarle e rapportarle al momento della consumazione del supposto reato.

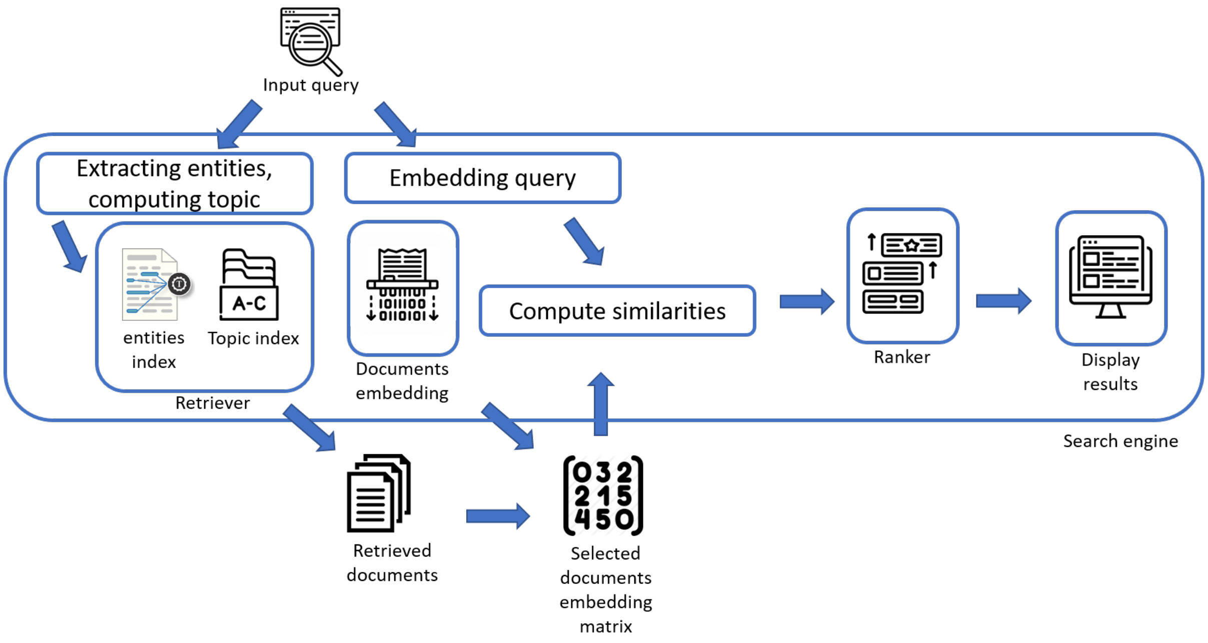
Figure 24: Mockup of SERP.

5.8 Pipeline Description

In this chapter, we had gone through our approach for each of search engines components, describing the performed techniques and the decisions made towards the end product. However, as long as the chapter is fairly verbose,it is worth recalling how the search process is accomplished in short. The procedure starts when an user enters a query string in the apposite field and presses the submit button. Immediately, the query is processed and encoded according to the embedding technique in use. Done this, retrieval is performed, query entities are analyzed, query topic is computed and the corresponding documents are selected. At this point, the system estimate the cosine similarity between the query and each sentence in each selected document, in order to filter out the highest scored phrase per document. Thus, the ranker sort these selected periods by cosine similarity score in descending order. Finally, during visualization step the results are provided to the user. The described process represents the usual sequence of operations performed by our search engine for searching, provided the following pre-computed files:

- Acronyms dictionary
- Processed documents
- LDA model
- Embedding model(s)
- Document sentences embedding vectors
- Entities file per document

- topic-documents inverted index
- entity-documents inverted index



5.9 <u>Search Engine maintenance</u>

Considering the system maintenace, we refer to these tasks which involve search engine improvements and updates. In particular, apart from possibly changing the embedding technique, the ranking method, the user interface and in general replacing whole units, we are interested in describing how to handle the addition of documents to the corpus. Indeed, as Table I demonstrates, this is an explicit requirement. We distinguish two approaches for adding documents, based on the amount of files that user needs to include in the search process. We will talk about 'soft' addition if the corpus increases by a number of document negligible compared to the collection size, otherwise we will use the term 'hard' addition. 'Soft' addition, as it has a limited impact on the corpus and the system itself, would require minor changes. Basically, these are:

- Processing new documents, performing both text extraction and text elaboration
- Extracting entities from new documents to update the corresponding inverted index
- Computing the topic of new documents to update the corresponding inverted index

Oppositely, concerning the 'hard' addition, as long as it deals with a critical variation, it has a more expansive application, both in terms of time and computation, involving:

- Processing new documents, performing both text extraction and text elaboration
- Computing new embedding model and encoding the sentences of corpus documents according to the new paradigm
- Extracting entities from new documents to update the corresponding inverted index
- Computing a new topic model and mapping each new topic to the set of corresponding documents

Among these steps, indeed, surely computing new embedding and topic models is extremely costly and time-consuming operation. Also, it is worth noting that 'soft' changes can be applied without causing a service disruption, instead 'hard' addition would require to shut down the system to make the described changes.

5.10 Summary

In this chapter, we presented the solution we realized in order to achieve the desired goals. We walked through all the search engine components and we mentioned the functioning process for each of them, as well as its role within the overall architecture. In conclusion, we described the pipeline sequence of operations performed in answering to user queries.

CHAPTER 6

SEARCH ENGINE EVALUATION

Up to now, we highlighted the contribution offered by this dissertation. We introduced the state-of-the-art in Information Retrieval, presenting also search engines as the reference software for Text Retrieval. Afterwards, we described the approaches considered for the problem addressed by this thesis and we explored the data used and the requirements to comply with. Ultimately, we analyzed in depth our proposal of solution, detailing all the steps required to obtain the level of performances guaranteed by our end-product.
As of the evaluation, recalling the thesis goal of developing a search engine which features both a cognitive and a keyword-based approaches, we are setting up a comparison among the proposed embedding techniques, main vectors of the syntactic and semantic capabilities of the search engine.

In this chapter, we define the methods applied for evaluating the most promising embedding approach, we declare the expected results and we present the experimental observations. In the end, we provide a key for interpreting them.

6.1 Evaluation approach

The task we addressed is an unsupervised problem, and the estimation of the performances is difficult. Ideally, we should have had a set of predefined queries and corresponding optimal result rankings so that we could have evaluated how close is our solution with respect to the ground truth given. Alternatively, if we were provided with known relevant and non relevant documents for some queries, we could have estimated automatically precision and recall and we would have had strong metrics for assessing the performances of our solution. As a matter of facts, moving on from the above supervised approaches that are impractical for us, we had to define a reasonable assessment process. As we said in the introduction of this chapter, we decide to set up a test based on human feedback to compare the six embedding techniques we considered:

- fastText
- fastText & TF-IDF
- fastText & ELMo
- fastText & TF-IDF & ELMo
- skip-thoughts
- fastText & skip-thoughts

In doing so, we define our evaluation paradigm as follows:

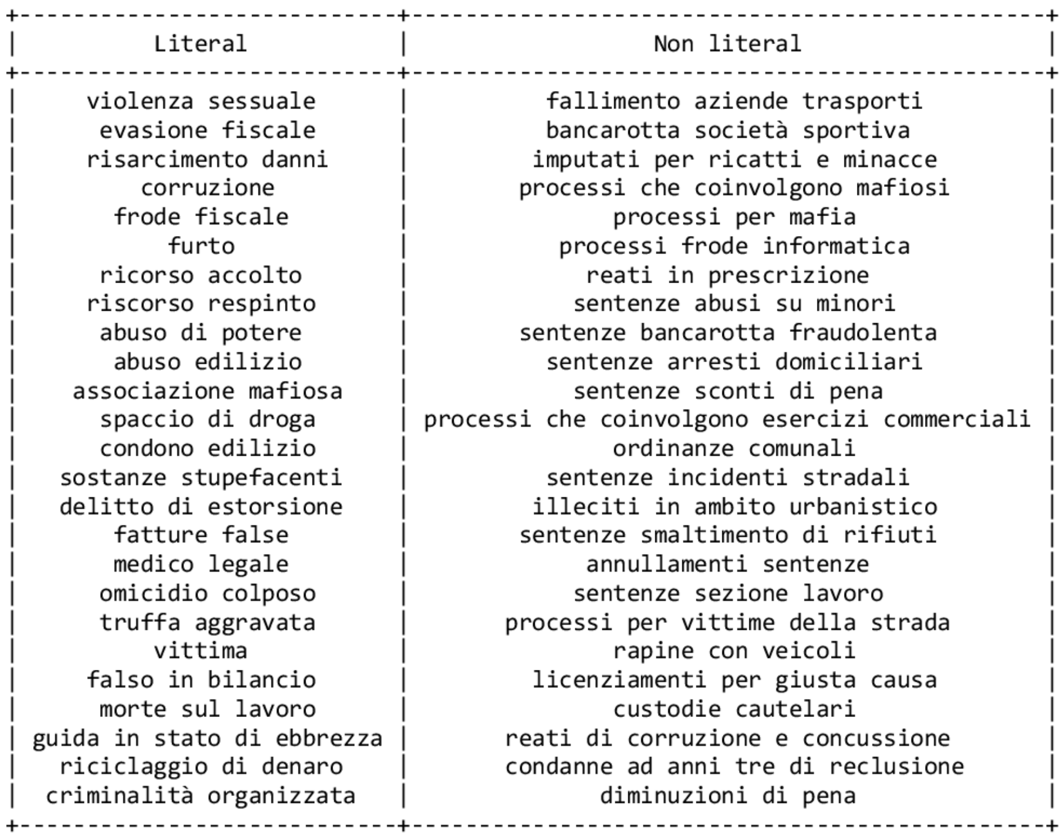
1. We identify 50 different queries, shown in Figure 25, half of which literal (keyword-based), half non-literal (contextual).
2. For each of the six embedding algorithms we run a search against the 50 queries and we keep the first search result (top-1) obtained with the different encodings.

