Corso di Dottorato di Ricerca in Informatica e Scienze Matematiche e Fisiche

XXX Ciclo

Advances in Automatic Keyphrase Extraction

Dottorando
Marco Basaldella

Supervisore
Federico Fontana

Co-supervisore
Carlo Tasso

Anno Accademico 2016/2017

ADVANCES IN AUTOMATIC KEYPHRASE EXTRACTION

MARCO BASALDELLA

Ph.D. Thesis
Department of Mathematics, Computer Science and Physics
Università degli Studi di Udine

Marco Basaldella: *Advances in Automatic Keyphrase Extraction,* © 2017

SUPERVISORS:
Prof. Carlo Tasso
Prof. Federico Fontana

Dedicated to the loving memory of
Aldo Pecile and Gianni Basaldella.

ABSTRACT

The main purpose of this thesis is to analyze and propose new improvements in the field of Automatic Keyphrase Extraction, i.e., the field of automatically detecting the key concepts in a document. We will discuss, in particular, supervised machine learning algorithms for keyphrase extraction, by first identifying their shortcomings and then proposing new techniques which exploit contextual information to overcome them.

Keyphrase extraction requires that the key concepts, or *keyphrases*, appear verbatim in the body of the document. We will identify the fact that current algorithms do not use contextual information when detecting keyphrases as one of the main shortcomings of supervised keyphrase extraction. Instead, statistical and positional cues, like the frequency of the candidate keyphrase or its first appearance in the document, are mainly used to determine if a phrase appearing in a document is a keyphrase or not. For this reason, we will prove that a supervised keyphrase extraction algorithm, by using only statistical and positional features, is actually able to extract good keyphrases from documents written in languages that it has never seen. The algorithm will be trained over a common dataset for the English language, a purpose-collected dataset for the Arabic language, and evaluated on the Italian, Romanian and Portuguese languages as well.

This result is then used as a starting point to develop new algorithms that use contextual information to increase the performance in automatic keyphrase extraction. The first algorithm that we present uses new linguistics features based on anaphora resolution, which is a field of natural language processing that exploits the relations between elements of the discourse as, e.g., pronouns. We evaluate several supervised AKE pipelines based on these features on the well-known SEMEVAL 2010 dataset, and we show that the performance increases when we add such features to a model that employs statistical and positional knowledge only.

Finally, we investigate the possibilities offered by the field of Deep Learning, by proposing six different deep neural networks that perform automatic keyphrase extraction. Such networks are based on bidirectional long-short term memory networks, or on convolutional neural networks, or on a combination of both of them, and on a neural language model which creates a vector representation of each word of the document. These networks are able to learn new features using the the whole document when extracting keyphrases, and they have the advantage of not needing a corpus after being trained to extract keyphrases from new documents, since they don't rely on corpus-based features like TF-IDF. We show that with deep learning based architectures we are able to outperform several other keyphrase extraction algorithms, both supervised and not supervised, used in literature and that the best performances are obtained when we build an additional neural representation of the input document and we append it to the neural language model.

Both the anaphora-based and the deep-learning based approaches show that using contextual information, the performance in supervised algorithms for automatic keyphrase extraction improves. In fact, in the methods presented in this thesis, the algorithms which obtained the best performance are the ones receiving more contextual information, both about the relations of the potential keyphrase with other parts of the document, as in the anaphora based approach, and in the shape of a neural representation of the input document, as in the deep learning approach. In contrast, the approach of using statistical and positional knowledge only allows the building of language agnostic keyphrase extraction algorithms, at the cost of decreased precision and recall.

PUBLICATIONS

Some ideas and figures have appeared previously, or will appear, in the following publications:

- Dario De Nart, Dante Degl'Innocenti, Andrea Pavan, Marco Basaldella, and Carlo Tasso. "Modelling the User Modelling Community (and Other Communities as Well)." In: *User Modeling, Adaptation and Personalization - 23rd International Conference, UMAP 2015, Dublin, Ireland, June 29 - July 3, 2015. Proceedings*. 2015, pp. 357–363

- Marco Basaldella, Dario De Nart, and Carlo Tasso. "Introducing Distiller: A Unifying Framework for Knowledge Extraction." In: *Proceedings of 1st AI*IA Workshop on Intelligent Techniques At LIbraries and Archives co-located with XIV Conference of the Italian Association for Artificial Intelligence, IT@LIA@AI*IA 2015, Ferrara, Italy, September 22, 2015*. Ed. by Stefano Ferilli and Nicola Ferro. Vol. 1509. CEUR Workshop Proceedings. CEUR-WS.org, 2015

- Dario De Nart, Dante Degl'Innocenti, Marco Basaldella, Maristella Agosti, and Carlo Tasso. "A Content-Based Approach to Social Network Analysis: A Case Study on Research Communities." In: *Digital Libraries on the Move - 11th Italian Research Conference on Digital Libraries, IRCDL 2015, Bolzano, Italy, January 29-30, 2015, Revised Selected Papers*. 2015, pp. 142–154

- Muhammad Helmy, Marco Basaldella, Eddy Maddalena, Stefano Mizzaro, and Gianluca Demartini. "Towards building a standard dataset for Arabic keyphrase extraction evaluation." In: *2016 International Conference on Asian Language Processing, IALP 2016, Tainan, Taiwan, November 21-23, 2016*. 2016, pp. 26–29

- Marco Basaldella, Giorgia Chiaradia, and Carlo Tasso. "Evaluating anaphora and coreference resolution to improve automatic keyphrase extraction." In: *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*. Ed. by Nicoletta Calzolari, Yuji Matsumoto, and Rashmi Prasad. ACL, 2016, pp. 804–814

- Marco Basaldella, Muhammad Helmy, Elisa Antolli, Mihai Horia Popescu, Giuseppe Serra, and Carlo Tasso. "Exploiting and Evaluating a Supervised, Multilanguage Keyphrase Extraction pipeline for under-resourced languages." In: *Recent Advances In Natural Language Processing 2017 (RANLP 2017), Varna (Bulgaria), September 4-6 2017*. 2017

- Marco Basaldella, Elisa Antolli, Giuseppe Serra, and Carlo Tasso. "Bidirectional LSTM Recurrent Neural Network for Keyphrase Extraction." In: *14th Italian Research Conference on Digital Libraries (IRCDL 2018), Udine, Italy, January 25-26, 2018 (Accepted for Publication)*. Udine, 2018

- Marco Basaldella, Giuseppe Serra, and Carlo Tasso. "The Distiller Framework: current state and future challenges." In: *14th Italian Research Conference on Digital Libraries (IRCDL 2018), Udine, Italy, January 25-26, 2018 (Accepted for Publication)*. 2018