3. We define these two kinds of tests:

    - **best1**: Given the six top-1 results and a corresponding query, the user is required to select the sentence that he/she considers most relevant to the query.
    - **best3**: Given the six top-1 results and a corresponding query the user is required to select the three sentences that he/she considers most relevant to the query.

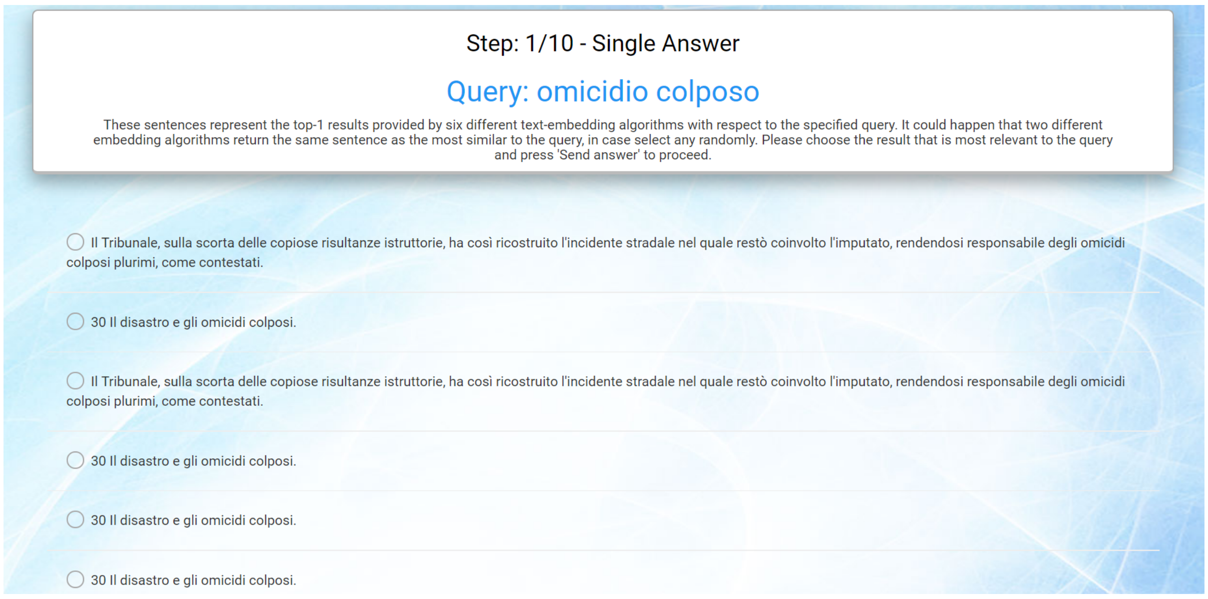
4. We create actual test cases by randomly picking a query and the corresponding top-1 results, generating either a best1 or a best3 test and submitting it to the user.
5. We store the user feedback in a database, using the MongoDB service, to use collected data for further analysis.

In figure Figure 25, there is an overview on the select queries, divided into **literal** and **non literal**. As long as the aim of this work is to develop a search engines which features both a keyword-based and a cognitive approach, we evaluate the performances with respect to both literal and non literal queries. In practice, using the web Python framework named Flask, we implement a web application to collect feedback according to the defined evaluation technique. In Figure 26 and Figure 27 are shown examples of, respectively, best1 and best3 test case web pages. Obviously, in order to make the test reliable, sentence positions are randomized among the six embedding techniques and, clearly, it is not specified which algorithm the result periods belong to. As shown in Figure 26, it can happen to have equal top-1 search results out of different algorithms. In handling such situations, if there are repeated identical sentences



```
+---------------------------+-------------------------------------------------+
|          Literal          |                   Non literal                   |
+---------------------------+-------------------------------------------------+
|     violenza sessuale     |           fallimento aziende trasporti          |
|      evasione fiscale     |            bancarotta società sportiva          |
|     risarcimento danni    |           imputati per ricatti e minacce        |
|         corruzione        |          processi che coinvolgono mafiosi       |
|       frode fiscale       |                processi per mafia               |
|           furto           |            processi frode informatica           |
|      ricorso accolto      |              reati in prescrizione              |
|     riscorso respinto     |              sentenze abusi su minori           |
|      abuso di potere      |          sentenze bancarotta fraudolenta        |
|      abuso edilizio       |           sentenze arresti domiciliari          |
|    associazione mafiosa   |              sentenze sconti di pena            |
|      spaccio di droga     | processi che coinvolgono esercizi commerciali |
|      condono edilizio     |                ordinanze comunali               |
|   sostanze stupefacenti   |            sentenze incidenti stradali          |
|   delitto di estorsione   |           illeciti in ambito urbanistico        |
|       fatture false       |          sentenze smaltimento di rifiuti        |
|       medico legale       |              annullamenti sentenze              |
|      omicidio colposo     |              sentenze sezione lavoro            |
|      truffa aggravata     |          processi per vittime della strada      |
|          vittima          |                rapine con veicoli               |
|     falso in bilancio     |           licenziamenti per giusta causa        |
|      morte sul lavoro     |                custodie cautelari               |
| guida in stato di ebbrezza |          reati di corruzione e concussione      |
|   riciclaggio di denaro   |          condanne ad anni tre di reclusione     |
|  criminalità organizzata  |               diminuzioni di pena               |
+---------------------------+-------------------------------------------------+
```

Figure 25: The 50 queries involved in test cases.


Step: 1/10 - Single Answer

Query: omicidio colposo

These sentences represent the top-1 results provided by six different text-embedding algorithms with respect to the specified query. It could happen that two different embedding algorithms return the same sentence as the most similar to the query, in case select any randomly. Please choose the result that is most relevant to the query and press 'Send answer' to proceed.

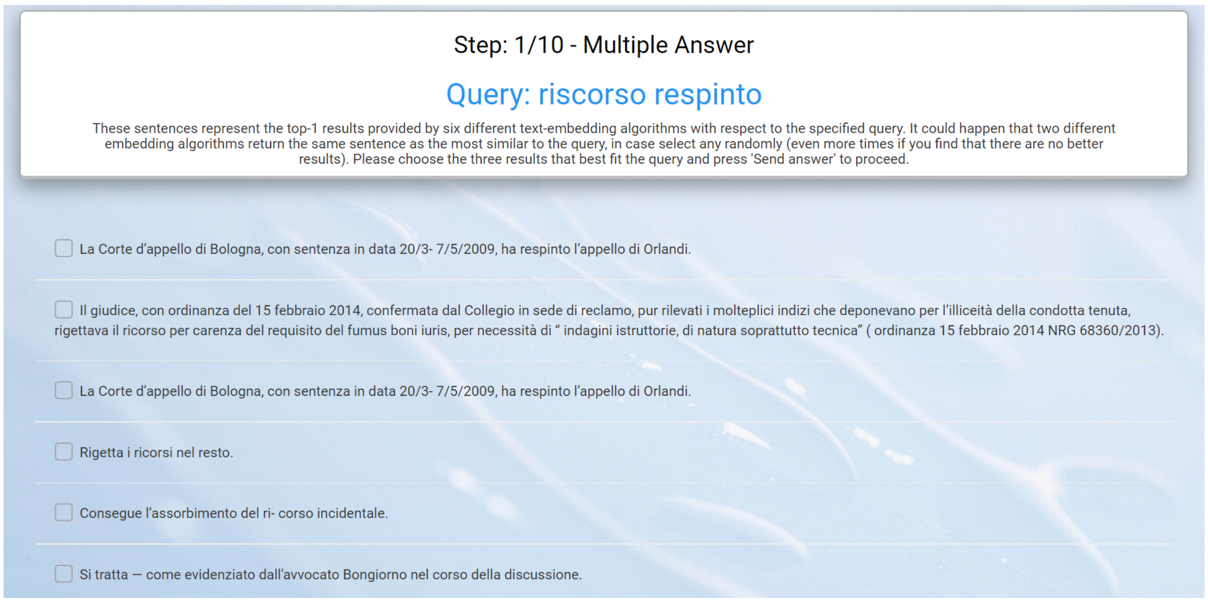
- Il Tribunale, sulla scorta delle copiose risultanze istruttorie, ha così ricostruito l'incidente stradale nel quale restò coinvolto l'imputato, rendendosi responsabile degli omicidi colposi plurimi, come contestati.
- 30 Il disastro e gli omicidi colposi.
- Il Tribunale, sulla scorta delle copiose risultanze istruttorie, ha così ricostruito l'incidente stradale nel quale restò coinvolto l'imputato, rendendosi responsabile degli omicidi colposi plurimi, come contestati.
- 30 Il disastro e gli omicidi colposi.
- 30 Il disastro e gli omicidi colposi.
- 30 Il disastro e gli omicidi colposi.

Figure 26: Best1 test case web page.

within the same test case and one of them got the user preference, the positive feedback is given to all the algorithms represented by that sentence.

6.2 Expected results

Before discussing the actual results obtained, it is worth analyzing briefly the outcome we expect to observe. The rationale behind the evaluation carried out consists in expecting to empirically observe **fastText** and **fastText & TF-IDF** being outperformed by other approaches, as these two are considered strong baseline techniques. More in detail, as long as **fastText** and **TF-IDF** are mostly suitable for encoding syntactical and morphological text properties, we expect them to show poor performances in dealing with non literal queries, contrary to other


Step: 1/10 - Multiple Answer

Query: riscorso respinto

These sentences represent the top-1 results provided by six different text-embedding algorithms with respect to the specified query. It could happen that two different embedding algorithms return the same sentence as the most similar to the query, in case select any randomly (even more times if you find that there are no better results). Please choose the three results that best fit the query and press 'Send answer' to proceed.

- [ ] La Corte d'appello di Bologna, con sentenza in data 20/3- 7/5/2009, ha respinto l'appello di Orlandi.
- [ ] Il giudice, con ordinanza del 15 febbraio 2014, confermata dal Collegio in sede di reclamo, pur rilevati i molteplici indizi che deponevano per l'illiceità della condotta tenuta, rigettava il ricorso per carenza del requisito del fumus boni iuris, per necessità di " indagini istruttorie, di natura soprattutto tecnica" (ordinanza 15 febbraio 2014 NRG 68360/2013).
- [ ] La Corte d'appello di Bologna, con sentenza in data 20/3- 7/5/2009, ha respinto l'appello di Orlandi.
- [ ] Rigetta i ricorsi nel resto.
- [ ] Consegue l'assorbimento del ri- corso incidentale.
- [ ] Si tratta — come evidenziato dall'avvocato Bongiorno nel corso della discussione.

Figure 27: Best3 test case web page.

approaches which feature context-based models like **ELMo** and **Skip-thoughts**. Thus, our expectation is mostly endorsed by the fact that, aiming at developing a search engine for both keyword-based and semantic-based queries, algorithms composed of both pure keyword-based models and cognitive approaches should lead to better overall performances. Considering each embedding algorithm separately, we can suppose that both **fastText** and **fastText & TF-IDF** show excellent results for literal queries and behave oppositely for non literal ones. A totally opposite attitude is predicted for the pure cognitive Skip-thoughts model. Other approaches, namely **fastText & ELMo**, **fastText & ELMo & TF-IDF** and **fastText & Skip-thoughts** are those we expect to show valuable results both for literal and non literal queries.

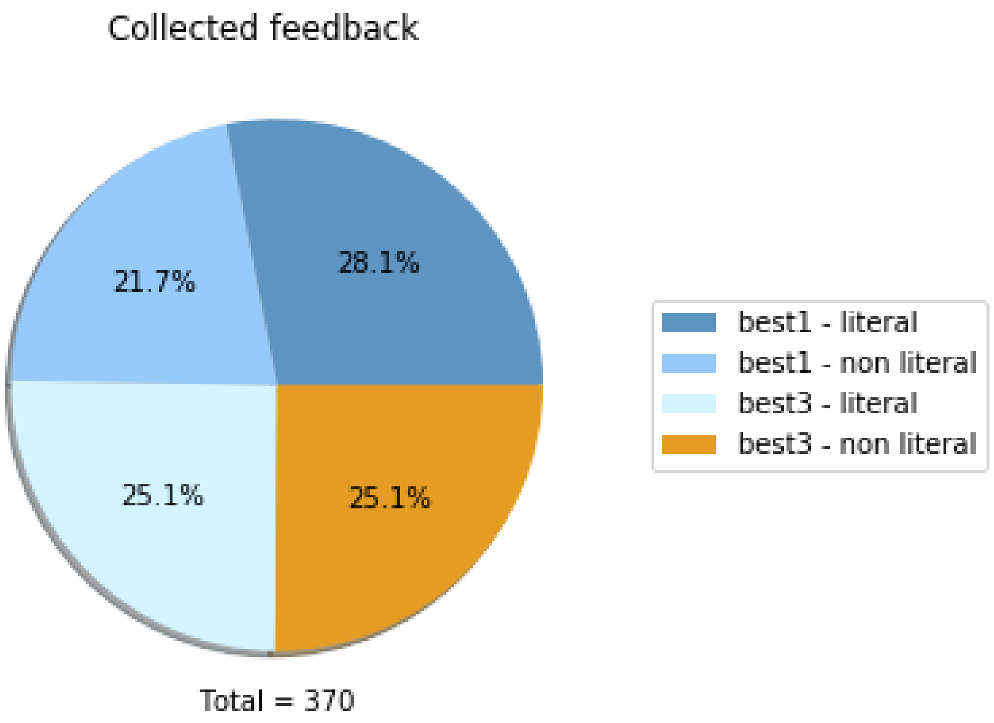


Figure 28: Statistics about collected feedback distribution over test cases.

6.3 Experimental results

In this section we are presenting the results collected through our tests. It is worth noting that the feedback have been collected mostly from male people aged between 20 and 40 with a background in statistics and computer science, unfamiliar with judicial ambit. In total we collected 370 feedback, distributed over the 50 queries shown in figure Figure 25 and the two test cases described in section 6.1. An overview on the distribution is exposed in figure Figure 28.

We now show the results for the four test categories:

- **best1 literal**: best3 test performed on a 'literal' query.
- **best1 non-literal**: best3 test performed on a 'non-literal' query.
- **best3 literal**: best1 test performed on a 'literal' query.
- **best3 non-literal**: best1 test performed on a 'non-literal' query.

We organize the study of the experimental results by considering best1 and best3 tests separately at first. In doing so, we present collected data for literal and non literal queries and then we compare them, briefly commenting each analysis. In conclusion, we inspect the four tests together and we draw the final remarks.

6.3.1 Best3 literal test

Considering the best3 test for literal queries, whose results are displayed in Figure 29, we can observe that **fastText** and **fastText & ELMo** received limited feedback compared to the other four algorithms. Analyzing these four, we find out that **Skip-thoughts**, pure contextual model, and **Skip-thoughts & fastText** received almost the same number of feedback of **fastText & TF-IDF** and **fastText & TF-IDF & ELMo**. This reduced distance among the best models and the least worst ones is explained by the best3 test requirement to express three preferences. Indeed, this kind of test prevents to figure out the absolute best algorithm but allows to clearly state which are the worse ones, namely **fastText** and **fastText & ELMo**, that resulted the least preferred models.

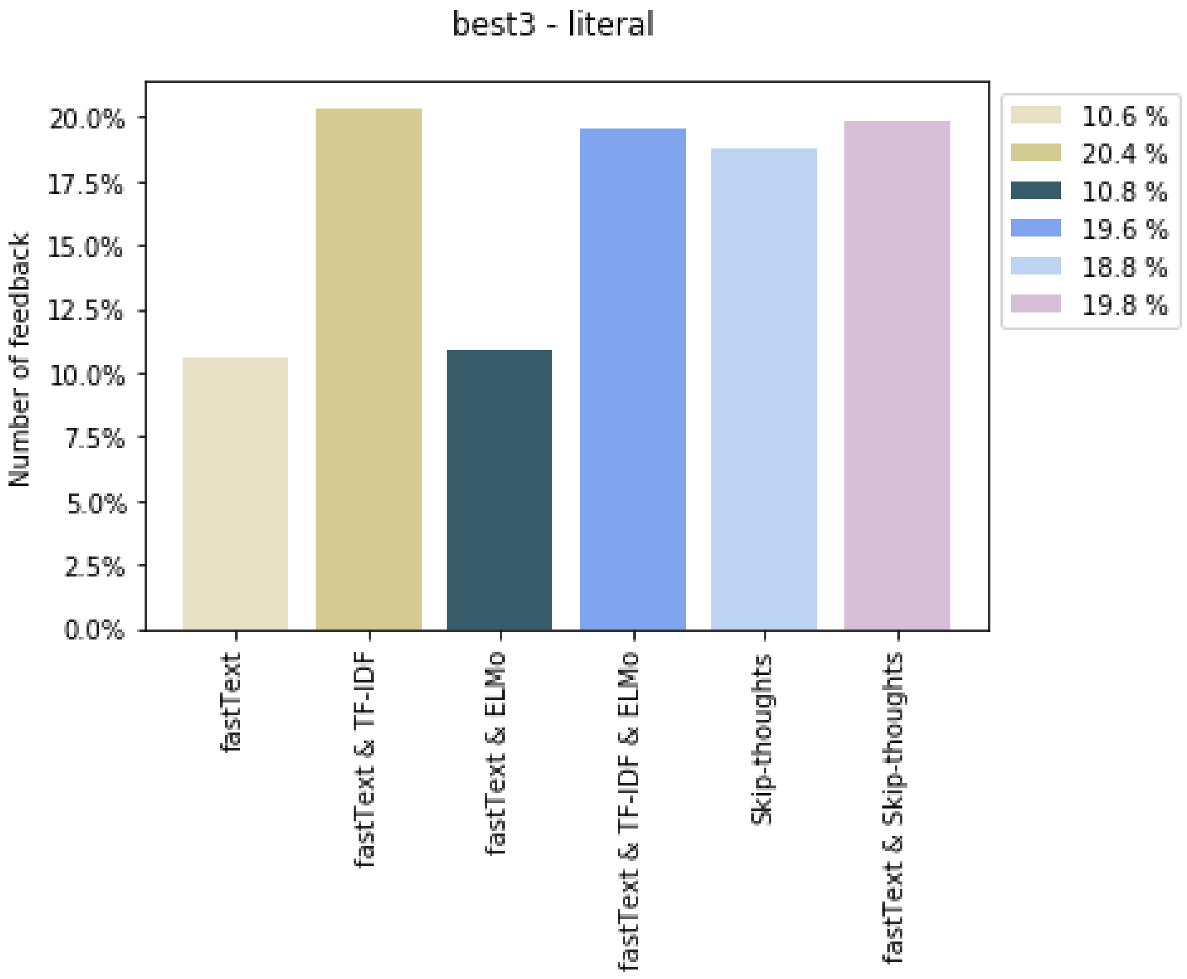


Figure 29: Best3 literal test collected feedback.