Moreover, during my Ph.D. I had the opportunity and the luck to collaborate on the following papers, which topics are not covered in the present dissertation:

- Marco Basaldella, Lenz Furrer, Nico Colic, Tilia Ellendorff, Carlo Tasso, and Fabio Rinaldi. "Using a Hybrid Approach for Entity Recognition in the Biomedical Domain." In: *Proceedings of the 7th International Symposium on Semantic Mining in Biomedicine, SMBM 2016, Potsdam, Germany, August 4-5, 2016.* 2016, pp. 11–19

- Eddy Maddalena, Marco Basaldella, Dario De Nart, Dante Degl'Innocenti, Stefano Mizzaro, and Gianluca Demartini. "Crowdsourcing Relevance Assessments: The Unexpected Benefits of Limiting the Time to Judge." In: *Proceedings of the 4th AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2016)*. 2016

- Marco Basaldella, Lenz Furrer, Carlo Tasso, and Fabio Rinaldi. "Entity recognition in the biomedical domain using a hybrid approach." In: *Journal of Biomedical Semantics (Accepted for Publication)* (2017)

ACKNOWLEDGEMENTS

First and foremost, a huge thanks to my girlfriend Linda, my mother Annamaria, my father Franco and my sister Marta, who supported me during these three years.
A big thanks to my paper co-authors, to my Ph.D. colleagues in Udine and Zurich, to the lab mates at the Artificial Intelligence Laboratory, and to the whole academic staff in Udine; in particular, in a random order, I'd like to thank Eddy Maddalena, Lubos Vaci, Giuseppe Serra, Peter Makarov, Lenz Furrer, Andrea Baruzzo, Giorgia Chiaradia, and Stefano Mizzaro.
Many thanks go to Fabio Rinaldi and the Institut für Computerlinguistik of the University of Zürich for hosting me in Switzerland in the spring of 2016. It was a huge opportunity and honor to be a guest in such an important institution.
Lastly, I would like to thank my supervisor, prof. Carlo Tasso, for the invaluable support I received during these three years, for teaching me how to write a paper, how to organize myself when doing research, and for the uncountable suggestions he gave me during my career as a Ph.D. student.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AKEC	Arabic Keyphrase Extraction Corpus
API	Application Programming Interface
AR	Anaphora Resolution
Bi-LSTM	Bidirectional LSTM
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DL	Deep Learning
DLKE	Deep Learning Keyphrase Extraction
KE, AKE	Keyphrase Extraction, Automatic Keyphrase Extraction
KP	Keyphrase
NLM	Neural Language Model
NLP	Natural Language Processing
LSTM	Long Short Term Memory neural network
QA	Question Answering
PoS	Part of Speech
RNN	Recurrent Neural network

Part I

INTRODUCTION

This Part introduces the field of Automatic Keyphrase Extraction, by defining it and describing the typical approaches used by the research community to solve this problem. Moreover, in this part, we'll introduce the notation used throughout this work.

1 INTRODUCTION TO KEYPHRASE EXTRACTION

1.1 WHAT IS KEYPHRASE EXTRACTION?

Many scholars already gave their own definition of what *Keyphrase Extraction* (henceforth KE) is, thus we begin this thesis safely by standing on the shoulders of giants.

The historical definitions of KE are the following ones:

- Turney [115] defined keyphrase extraction as "the automatic selection of important, topical phrases from within the body of a document". Moreover, the author stated that "a keyphrase list [is defined] as a short list of phrases (typically five to fifteen noun phrases) that capture the main topics discussed in a given document."
- Witten et al. [121] defined keyphrases as phrases that "provide semantic metadata that summarize and characterize documents."
- According to Mihalcea and Tarau [80], "the task of a keyword extraction application is to automatically identify in a text a set of terms that best describe the document."

Let us consider, for example, the page from the online encyclopedia *Wikipedia* about itself [119]:

> Wikipedia is a free online encyclopedia with the aim to allow anyone to edit articles. Wikipedia is the largest and most popular general reference work on the Internet and is ranked the fifth-most popular website. Wikipedia is owned by the nonprofit Wikimedia Foundation.
>
> Wikipedia was launched on January 15, 2001, by Jimmy Wales and Larry Sanger. Sanger coined its name, a portmanteau of wiki and encyclopedia. There was only the English-language version initially, but it quickly developed similar versions in other languages, which differ in content and in editing practices. With 5,469,575 articles, the English Wikipedia is the largest of the more than 290 Wikipedia encyclopedias. Overall, Wikipedia consists of more than 40 million articles in more than 250 different languages and, as of February 2014, it had 18 billion page views and nearly 500 million unique visitors each month.
>
> As of March 2017, Wikipedia has about forty thousand high-quality articles known as Featured Articles and Good

> Articles that cover vital topics. In 2005, Nature published a peer review comparing 42 science articles from Encyclopædia Britannica and Wikipedia, and found that Wikipedia's level of accuracy approached that of Encyclopædia Britannica.
>
> Wikipedia has been criticized for allegedly exhibiting systemic bias, presenting a mixture of "truths, half truths, and some falsehoods", and, in controversial topics, being subject to manipulation and spin.

Some example keyphrases (abbreviated KPs) may be "*Wikipedia*", "*online encyclopedia*", "*Jimbo Wales*", "*Larry Sanger*", "*250 different languages*", "*40 million articles*", and so on. These are my personal keyphrases; however, an individual biased against Wikipedia's open edit policy may choose to add "*system bias*" to the list, while someone who blindly supports the online encyclopedia's philosophy may decide to ignore its potential weaknesses and disregard it.

This problem arises because *all* the above definitions of keyphrase extraction rely on some very non-formal terms, suggesting that, to extract keyphrases, we should "*characterize documents*", identify their "*main topics*", or find the words that "*best describe*" them. Unfortunately, this non-formality is innate in the task of keyphrase extraction, which is subjective by its own nature. The Wikipedia example showed how easy is to find controversial keyphrases even for a small document.

Even if keyphrase extraction is, due to this ambiguity, an inherently difficult task, it has application in several domains, such as document summarization [125], document clustering [50], recommender systems [99], user modeling [89], and so on. Moreover, AKE has received widespread attention in the research community: for example, KE has been the subject of two different SEMEVAL competitions in 2010 and 2017 [8, 63].

The remainder of this Chapter will be devoted to the description of the principal techniques which has been used to attempt to solve the KE task, the principal datasets used in the KE community, and the definitions and the notation I will use throughout this thesis.