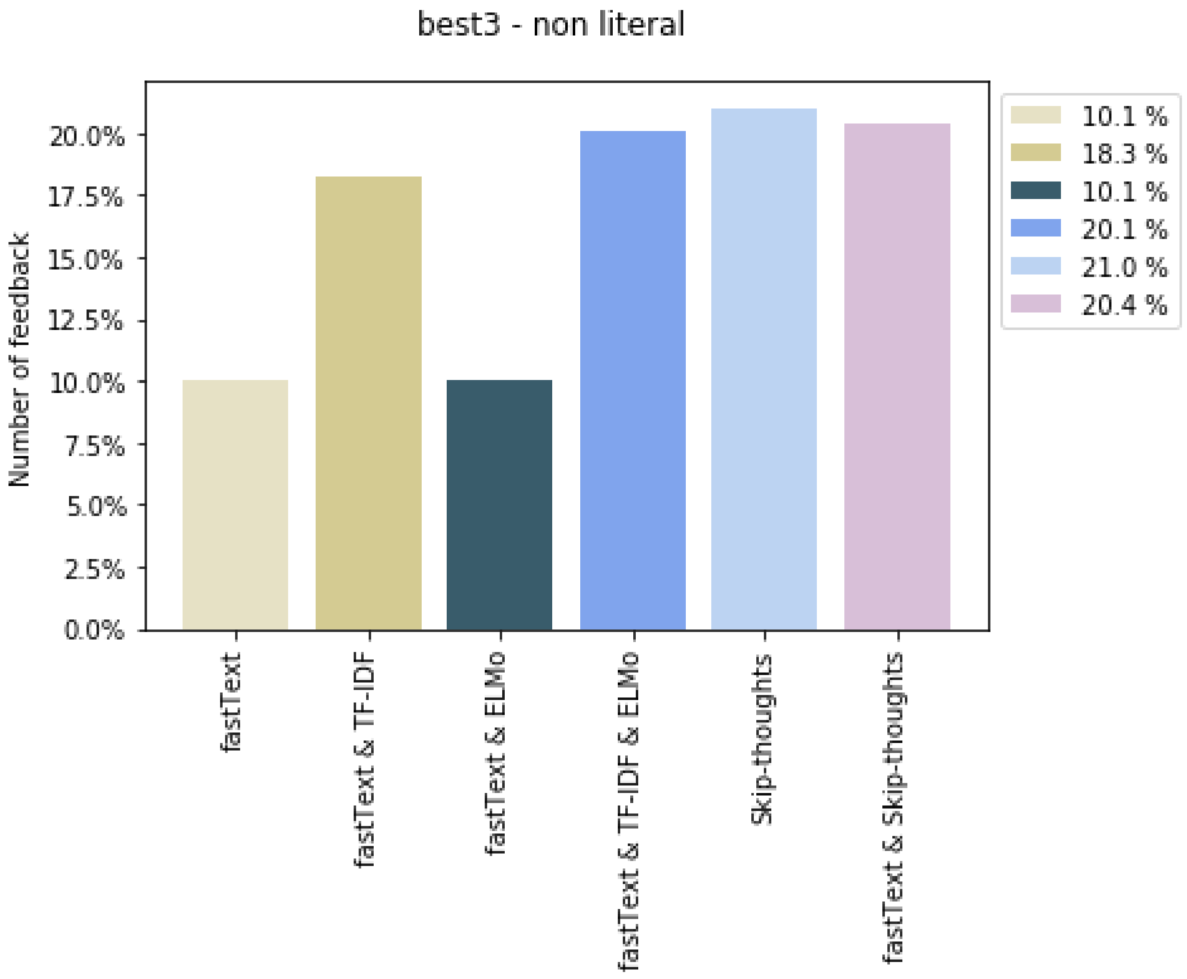


Figure 30: Best3 non-literal test collected feedback.

6.3.2 Best3 non-literal test

As of the results related to best3 test for non-literal queries, shown in Figure 30, we can note that the general trend observed in the best3 test for literal queries is kept. Considering **fastText & TF-IDF** and **fastText & TF-IDF & ELMo**, we can see how the distance among them increased in favor of the latter, with respect to the best3 test for literal queries. In addition, according to this test, they gave way to **Skip-thoughts** and **Skip-thoughts & fastText** as favourite approaches. In particular, the pure **Skip-thoughts** model, received the majority of the preferences. However, the distance among the four top-voted models ranges in less than 3% points, 0.9% points if we consider the best three approaches, a very limited gap. Once more, **fastText** and **fastText & ELMo** received, by far, the fewest preferences.

6.3.3 Best3 literal/non-literal comparison

The bar plot in fig. Figure 31 allows us to observe the results of best3 test, comparing the feedback for literal queries with those for non-literal ones. We can notice that the two rankings for literal and non-literal queries present differences in the first four placements. In particular, the two rankings are:

- **best3 literal**:

  1. fastText & TF-IDF
  2. fastText & TF-IDF & ELMo
  3. fastText & Skip-thoughts
  4. Skip-thoughts

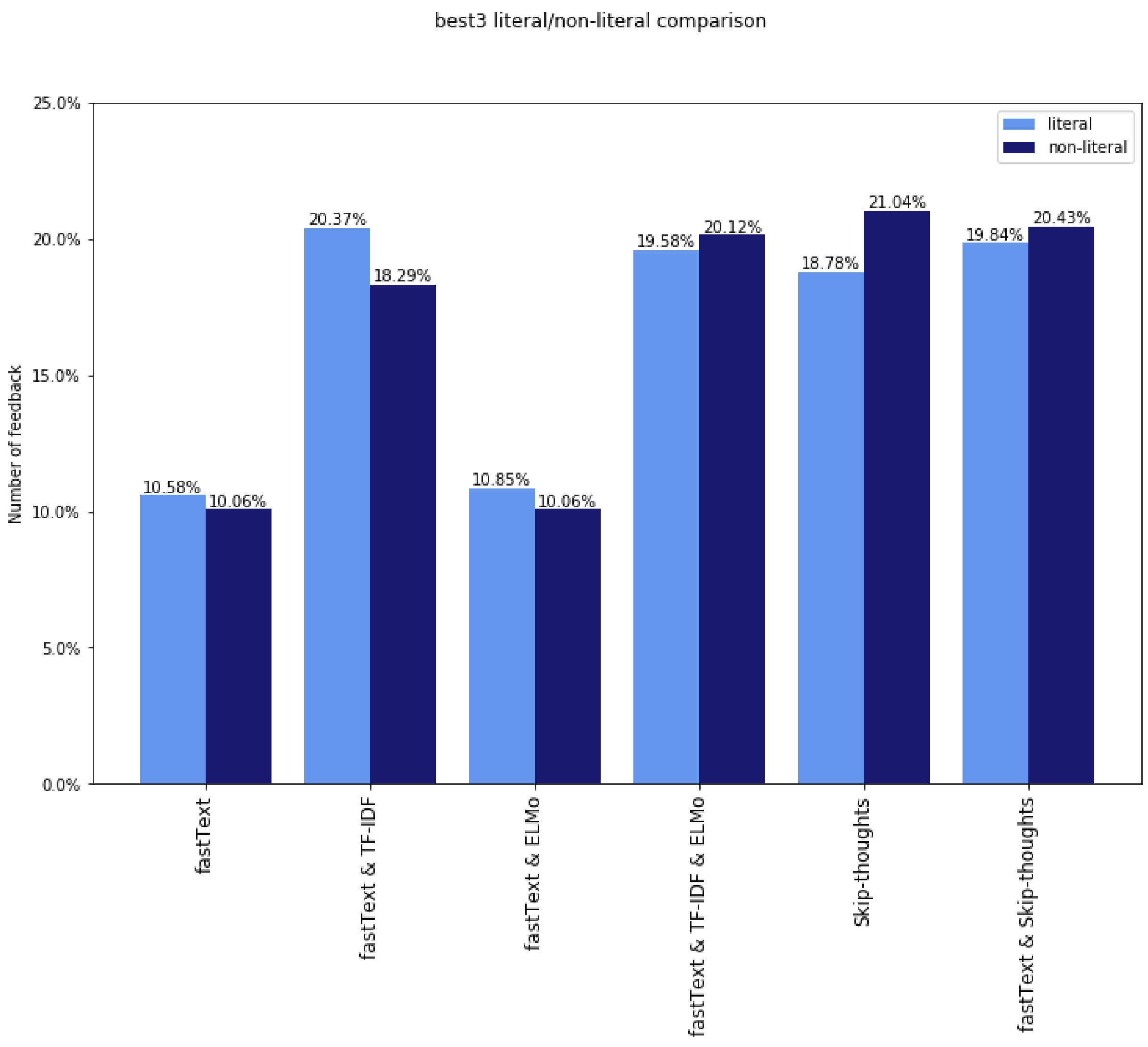


Figure 31: Best3 literal/non-literal tests comparison.

- **best3 non-literal**

  1. Skip-thoughts
  2. fastText & Skip-thoughts
  3. fastText & TF-IDF & ELMo
  4. fastText & TF-IDF

Basically, the two rankings are the inverse of each other. However, as we pointed out in section 6.3.2, there are so few points of distance among these top four algorithms that any conclusion about the reversed ranking causes and the overall best algorithm would be weakly supported by the observations. Thus, we provide now best1 test results.

6.3.4 Best1 literal test

The main conclusion we can infer from the best1 - literal test, presented in figure Figure 32, regards the crucial impact of **TF-IDF**. Indeed, looking at the bar plot, it is evident how both **fastText**, surprisingly, and **fastText & ELMo** received a limited number of feedback compared to **fastText & TF-IDF** and **fastText & TF-IDF & ELMo**, which resulted the preferred approaches. Unexpectedly, we can also notice that Skip-thoughts, typically a context based model, received a significant number of preferences, whether by itself or combined with **fastText**. Considering all the embedding models, it clearly turns out that pure **fastText** impact is negligible with respect to literal queries. In fact, looking at the approaches that feature **Skip-thoughts** encoding, we can see that whether with or without **fastText**, they got almost the same number of preferences.

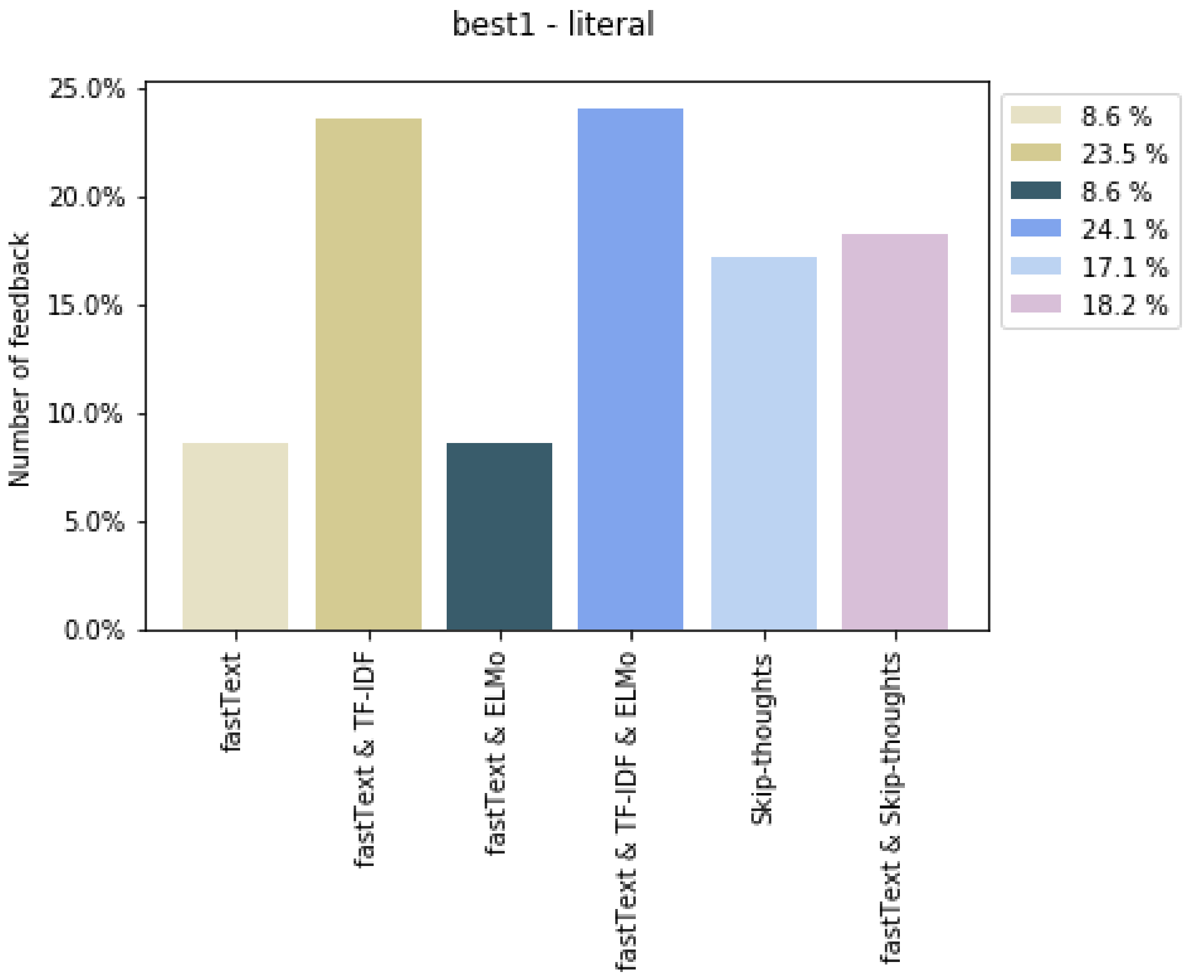


Figure 32: Best1 literal test collected feedback.

6.3.5 Best1 non-literal test

Looking at the best1 - non literal feedback bar plot, fig. Figure 33, immediately we can observe the same trend noted for best1 - literal test. However, considering this test case, the prevalence of **fastText & TF-IDF & ELMo** is more emphasized, probably thanks to the contribution of context-based **ELMo**. Still **fastText & TF-IDF** results the second preferred model, even if, this time, it is closer to the two models which feature **Skip-thoughts**, rather than to the best embedding approach. With a negligible number of preferences, we can find **fastText** and **fastText & ELMo**, that, as expected, showed a poorer performance for non-literal queries than for literal ones.

6.3.6 Best1 literal/non-literal comparison

Our interest in the literal/non literal comparison for best1 test, shown in Figure 34, is directed to discover potential discrepancies in the trends of the one with respect to the other. Actually, analyzing the graph is it clear how both literal and non-literal present the same algorithm ranking per number of feedback. This aspect is quite unexpected, as, in principle, there are embedding techniques that should be most suitable for literal query and viceversa. However, this peculiar evidence on literal and non-literal comparison contributes to make the obtained results more reliable and it also suggests us that we obtained the desired algorithm to capture both syntactic and semantic text proprieties.

6.3.7 Overall test comparison

Considering the graph shown in Figure 35, we can observe a matrix structure with best1 and best3 tests as rows and literal and non-literal as columns. Looking at all the test results at

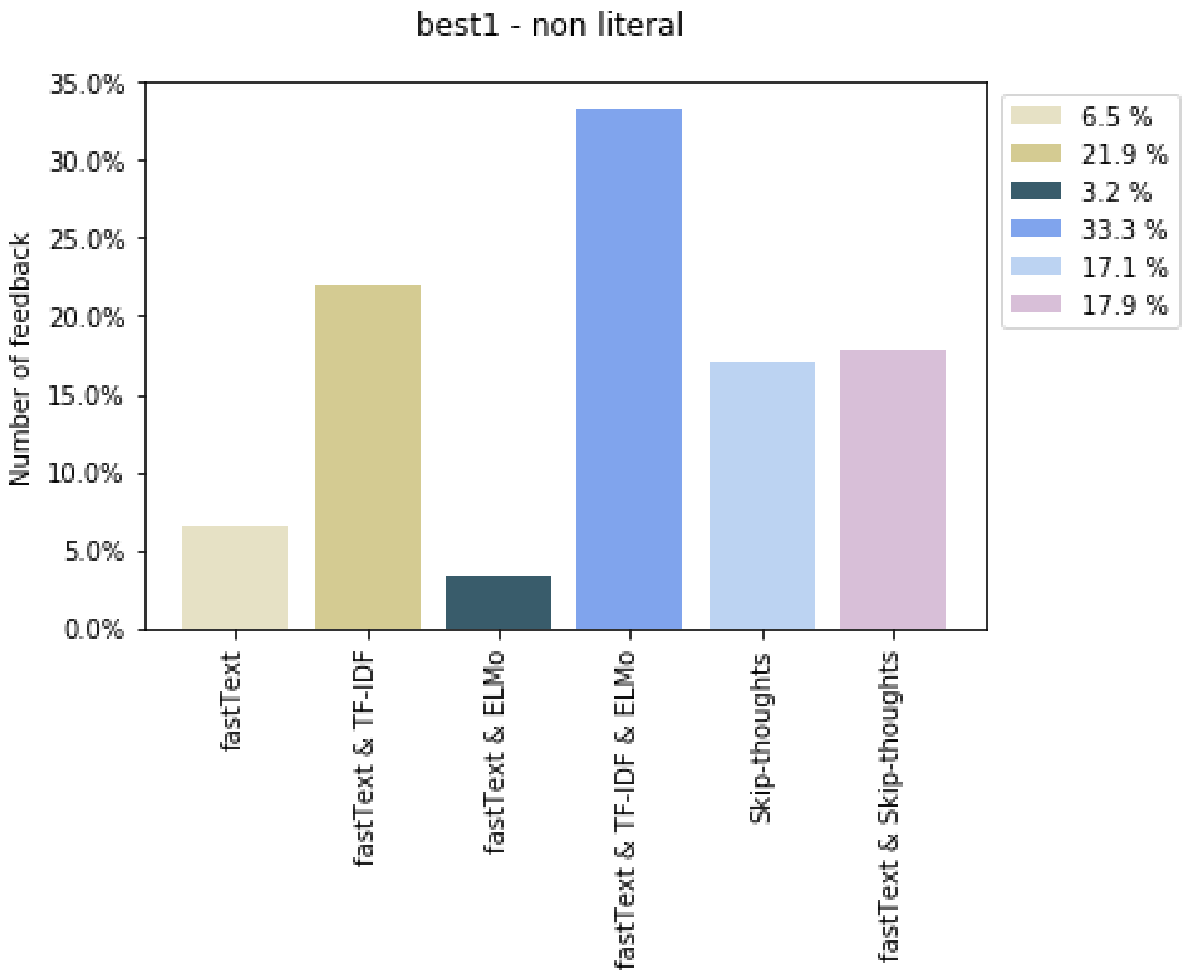


Figure 33: Best1 non-literal test collected feedback.