To end this section, we give our definition of the task of Keyphrase Extraction in Definition 1.1.1.

Definition 1.1.1 (Automatic Keyphrase Extraction)**.** The task of Automatic Keyphrase Extraction (AKE) is the problem of automatically identifying a list of short phrases (typically, from one to five words long) in the content of a document that allow someone to grasp its content *without* actually reading it.

1.2 ALGORITHMS FOR KEYPHRASE EXTRACTION

The performance of the state of the art KE systems is still much lower than many other NLP tasks. A hint of the current top performing systems can be obtained by looking at the results of "SEMEVAL 2010 Task 5: Automatic Keyphrase Extraction from Scientific Articles" [63], where systems were ranked by F-score on the top 15 extracted keyphrases. The best system, presented by Lopez and Romary [72], achieved a F1-score of 27.5. Seven years later, a new SEMEVAL competition was held, and the best system achieved an F1-Score of 43.0 [8]. While the performance improvement may seem important, it is worth noting that the documents SEMEVAL 2010 competition were full papers, while for SEMEVAL 2017 only abstract were used. This difference is crucial, as it is apparently easier to extract keyphrases from shorter documents [52]: for example, Hulth [58] obtained results comparable with the SEMEVAL 2017 competition more than ten years before, but on the Inspec abstract corpus [57].

We can categorize the algorithms for keyphrase extraction in three categories. The first two make up the "classic" approaches to KE, namely supervised machine learning based KE and unsupervised graph-based KE. The third approach, that we'll call *deep* keyphrase extraction (abbreviated DLKE), emerged only recently, and while it is still based on supervised machine learning (henceforth ML), its huge differences with "classic" supervised extraction deserves an analysis on its own.

1.2.1 *Supervised Keyphrase Extraction*

The first algorithms to appear in literature were the Supervised AKE ones, in the late 90s of the last century [115, 121]. Since then, all the supervised algorithms had almost the same overall structure:

1. First, low-level NLP operations are performed, i.e. sentence splitting, tokenization, part-of-speech tagging, and stemming;

2. The resulting information is used to generate the candidate keyphrases, for example by selecting n-grams that match with certain part-of-speech patterns or using chunking algorithms [57];

3. Each candidate keyphrase is associated with some *feature*, e.g. its frequency in the document or in a corpus, the number of words in the candidate keyphrase, and so on;

4. Each candidate keyphrase is *scored* by a machine learning algorithm trained using the values of the features calculated in the previous step and a corpus of documents and their human assigned keyphrases.

Many of the most influential or successful methods are based on this simple structure. For example, of the works cited until now, Witten et al. [121], Turney [115], Hulth [57], and Lopez and Romary [72] rely on this approach.

There is no general agreement on what is the best ML algorithm for keyphrase extraction. A non comprehensive list of ML algorithms used throughout the year contains Support Vector Machines [72, 118], decision trees [11, 48, 72, 115], neural networks [11, 16, 72], logistic regression [11, 48], Naïve Bayes [121], and so on.

However, there is *even less* agreement on what are the best features for keyphrase extraction. Many scholars build on top of KEA's original features [121], e.g. Hulth [57, 58], Kim and Kan [62], Lopez and Romary [72], and Nguyen and Kan [90] and the author himself in Basaldella, Chiaradia, and Tasso [11] and Basaldella et al. [16]. KEA used only two features: TF-IDF and the position of the first appearance of the keyphrase in the document, which have been used as an inspiration for a number of similar features. For example, Haddoud and Abdeddaim [47] and Kim and Kan [62] develop their own variants of TF-IDF by calculating it over specific sections of the input documents; other scholars use the position of the *last* appearance of the keyphrase as a feature as well, like for example Kim and Kan [62] and Pudota et al. [99], while Medelyan, Frank, and Witten [77] and Pudota et al. [99] use the length of the portion of the document between the first and the last appearance of the keyphrase as a feature.

In Table 1.1 we collected a list of over 60 features used in supervised keyphrase extraction. While the list is by no means exhaustive, since it has been compiled by using only about 15 papers from the KE domain, it can help to give an idea of the type of features that are used by scholars.

The features found can be categorized by their *format*, i.e., they are either *numbers*, *strings*, or *Booleans*, and by how they are calculated, i.e. using only the input document, using a corpus, or using an external source, like e.g., a search engine [116] or Wikipedia [87]. It is straightforward to note that the majority of the features are numeric and document-based. Most of the features, in fact, rely on the frequency of a keyphrase in different variations, e.g. on the frequencies of the words that compose the keyphrase, of their stems, and so on.

Feature	**Meaning**	**Data**	**Type**	**Used in**
Number of words	Number of words in the candidate keyphrase	N	D	[48, 62, 72, 115]
Number of characters	Number of characters in the candidate keyphrase	N	D	[76]

Feature	Meaning	Data	Type	Used in
Candidate first occurrence	First occurrence of the stemmed phrase in the document, counting with words	N	D	[39, 48, 57, 62, 72, 87, 90, 99, 115, 121]
Candidate last occurrence	Last occurrence of the stemmed phrase in the document, counting with words	N	D	[62, 87, 99]
Lifespan on sentences	Difference between the last and first appearance in the document	N	D	[77, 87, 99]
Candidate stem first occurence	First occurrence of a stemmed word of the candidate, counting with words	N	D	[115]
Normalized phrase frequency (TF)	Frequency of the stemmed phrase in the document (TF)	N	D	[48, 57, 87, 99, 115]
Relative length	Number of characters of the candidate	N	D	[115]
Proper noun flag	Candidate is a proper noun	B	D	[115]
Final adjective flag	Candidate ends with an adjective	B	D	[115]
Verb flag	Candidate contains a known verb	B	D	[115]
Acronym flag	Candidate is an acronym	B	D	[62, 90]
TF-IDF over corpus	TF-IDF of the candidate in the corpus	N	C	[39, 47, 48, 57, 62, 72, 76, 90, 115, 116, 121]
Keyphrase frequency	Frequency of the candidate as a keyphrase in a corpus	N	C	[39, 72, 116]
Candidate frequency	Frequency of the candidate in the corpus	N	C	[57]
POS sequence	Sequence of the POS tags of candidate	S	D	[57, 76, 90]
Distribution of the POS sequence	Distribution of the POS tag sequence of candidate in the corpus	N	C	[62]
Number of named entities	Number of named entities in the candidate	N	D	[76]
Number of capital letters	Used to identitify acronyms	N	D	[76]
IDF over document	Inverse document frequency	N	D	[48]
Variant of TF-IDF - 1	log(TF-IDF)	N	C	[48]