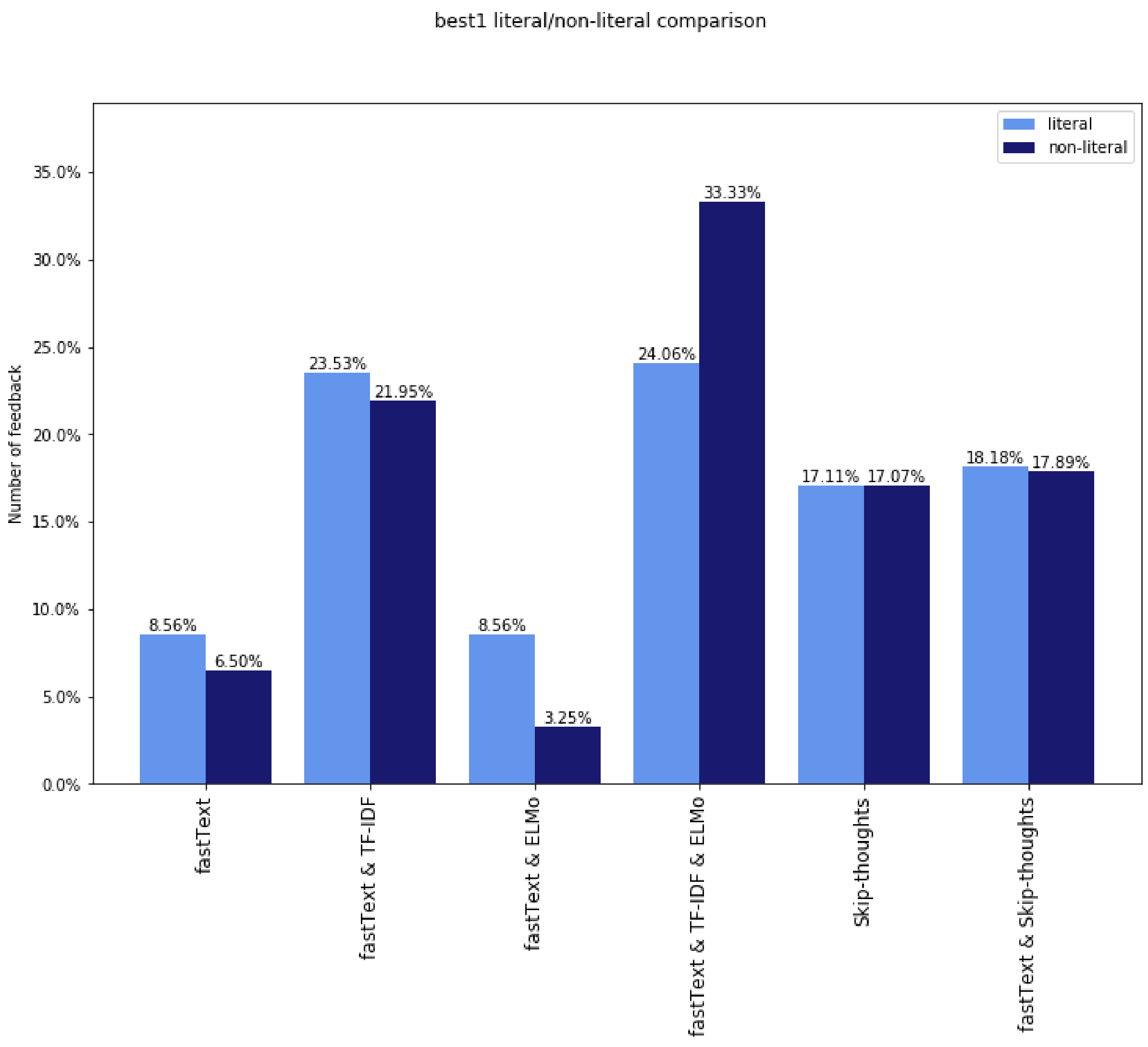


Figure 34: Best1 literal/non-literal tests comparison.

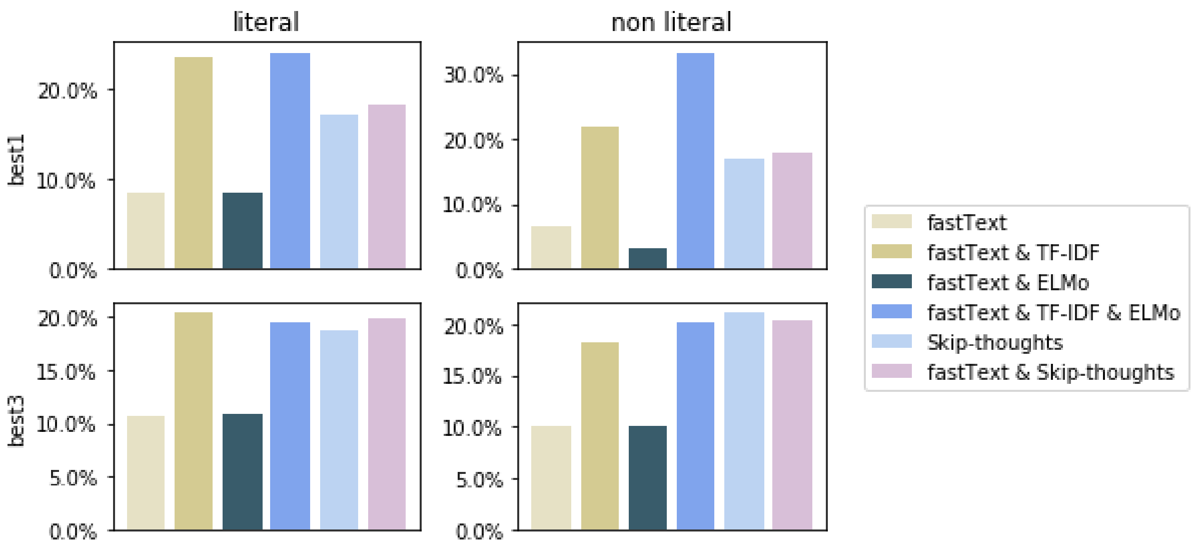


Figure 35: Comparison among the four tests performed.

once, it clearly emerges the predominance of two bars, **fastText & TF-IDF** and **fastText & TF-IDF & ELMo** ones, which in total collected more positive feedback than others. More in depth,is of exception the best3 non-literal test, where **fastText & TF-IDF** and **fastText & TF-IDF & ELMo** result respectively the fourth and the third models for few preferences. Also, in best3 literal **fastText & TF-IDF & ELMo** results third in the number of feedback received. Another evident outcome from the general comparison regards the **fastText** and **fastText & ELMo** models, that appear as the least favourite models in general. More complex is the behaviour of **Skip-thoughts** and **Skip-thoughts & fastText**, whose performances range from mediocre in best1 tests to highly voted in best3 tests. Analyzing Figure 36 and Figure 37,