Feature	**Meaning**	**Data**	**Type**	**Used in**
First sentence	First occurrence of the phrase in the document, counting with sentences	N	D	[48]
Head frequency	Number of occurrences of the candidate in the first quarter of the document	N	D	[48]
Average sentence length	Average length of the sentences that contain a term of the candidate	N	D	[48]
Substring frequencies sum	Sum of the term frequency of all the words that compose the candidate	N	D	[48]
Generalized Dice coefficient	See [48, 72]	N	D	[48, 62, 72]
Maximum likelihood estimate	Estimation of the probability of finding the candidate in the document	N	D	[48]
Kullback-Leibler divergence	See [48]	N	C	[48]
Document phrase maximality index (DPM)	Ratio of the frequency of the keyphrase and its superstrings	N	D	[48]
DPM × TF-IDF	Self-explanatory	N	C	[48]
Variant of TF-IDF - 2	TF-IDF of the candidate / TF-IDF of its most important word	N	C	[48]
k-means of the position	See [48]	N	C	[48]
GRISP presence	Presence in the GRISP database	B	E	[72]
Wikipedia keyphraseness	Probability of the candidate to be an anchor in Wikipedia	N	E	[72]
Title presence	Presence of the candidate in the title	B	D	[72]
Abstract presence	Presence of the candidate in the abstract	B	D	[72]
Introduction presence	Presence of the candidate in the introduction	B	D	[72]
Section title presence	Presence of the candidate in a title of a section	B	D	[72]
Conclusion presence	Presence of the candidate in the conclusions	B	D	[72]

Feature	**Meaning**	**Data**	**Type**	**Used in**
Reference or book title presence	Presence of the candidate in at least one reference or book title	B	D	[72]
Variant of TF-IDF - 3	TF includes the TF of substrings of the candidate	N	C	[62]
Variant of TF-IDF - 4	TF of substrings of the candidate without the TF of the candidate	N	C	[62]
Variant of TF-IDF - 5	TF normalized by candidate types (noun phrases vs simplex words vs...)	N	C	[62]
Variant of TF-IDF - 6	TF normalized by candidate types as a separate feature	N	C	[62]
Variant of TF-IDF - 7	IDF using Google n-grams	N	E	[62]
Section information	Weight the candidate based on its location (abstract, title, ...)	N	D	[62, 90]
Section TF	TF of the candidate in key sections	N	D	[62]
Candidate co-occurrence	Number of sections in which the candidates co-occur	N	D	[62]
TF Occurence in titles	Occurrence in the CiteSeer title collection as substring of a title	B	E	[62]
Occurence in titles	TF of the candidate in the CiteSeer title collection as substring of a title	N	E	[62]
Semantic similarity - 1	Contextual similarity among candidates	N	D	[62]
Semantic similarity - 2	Semantic similarity among candidates using external knowledge	N	E	[116]
Variant of Dice coefficient - 1	Normalized TF by candidate types (noun phrases vs simplex words...)	N	D	[62]
Variant of Dice coefficient - 2	Weighting by candidate types (noun phrases vs simplex words...)	N	D	[62]
Variant of Dice coefficient - 3	Normalized TF and weighting by candidate types (noun phrases vs simplex words...)	N	D	[62]

Feature	Meaning	Data	Type	Used in
Suffix sequence	Sequence of the suffixes of the words that from the candidate	S	D	[62, 90]
Wikiflag	Presence of the candidate as a Wikipedia page title or surface (e.g. Big Blue vs IBM)	B	E	[87]
Noun value	Number of nouns in the candidate	N	D	[87, 99]

Table 1.1: A non-comprehensive list of features used in supervised keyphrase extraction. In the column **Data**, the letter **N** marks a feature which value is *numeric*, the letter **B** marks a *boolean* feature and the letter **S** marks a feature where the value is a *string*. In the column **Type**, the letter **D** marks a feature that is calculated using only the input document, the letter **C** marks a *boolean* feature that is calculated using an external corpus, and the letter **E** marks a feature that is calculated using some other kind of external knowledge (e.g. an ontology).

TF-IDF is undoubtedly one of the most influential features, since it was used by almost all the supervised algorithms and we were able to identify seven different variants of this feature as well. In such variants, the TF-IDF value is normalized in different ways, or calculated using external knowledge, or multiplied with the value of other features. Other numerical features are calculated using statistical techniques, such as the Generalized Dice Coefficient used in Haddoud et al. [48], Kim and Kan [62], and Lopez and Romary [72], the frequency of the keyphrase in particular sections of the input document (e.g. the abstract), or by counting the number of letters that compose the keyphrase, the number of capital letters, and so on.

Only few features rely on linguistics; for example, [57] uses the sequence of part-of-speech tags of the words composing the keyphrase as a feature in string format. Pudota et al. [99] counts the number of nouns that appear in a keyphrase, while Kim and Kan [62] and Nguyen and Kan [90] follow another approach, by feeding the ML model with the sequences of the suffixes of the word that compose the candidate keyphrase (for example, the suffix sequence of "suffix sequence" is "`fix nce`"). This feature can be useful to catch certain technical words belonging to specific fields, e.g. words ending with "*-itis*" are very likely to be a disease (e.g. "*arthritis*"), and is indeed typical of the field of entity recognition in specialized domains [14]. Finally, Turney [115] used Boolean flags to check whether the candidate keyphrase contained a verb or ended with an adjective, but this feature has been superseded by the introduction of the candidate generation phase based on part-of-speech regular expressions [30, 47, 57, 58, 62, 72, 90, 99] which guarantees that the candidate keyphrases are well-formed phases (e.g. not ending with determiners).

Some features instead use *external knowledge* trying to provide valuable information to the ML model. Most notably, the winner system of the SEMEVAL 2010 competition, presented in [72], used the presence of the candidate in Wikipedia and in the technical term database GRISP [71] as a feature, starting from "the assumption [...] that a phrase which has been identified as controlled term in these resources tend to be a more important keyphrase." Nart and Tasso [87] used a Boolean flag which was set to `true` when the candidate keyphrase was the title of a page in Wikipedia. Kim and Kan [62] used Google Ngrams [79] and other scholars used search engines [116] to compute variants of TF-IDF on a wider corpus.

1.2.2 *Unsupervised Keyphrase Extraction*

Since this dissertation is mainly about supervised KE, I will not go deep in the details of unsupervised KE methods. However, a brief analysis of the history of such methods is important, because supervised and unsupervised KE techniques evolved somehow in parallel.

Historically, one of the first unsupervised approaches was proposed by Barker and Cornacchia [10], and it was a simple technique based on the extraction and filtering of noun phrases. Then, Tomokiyo and Hurst [112] proposed a technique for unsupervised KE which used a statistical language model to rank KPs.

Later, Mihalcea and Tarau [80] proposed a novel technique of extracting KPs called TextRank, where the document is represented using a graph structure, whose nodes are candidate KPs. Then, the *popularity* of each candidate is evaluated using graph algorithms usually derived from the PageRank algorithm [94]. Other scholars, most notably Bougouin, Boudin, and Daille [23], Litvak, Last, and Kandel [68], and Wan and Xiao [117] evolved the original TextRank algorithm with different goals, e.g. splitting the document in topics [23] or ensuring the language-independence of the algorithm [68]. The TextRank method was also challenged by Liu et al. [69], who were able to outperform the graph-based techniques using a novel approach based on clustering.

In general, unsupervised KE approaches obtained wide attention by the academic community, due to their inherent advantages of being both domain-independent and easier to port to different languages. However, to date, their performance has been lower when supervised and unsupervised techniques were evaluated in the same setting. In fact, in both the SEMEVAL 2010 [63] and 2017 [8] tasks on keyphrase extraction the winner algorithms were based on supervised techniques.

For a more exhaustive review of unsupervised KE methods, the author suggests the survey by Hasan and Ng [52], which covers also methods not mentioned in this chapter.

1.2.3 *Deep Learning and Keyphrase Extraction*

Deep Learning (hence DL), i.e. the application of neural networks with many hidden layers, is a relatively new branch of Artificial Intelligence that gained popularity in the early 2010s due to the availability of new software and hardware tools that made the training of such networks feasible in short time. DL revolutionized many fields, from image classification to speech recognition [43], and it found many applications in Natural Language Processing (abbrv. NLP) thanks to the concurrent discovery of word embeddings by Mikolov et al. [81]. Chapter 4 will serve as a brief introduction of the DL concepts relevant to KE.

The applications of DL on keyphrase extraction still are relatively few. For example, Zhang et al. [123] used Recurrent Neural Networks to extract keyphrases from Tweets. Meng et al. [78] used deep neural networks to extract and generate keyphrases from scientific publications, obtaining good results on many datasets. At the time of writing, the most recent examples of DLKE are the systems proposed at the SEMEVAL 2017 competition [8], where the winner system was based on Long-Short Term Memory networks (LSTM) [6]. Actually, the top three systems in SEMEVAL 2017 were all based on DL techniques, using DL in combination with other supervised machine learning algorithms (in particular, with Conditional Random Fields).

1.3 DATASETS

1.3.1 *English datasets*

The most popular language for AKE in academia is undoubtedly English, since most of the datasets are available in this language; the only two competitions organized on keyphrase extraction, SEMEVAL 2010 [63] and 2017 [8], used English datasets. In particular, the results of 2010 edition of the competition served as reference for many scholars afterwards, since they had a reference dataset and shared evaluation scripts to assess their work [11, 16, 23, 47, 48, 78]. At the time of writing, it is still to early to tell if the 2017 edition will have the same impact on the AKE community as the 2010 edition, but given the success of the previous one, it is likely that it will be the case.

Other publicly available datasets worth mentioning here are the INSPEC corpus, introduced by Hulth [57], and later used by other scholars to compare with her work, as for example Meng et al. [78], Mihalcea and Tarau [80], and Pudota et al. [99] did. Wan and Xiao [117] evaluated their work on documents from the DUC 2001 dataset, a dataset for text summarization from the 2001 edition of the Document Understanding Conferences series, while Nguyen and Kan [90] collected scientific papers from the internet instead.

The typical approach to collect keyphrases for all the previously mentioned datasets is to have "experts" annotating the documents. These experts can be either the author(s) of the document, or university students, or both. However, it is very rare that keyphrases are produced by more than one annotator, and it is instead very common that the quality of the annotations is low. According to Meng et al. [78], who analyzed some of the previously mentioned datasets, in some of them *more than 50%* of the annotated keyphrases are not present in the documents.

However, recently, new approaches to collect and evaluate KPs using *crowdsourcing* emerged. Crowdsourcing, portmanteau of *crowd* and *outsourcing*, is the act of using internet users (the *crowd*) as workforce to solve a problem or execute a task, and it has recently gained popularity in both academia and industry as a quick and relatively cheap solution for obtaining data. Marujo et al. [76] collected an AKE dataset for English using Amazon Mechanical Turk, while Chuang, Manning, and Heer [26] used crowd workers to rate the quality of automatically selected KPs. We will continue to investigate the use of crowd sourcing for KE in Chapter 2.

1.3.2 *Non-English Keyphrase Extraction*

Even if the majority of the datasets for KE are available for the English language, this does not necessarily mean the AKE is an English-only field. In fact, scholars started working with algorithms for multi lingual AKE from the beginning. For example, Tseng [113] used AKE in order to provide approaches for IR users to handle multilingual documents, by building an unsupervised, language-independent AKE system and demonstrating its effectiveness on English and Chinese documents. Since then, many language-independent approaches have been proposed but most of them are unsupervised, as anticipated in Section 1.2.2. Unfortunately, due to the lack of non-English resources, such algorithms often require the collection of ad-hoc corpora when evaluated on different languages.

For example, DegExt [68] was introduced as an unsupervised language independent keyphrase extractor. DegExt uses a simple graph-based syntactic representation of text and web documents [107]. The evaluation was performed on an English corpus (DUC 2002) and on one purpose-built corpus of 50 Hebrew documents. Paukkeri and Honkela [95] and Paukkeri et al. [96] presented Likey, a language-independent keyphrase extraction method based on statistical analysis and the use of a reference corpus. Likey has been tested using exactly the same configuration with 11 European languages using an automatic evaluation method based on Wikipedia intra-linking, and on the English language using the SEMEVAL 2010 corpus [63]. Bougouin, Boudin, and Daille [23] proposed TopicRank as an unsupervised, language-independent AKE system. TopicRank creates a graph of the document where each node is a topic appearing in the document, then it uses graph ranking techniques to score the topics and select keyphrases belonging to the top ranked ones. The authors evaluate their approach on four datasets, two on the English language (the SEMEVAL 2010 corpus [63] and the Inspec dataset [57]), and two on the French language, one freely available and one purpose-built by them.