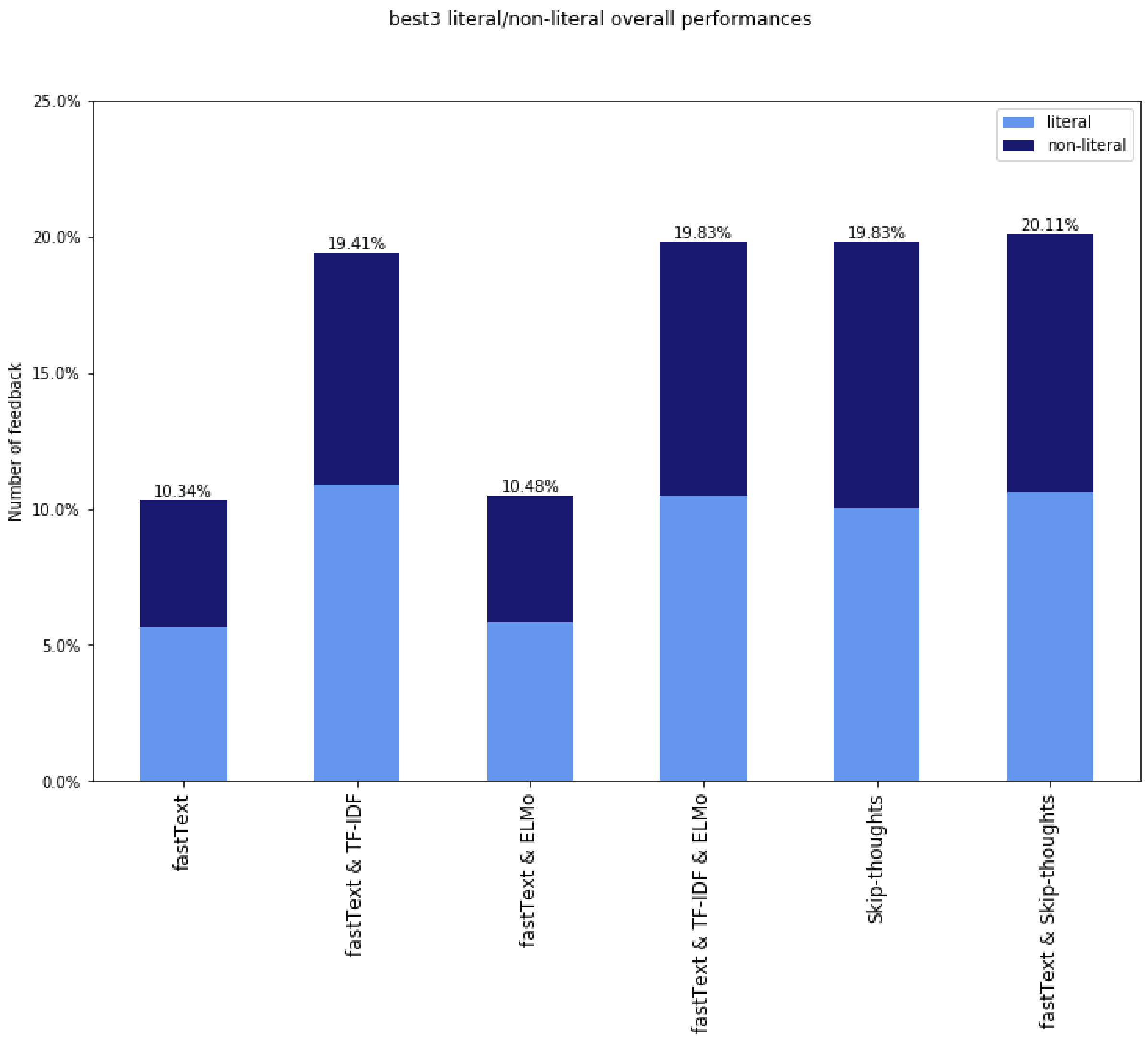


Figure 36: Best3 literal & non-literal overall results.

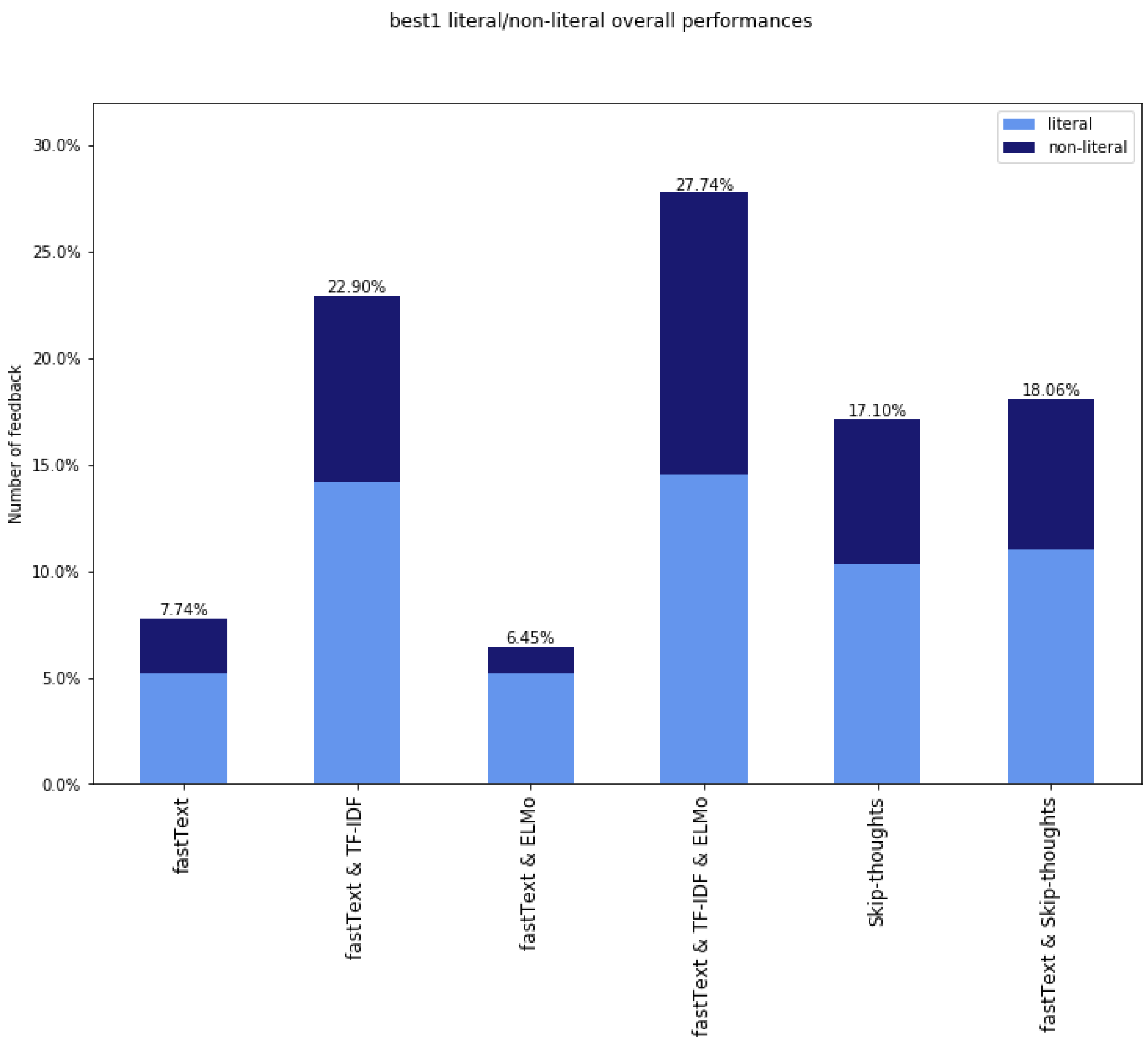


Figure 37: Best1 literal & non-literal overall results.

we can have a look to the overall performances in handling literal and non-literal queries of each embedding technique. Actually, the best3 test, whose results are shown in Figure 36, is not very meaningful in identifying the best model. On the other hand, it significantly shows that the worst models are **fastText** and **fastText & ELMo**. Indeed, regardless to the nature of the query, whether literal or non-literal, these models have rarely been select among the top-3 embeddings. Oppositely, the other four embeddings present almost the same overall results, however we need to consider best1 results to derive conclusions about the single most favoured model. Studying figure Figure 37, we can observe more marked results. On one hand, **fastText** and **fastText & ELMo** confirm the performances pointed out in best3 test. On the other hand, **fastText & TF-IDF & ELMo** emerges as the model that received the highest number of feedback, thanks to the performances with respect to non-literal queries, appearing quite distant from **fastText & TF-IDF** and from the **Skip-thoughts** models. Moreover, **fastText & TF-IDF** clearly turns out to be second favoured model, at the expense of **Skip-thoughts** and **fastText & Skip-thoughts**. Also, **fastText & TF-IDF** results as close to the third and the fourth model, respectively **fastText & Skip-thoughts** and **Skip-thoughts**, as to the second one, with a distance around 5%.

6.4 Evaluation remarks

Analyzing the outcome of our evaluation, we are fairly satisfied. As we pointed out in 6.2, we basically expect to see that at least one of the innovative models proposed outperforms **fastText** and **fastText & TF-IDF**. We tend to consider best3 test less reliable than best1 test, as there is a higher probability that some model has been selected for being least worst than others. Hence, looking at best1, we can see, that **fastText & TF-IDF** and **fastText & TF-IDF & ELMo** show similar performances in handling literal queries, but the latter outperforms the former in dealing with non-literal queries. Thus, the contribute of **ELMo** to the baseline model **fastText & TF-IDF**, increases, as expected, the capability to answer effectively to non-literal queries. On the other hand, we are fairly surprised to note that models which feature **Skip-thoughts** performed better for literal queries than for non-literal ones, considering best1 test context. Taking into consideration best3 tests without neglecting best1 cases, allows us to establish the overall worst models. Indeed, summing up the contribute of best1 and best3 tests, it is clearly evident that **Skip-thoughts** and **fastText & Skip-thoughts** represent respectively the fourth and the third models in term of user preferences. However, suprisingly, **fastText & Skip-thoughts** received less preferences than the pure context-based **Skip-thoughts** looking at the best1 non-literal test. Finally, it is also clear that **fastText** and **fastText & ELMo** are the least favoured approaches. Unexpectedly, **fastText & ELMo** shows overall terrible performances and in particular, remarkably, it performs worse than pure **fastText** for non-literal queries, according to best1 tests. In the other tests the number of positive feedback received by these two embedding approaches is comparable. Trying to interpret the observation raised from the evaluation, we have determined the following.

- **fastText** poor performances are probably related to the fact that it produces word-level encoding vectors, unlike other models which generate sentence embeddings.
- As expected, **ELMo** is of low impact in handling literal queries but it strongly affects results for non-literal ones.

- **Skip-thoughts**, as a sentence-level embedding model, works relatively well regardless to the query type. Also, when combined with **fastText**, it presents improved performances on literal queries.
- The impact of **TF-IDF** is significant both for literal and non-literal queries as can be seen analyzing the number of feedback for **fastText**, **fastText & TF-IDF**, **fastText & ELMo** and **fastText & TF-IDF & ELMo**

6.5 Summary

In this chapter, we described the evaluation technique applied in order to asses the performances of our proposed approach, we declared the expected outcome and we presented the experimental results, concluding with some considerations on the evaluation carried out.