However, supervised systems were used for multi language keyphrase extraction as well. DIKpE-G [30] was proposed as a novel multi-language, unsupervised, knowledge-based approach towards keyphrase generation. DIKpE-G integrates several kinds of knowledge for selecting and evaluating meaningful keyphrases, ranging from linguistic to statistical, meta/structural, social, and ontological knowledge, and it has been evaluated on the Italian language using a custom-built dataset of 50 scientific papers. LIKE [7] was presented as a supervised method that uses feed-forward neural networks for automatically extracting keywords from a document regardless of the language used in it. Unfortunately, while the authors claim that LIKE is a truly language-independent AKE system, it is trained and evaluated only over the English language.

There is a growing community of scholars working on AKE on the Arabic language as well. For example, in order to train and evaluate their KP-Miner system for Arabic AKE, El-Beltagy and Rafea [32] used manually annotated documents, including Wikipedia articles and their meta-tags. Awajan [9] instead used a mixed approach of previously tagged document and manually annotated ones as training and evaluation dataset. However, in this dataset, only 73% of the human-generated keywords are actually found in the text, severely undermining the quality of the dataset and the performance of the KPE algorithm itself.

1.4 APPLICATIONS OF AUTOMATIC KEYPHRASE EXTRACTION

Keyphrases have been widely used in many applications throughout the years in many fields, e.g. in document indexing [45], clustering [45, 50, 51], similarity [120], and summarization [124]. The author contributed to two applications of AKE as well, in the fields of User Modeling (Nart et al. [89]) and of Social Network Analysis (Nart et al. [88]).

We will not analyze these works in depth, as this thesis is about improving AKE, rather than on its applications. However, as an example of application of AKE, in Figure 1.1 we show the temporal analysis of the evolution of the topics in the Italian Research Conference on Digital Libraries. As we have described in Nart et al. [88], in order to achieve temporal modeling, papers are grouped by year, then a

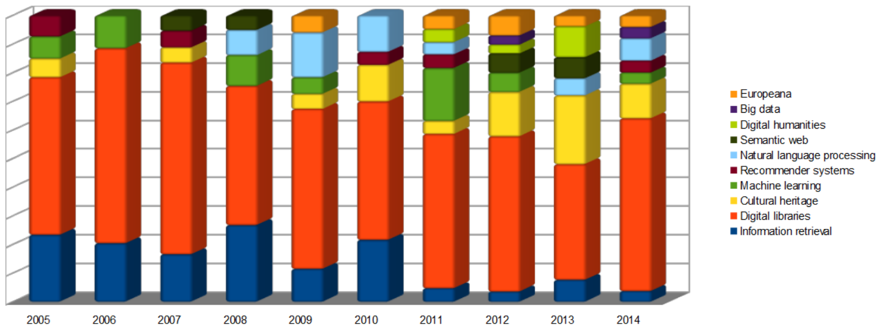


Figure 1.1: An example application of AKE: the temporal evolution of the topics of the IRCDL conference, reproduced from Nart et al. [88].

topic graph is built by means of AKE, under the assumption that the size of a fraction of papers including a specific term is a significant measure of how much widespread such term is at a specific time. We can see that "Digital Libraries", i.e. the topic of the conference, is by far the most important topic in the papers accepted over the ten year time span we analyzed. On the other hand, we can also notice how the first editions covered less topic than the most recent ones, with an minimum of three topics in 2006, and maximum of 8 topics in 2011, 2012 and 2014. This happened due to the emergence of new trends, i.e. the appearance of topics like "Europeana", "Big Data", or others, which were not covered in the first editions.

1.5 TERMINOLOGY

Throughout this work, unless stated otherwise, I will use the terminology defined in Table 1.2. Mathematical operators are used with their usual meaning, e.g., $\vec{x}$ is a vector, $|D|$ is the cardinality of the set D, and so on.

Symbol	**Definition**
D	A document corpus
$d \in D$	A document belonging to a corpus D
S_d	The set of the sentences belonging to a document d
$s_{d,i} \in S_d$	The i-th sentence of the document d
kp	A keyphrase
t	A token
$\mathtt{freq}(t, d)$ and $\mathtt{freq}(kp, d)$	Functions that calculate the frequency of t or of kp in a document d.

Table 1.2: Terminology used in the present thesis.

1.6 EVALUATION

The task of evaluating an AKE algorithm is not a simple one. In fact, several techniques have been investigated throughout the years in order to assess the performance of AKE. However, the most common used metrics are *precision*, *recall* and *f-measure*.

Definition 1.6.1 (Precision)**.** Let $KP_{correct}$ be the set of the correctly extracted keyphrases and $KP_{extracted}$ the set of the extracted keyphrases (both correct and wrong), the **precision** P of an AKE algorithm is defined as follows:

$$P = \frac{|KP_{correct}|}{|KP_{extracted}|} \tag{1.1}$$

i.e. the ratio between the number of correctly extracted keyphrases, and the number of all the extracted keyphrases.

Definition 1.6.2 (Recall)**.** Given $KP_{correct}$ as before and let KP_{gold} be the set of the human-annotated keyphrases, the **recall** R of an AKE algorithm is defined as follows:

$$R = \frac{|KP_{correct}|}{|KP_{gold}|} \tag{1.2}$$

i.e. the ratio between the number of correctly extracted keyphrases, and the number of human-annotated keyphrases (i.e. the size of the gold standard).

Definition 1.6.3 (F-Measure)**.** Let P and R precision and recall, the **F-measure** F1 is defined as follows:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{1.3}$$

i.e. the harmonic mean of P and R.

Note that these metrics are often used on a subset of the keyphrases extracted by an AKE systems, namely on the top n; in this case, the metrics are called "Precision@n" , "Recall@n", "F1-Score@n", abbreviated respectively as P@n, R@n, F1@n. For example, the systems trained on the SEMEVAL 2010 dataset are typically evaluated using $n = 5$, $n = 10$ and $n = 15$ [63]. However, these values are not the standard ones: for example, Meng et al. [78] evaluate their system with F1@5 and F1@10 only, and El-Beltagy and Rafea [32] use $n = 7$, $n = 15$ and $n = 20$ as thresholds.

Some scholars used other metrics to evaluate their own systems. Litvak, Last, and Kandel [68] used Area Under Curve (AUC), which is the area under the Receiver Operating Characteristics (ROC) curve. An efficient algorithm to calculate AUC is presented by Fawcett [37].

Liu et al. [70] introduced *binary preference measure* (Bpref) and *mean reciprocal rank* (MRR) as tools to evaluate AKE. Bpref is used to evaluate the performance of AKE algorithms considering *the ordering* of the extracted keyphrases, while MRR is used to evaluate how the first correct keyphrase for each document is ranked.

Definition 1.6.4 (Binary Preference Measure (Bpref))**.** Given a document and a KP extraction system, M is the total number of keyphrases extracted by the system, of which R are correct. We say that r is a correctly extracted KP and n is an incorrectly extracted KP. So, the **binary preference measure** is defined as follows:

$$\text{Bpref} = \frac{1}{R}\sum_{r \in R}\left(1 - \frac{|n \text{ ranked higher than } r|}{M}\right) \tag{1.4}$$

Definition 1.6.5 (Mean reciprocal rank (MRR))**.** Given a document set D, a document $d \in D$, the rank of the first correct keyphrase within all the extracted keyphrases is denoted as $rank(d)$. Then, the **mean reciprocal rank** is defined as follows:

$$\text{MRR} = \frac{1}{|D|}\sum_{d \in D}\frac{1}{rank(d)} \tag{1.5}$$

Marujo et al. [76] instead proposed *normalized Discounted Cumulative Gain* (nDCG) as a metric to evaluate their system against the crowd-assigned keyphrases, and Schluter [108] cited the same metric as ideal for evaluating an AKE system. Note that this metric needs a dataset where the KPs are ranked to be used, so it is not applicable on most of the datasets mentioned in Section 1.3. Here we use the definitions of DCG and nDCG as presented by Marujo et al. [76]:

Definition 1.6.6 (Discounted cumulative gain (DCG))**.** Let rel_i be the number of workers which selected the i-th phrase as relevant, the **Discounted Cumulative Gain** is defined as follows:

$$\text{DCG} = rel_1 + \sum_{i=2}^{n}\frac{rel_i}{\log_2 i} \tag{1.6}$$

Definition 1.6.7 (Normalized Discounted cumulative gain (nDCG))**.** Given DCG, and let iDCG be the ideal ordering for a list of keyphrases, the **normalized Discounted Cumulative Gain** is defined as follows:

$$n\text{DCG} = \frac{\text{DCG}}{i\text{DCG}} \tag{1.7}$$

In this thesis we will mainly use Precision, Recall and F1-Score as metrics, but we believe that ranking has to be held in account when evaluating a system as well. In fact, we suggest that an AKE system may offer an interesting contribution even if it is just able to provide a

A	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
B	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
C	✓	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓

	P	R	F1	MAP
A	0.33	0.33	0.33	0.33
B	0.33	0.33	0.33	0.02
C	0.40	0.40	0.40	0.31

Table 1.3: Hypothetical Precision, Recall, F1-Score and MAP of three systems A, B and C over a document with 15 correct keyphrases. A tick mark (✓) indicates a system assigned correct keyphrase, while an x mark (✗) a wrong one.

better ranking of the keyphrases. A good system is, arguably, one that puts the better keyphrases on the top of the output, while leaving the bad, potentially wrong, keyphrases on the bottom.

However, this better ranking could not be caught by Precision, Recall and F1. For example, suppose we have two systems participating in SEMEVAL 2010, where algorithms are ranked by the F1-Score of the top 15 keyphrases returned by the algorithms. We call this systems A and B. Suppose that, looking at A's output, only the first 5 keyphrases are correct, while the other 10 are wrong. Then, suppose that B's output is the opposite, i.e. the first 10 keyphrases are wrong, and the last 5 are good. Thus, these systems will have the same precision (33%), the same recall and the same F1-Score, but the ranking of the system A is arguably better than the ranking provided by B.

Therefore, when necessary to evaluate the ranking produced by AKE systems, we will use the Mean Average Precision (MAP) metric as an alternative to the metrics presented in this section, as it is more suitable to evaluate a ranking than simply using precision and recall. MAP is easy to calculate (differently from AUC), it takes into account the ranking of all the KPs selected by the system (differently from MRR), it is closely related to the already used P@n metric (differently from Bpref), and it does not need a ranked output (differently from nDCG). We give our definition of MAP for keyphrase extraction by adapting the definition given by Manning, Raghavan, and Schütze [74] in Definition 1.6.8.

Definition 1.6.8 (Mean Average Precision (MAP)). Let $D = \{kp_{(D,1)}, \ldots, kp_{(D,n)}\}$ be the set of correct keyphrases for a document and $R_{D,k}$ be the set of retrieved keyphrases for the document D until you get to the k-th keyphrase, the **Mean Average Precision** is defined as

$$\mathrm{MAP}(D,R) = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{j}\sum_{k=1}^{j}\mathrm{Precision}(R_{D,k})$$

Where $\mathrm{Precision}(R_{D,k})$ is the Precision@k score of the system over document D for the first k retrieved documents.

As an example, we can see the systems A and B compared in Table 1.3, on a document with 15 gold keyphrases. The two systems share the same P, R and F1 scores, while MAP is radically different, being much higher for system A than for system B. If we suppose to have another system C, which gets the 1^{st},2^{nd},3^{rd},5^{th},10^{th} and 15^{th} keyphrases correct, we have that C shows higher P, R and F1 scores than systems A and B but lower MAP than A. This is due to A's better quality of the first results or, in other words, because of the higher *"weight"* of A's correct keyphrase in the 4^{th} position than C's correct keyphrases in 10^{th} and 15^{th} position.

1.7 RESEARCH QUESTION

The purpose of this thesis is to analyze the strengths and weaknesses of the current supervised approach to the automatic keyphrase extraction problem, and to propose new methods based on linguistics and deep learning that which can be helpful in the improvement of AKE using more contextual information.

As seen in Section 1.2, supervised AKE algorithms are currently considered the state-of-the art approach. However, as stressed in Section 1.2.1, at present most of the features used in such systems rely mostly on statistical, language-independent features: when training a model, in fact, scholars usually feed it neither with the whole document, nor even with the words that compose the keyphrase. This means that a supervised model actually "knows" very little of what the input document is about, what are its topics, what are the relationships between the keyphrases that appear in it, and so on.

For this reason, we will try to prove these claims:

- First, that the "classic" approach to supervised AKE, not receiving any information about the content of the input document, is basically language-independent. We will show that training a machine learning model for supervised AKE on a language yields good performance on other languages as well.
- Starting from this point, we will try to add more contextual information in supervised AKE, by using features based on linguistics and in particular on *anaphora resolution*.
- Finally, we will try to exploit deep learning techniques based on neural language models to build a supervised AKE algorithm that actually *reads* the document.