CONCLUSIONS & FURTHER WORKS

In this thesis we presented a domain-specific search engine for OCR documents built on the combination of cognitive and keyword-based approaches, with the aim of providing relevant answers to the user regardless to the type of the query, whether literal or non-literal.

Dividing our task into sub-problems, we can identify two main challenges, OCR text extraction and Text Retrieval. In performing OCR extraction, we basically applied state of the art techniques and personalized procedures to extract the text from images and process it. Considering Text Retrieval field, we selected the most suitable state of the art techniques with respect to the addressed task and we exploited them to come up with diverse custom solutions, so as to compare many approaches and find the most promising one.

Focusing on Text Retrieval as the main purpose of this thesis, we worked on the binomial syntax and semantics mainly on two levels, namely the word embedding and the retrieval search engine components. As of the former, responsible for capturing text proprieties and encoding them into a numbered vector, in general, we pursued the idea of combining keyword-based and context-based embeddings by concatenating them into a unique representation. Concerning the latter, the component that selects from the corpus these documents which are candidate to be relevant for the query in question, we joined an approach to select files that share keywords with the query with a technique to select texts whose topic is the argument of the query. We analyzed the search process through human feedback, willing to identify the best algorithm, in terms of user preferences, among those applied for embedding texts. Considering the results further, particularly those related to best1 test, we can derive diverse conclusions. First of all, it is worth noting that fastText model, originally considered a reliable baseline, resulted in poor performances, thus, we consider fastText & TF-IDF, which collected many positive feedback, a stronger baseline. In this scenario, we notice that pure Skip-thoughts and fastText & Skip-thoughts models received more user preferences than fastText, but less than the fastText & TF-IDF model, the stronger baseline. On the other hand, fastText & TF-IDF & ELMo outperformed, in terms of collected feedback, the stronger baseline, resulting in the best overall embedding model. Oppositely, fastText & ELMo, which performed worse than fastText, proved to be the worst model among those considered. Hence, with respect to our work, we can draw up the following ranking:

1. fastText & TF-IDF & ELMo
2. fastText & TF-IDF
3. fastText & Skip-thoughts
4. Skip-thoughts
5. fastText
6. fastText & ELMo

Looking at this ranking, we can visualize some of the observations drawn in interpreting the test results. Models that feature TF-IDF appear in the first two placements of the ranking, but the same models without TF-IDF turn out to be the worst ones. FastText & TF-IDF, which consists of two mainly keyword-based embeddings results better than combining a syntactic-based approach such as fastText and Skip-thoughts, a semantic-based technique. Pure Skip-thoughts resulted better than pure fastText, showing that a sentence-level encoding is more suitable than a word-level one for our task. Finally, we are rationally satisfied by the obtained results. Indeed, we figured out an embedding model that reaches state of the art algorithms in searching against literal queries and outperforms them in non-literal researches.

On the basis of this dissertation, many interesting paths can be further explored to improve and enrich this work. Walking through the pipeline structure, we are providing suggestions for additional contributions towards the purpose of this thesis. Starting from the choice of the domain and therefore of the data, we think that examining the performances of our search engine on a different domain would ensure us that the provided software is actually effective for searching against verbose documents from collections featuring various topics. Concerning OCR text extraction and document processing, there are so many options that investigating them is pointless, however, having the faculty to work with cloud platforms, could be of interest comparing 'Tesseract' and other commercial OCR engine performances. Word embedding is the component more prone to modifications and improvements. The most immediate idea for testing diverse solutions consists in changing some parameters of the models we have adopted, however further works in word emebedding are not limited to this. In recent years, Neural Network are to the fore and new architectures of wide applicability are discovered daily. As long as word embedding approaches, nowadays, exclusively rely on NNs, we encourage to apply new disclosed techniques for encoding text, possibly exploiting the idea of combining keyword-based and context-based models. In particular we suggest to combine fastText, Skip-thoughts and TF-IDF, indeed, considering Skip-thoughts an alternative, as a contextual model, to ELMo, and provided that fastText & TF-IDF & ELMo showed very good performances, we expect fastText & TF-IDF & Skip-thoughts to result comparable to the best model we implemented. In retrieving and ranking documents, many methodologies can be applied. Regarding these tasks, since may not be possible to exploit a commercial solution for extracting textual entities, several diverse approaches can be tested, such as the Rapid Automatic Keyword Extraction (RAKE) algorithm. Also, in building a topic model, alternative techniques to LDA can be tried out. Ranking itself can be performed according to several similarity measures, taking cues from the many distance equations available. Surely, even User Interface (UI) tasks, such as accepting the user query and presenting search results, can be achieved in many ways, based on several design ideas and presentation goals. In the end, regarding the task of estimating the performances, our search engine is suitable for various evaluation tests. In addition to our method, one could ask for feedback on how many results are relevant for top-10 rankings provided by each algorithm, willing to estimate precision and recall. Surely there are many other more or less creative testing approaches.

CITED LITERATURE

1. Singhal, A.: Modern information retrieval: a brief overview. BULLETIN OF THE IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON DATA ENGINEERING, 2001.

2. Mitra, B. and Craswell, N.: Neural models for information retrieval. 2017.

3. Wikipedia: Ocr. https://en.wikipedia.org/wiki/Optical_character_recognition. [Online] (Visited on 13-11-2019).

4. Smith, R.: An overview of the tesseract ocr engine. 2007.

5. Tarakeswar, K. and Kavitha, D.: Search engines:a study. Journal of Computer Applications (JCA), 4(1), 2011.

6. Mikolov, T., Chen, K., Corrado, G., and Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

7. Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T.: Bag of tricks for efficient text classification. CoRR, abs/1607.01759, 2016.

8. Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., and Fidler, S.: Skip-thought vectors. arxiv preprint arXiv:1506.06726, 2015.

9. Fuhr, N. and Buckley, C.: A probabilistic learning approach for document indexing. ACM Transactions on Information Systems (TOIS) - Special issue on research and development in information retrieval, 9(3), 1991.

10. Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T.: Enriching word vectors with subword information. CoRR, abs/1607.04606, 2016.

11. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L.: Deep contextualized word representations. CoRR, abs/1802.05365, 2018.

12. Aliprandi, S.: Juriswiki italia. http://juriswiki.it/. [Online] (Visited on 30-01-2019).

13. Brin, S. and Page, L.: The anatomy of a large-scale hypertextual web search engine. Comput. Netw. ISDN Syst., 30(1-7):107–117, April 1998.

14. Assefi, M.: Ocr as a service: An experimental evaluation of google docs ocr, tesseract, abbyy finereader, and transym. ISCV, 12 2016.

15. Facebook: fasttext unsupervised tutorial. https://fasttext.cc/docs/en/unsupervised-tutorial.html. [Online] (Visited on 10-10-2019).

16. Smith, R.: Das. In Tesseract Blends Old and New OCR Technology, 2016.

17. Facebook: fasttext. https://research.fb.com/fasttext/. [Online] (Visited on 13-11-2019).

18. Taylor, J.: Elmo: Contextual language embedding. https://towardsdatascience.com/elmo-contextual-language-embedding-335de2268604. [Online] (Visited on 07-05-2019).

19. Mihail, E.: Deep contextualized word representations with elmo. https://www.mihaileric.com/posts/deep-contextualized-word-representations-elmo/. [Online] (Visited on 07-05-2019).

20. Agarwal, S.: My thoughts on skip-thoughts. https://medium.com/@sanyamagarwal/my-thoughts-on-skip-thoughts-a3e773605efa. [Online] (Visited on 07-05-2019).

21. Ganegedara, T.: Intuitive guide to latent dirichlet allocation. https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-latent-dirichlet-allocation-437c81220158. [Online] (Visited on 25-04-2019).

22. Sik-Ho, T.: Review: Highway networks—gating function to highway (image classification). https://towardsdatascience.com/review-highway-networks-gating-function-to-highway-image-classification-5a33833797b5. [Online] (Visited on 07-05-2019).

23. CodeLingo: Keyword extraction using rake. https://codelingo.wordpress.com/2017/05/26/keyword-extraction-using-rake/. [Online] (Visited on 20-04-2019).

24. Shallue, C.: Skip-thought vectors. https://github.com/tensorflow/models/tree/master/research/skip_thoughts. [Online] (Visited on 07-05-2019).

25. AllenNLP: Elmo: Deep contextualized word representations. https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md#using-elmo-with-existing-allennlp-models. [Online].

CITED LITERATURE (continued)

26. Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S.: AllenNLP: A deep semantic natural language processing platform. In ACL workshop for NLP Open Source Software, 2018.

27. Marziali, M.: Cognitive and keyword-based search engine for legal documents corpus. 2019.

28. Ruder, S.: An overview of gradient descent optimization algorithms. CoRR, abs/1609.04747, 2016.

VITA

NAME	Matteo Marziali
EDUCATION	
	M.S. Computer Science, May 2020, University of Illinois at Chicago, USA. GPA: 4.0/4
	M.S. Computer Engineering, Jul 2019, Politecnico di Milano, Italy. Grade: 110/110
	B.S. Computer Engineering, Jul 2017, Politecnico di Milano, Italy. Grade: 104/104
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	A.Y. 2018/19 Six months of study abroad in Chicago, Illinois
	2018 - IELTS examination (7)
WORK EXPERIENCE AND PROJECTS	
Sep 2019 - Present	*Business Integration Partners - Data Scientist.*
Jan 2019 - Jul 2019	*Business Integration Partners - Data Scientist Intern.*
	Developed a search engine for scanned documents which both encompasses cognitive and keyword-based approaches combining DNN and frequency based word embedding models.