1.8 CONTRIBUTIONS AND OUTLINE

In order to prove the claims we made in Section 1.7, we developed several experiments which are described throughout the present dissertation. All these experiments have been developed in the Artificial Intelligence Laboratory at the university of Udine. Below, we summarize the main contributions presented in the present Thesis:

- Chapter 1, i.e. the present Chapter, presented the problem of Automatic Keyphrase Extraction. We investigated the current state-of-the-art approaches, we presented the most common dataset for AKE and the main metrics used to evaluate AKE algorithms.
- Chapter 2 covers the collection a new AKE dataset for the Arabic language, and its application in developing a multilanguage AKE pipeline. The work covered in this Chapter has been published in Basaldella et al. [16] and Helmy et al. [54].
- Chapter 3 describes the process of using linguistic in order to use contextual information to improve AKE. The results of this Chapter have been published in Basaldella et al. [14].
- Chapters 4 and 5 cover the applications of Deep Learning on AKE and the development of several DLKE pipelines that perform AKE by using the whole document. A seminal version of the work presented in has been accepted for publication in Basaldella et al. [17].

The experiments covered in Chapters 2 and 3 have been developed using the Distiller framework, a framework that the author developed during the course of its Ph.D. in order to provide the research community a flexible testbed for AKE and similar information extraction tasks. The Distiller framework is described in Appendix A, and it has been presented in Basaldella, Nart, and Tasso [12], and later its improvements have been analyzed in Basaldella, Serra, and Tasso [13]. We released all the code of the Distiller framework as Open Source software.

Appendix B contains the description of the software developed in order to perform the algorithms presented Chapter 5. As for the Distiller framework, we plan to release the source code presented in this Appendix as Open Source software at the moment of the publication of the present dissertation.

1.9 OTHER CONTRIBUTIONS

Since the present dissertation is mainly about AKE, it does not cover some of the works developed using the course of the author's Ph.D.. These works include:

- The assessment of the performance of workers in Crowdsourcing experiments with different time constraints. This work, presented in Maddalena et al. [73], has been helpful to gain experience in the Crowdsourcing field, and some of the concepts learned while performing this experiments have been reused while collecting the AKEC dataset presented in Chapter 2;

- The application of AKE techniques on dictionary-based entity recognition on the biomedical domain, and the subsequent integration of the Distiller with the OntoGene software developed by the university of Zurich [100] to build an hybrid dictionary-based/machine learning entity extraction system for biomedical entities. These works have been presented in Basaldella et al. [14, 15], they have been influential in some of the development choices of the Distiller framework, and they have been very helpful to gain experience in a natural language processing task closely related to AKE.

Part II

CLASSICAL SUPERVISED KEYPHRASE EXTRACTION

This Part describes the relationship between *language* and Automatic Keyphrase Extraction. First, we show how, due to the nature of supervised AKE algorithms, the input language seems to be not important at all, since many (if not all) of the classical supervised approaches do not rely on the *meaning* of the words contained in the document, but just on their statistical distribution. Then, we show how the use of linguistics can actually improve AKE. We use the information conveyed by *anaphors*, which allow us to exploit the relationships between a candidate keyphrase and its surrounding context.

2 MULTILANGUAGE KEYPHRASE EXTRACTION

As explained in Section 1.2.1, supervised AKE algorithms are typically implemented using mostly features that rely on statistical and/or positional knowledge, i.e. features that use the frequency of a keyphrase in the document or in a corpus, and features which use the position(s) of the keyphrase in the input document. This consideration suggests that such algorithms do not actually learn how to find the key concepts in the documents using information from its content. This is in strong contrast with unsupervised techniques, which instead rely typically on graph-based techniques that exploit the correlations between each word in the document to identify keyphrase, as already pointed out in Section 1.2.2.

Following this path, in this Chapter we will investigate a multilingual keyphrase extraction pipeline, which successfully performs keyphrase extraction in English, Arabic, Italian, Portuguese and Romanian, even if it's trained on the first two languages only. In order to do that, we first have to collect an AKE dataset in the Arabic language, since to the best of our knowledge no quality datasets are publicly available for any of these languages but English.

The work discussed in this chapter has been presented in Helmy et al. [54] at the 2016 International Conference on Asian Language Processing (IALP 2016), held in Tainan, Taiwan in 2016 and in Basaldella et al. [16] at the 2017 edition of the conference Recent Advances in Natural Language Processing (RANLP 2017) held in Varna, Bulgaria, in 2017.

2.1 INTRODUCTION

This Chapter will investigate the possibility of building a multi language, supervised keyphrase extraction pipeline, that is able to extract keyphrases from documents written in languages different from the one(s) it's trained with.

For this work, we choose as input languages for our AKE system English, Arabic, Italian, Portuguese and Romanian. The choice of English is straightforward, as it's the most popular language in the AKE community, so there is plenty of datasets available for this language. Arabic, Italian, Portuguese and Romanian were chosen because, in order to validate our approach, we were able to work with mother-tongue speakers.

To prove our point, we decided to train the AKE pipeline in two languages, and to evaluate it on the others. To the best of our knowledge there are no publicly available AKE datasets for Arabic, Italian, Portuguese and Romanian, so we had to collect a dataset in one of these languages to carry out our experiment.

We chose Arabic as a second training language since there is a growing interest around the problem of AKE in the Arabic language. Arabic is, in fact, the fifth most spoken language in the world, with more than 240 million native speakers[1]. Nevertheless, there is no shared, standard dataset that scholars can use to assess the performance of their Arabic AKE systems. For example, [32] and [9] used custom-made corpora in their work, remarking the absence of a standard dataset.

In Section 2.2, we will describe the process we followed to collect a KP dataset for the Arabic language. In Section 2.3, in order to prove our hypothesis, we first train a ML model that can extract keyphrases in English and Arabic, we use such model to extract keyphrases in the other languages, and finally we validate the proposed solution using expert knowledge from mother-tongue speakers.

2.2 AKEC: AN ARABIC KEYPHRASE EXTRACTION CORPUS

To collect a KP dataset, as already described in Section 1.3, there are different possible solutions. An easy solution is to use author-assigned tags, as it's a common practice in academia to use them to describe a paper upon submission. However, there is no constraint that such tags should appear verbatim in the body of the paper. For this reason, they are often integrated with reader-assigned keyphrases: students or experts are asked to read the document and choose keyphrases that *should* appear in the document. However, often such readers *also* choose keyphrases that do not appear in the original document [57, 63], undermining the quality of the dataset.

For this reason, to collect our dataset we use *crowdsourcing*, an approach that has already been explored by Marujo et al. [76]. The dataset collection process will be the following: first, we collect a set of documents in the Arabic language from several domains. Then, we use crowd workers from the Crowdflower platform[2] to assign keyphrases to such documents. Finally, we analyze and merge the selected keyphrase to provide a reliable, sorted keyphrase list for each document in the corpus. We will call our dataset the Arabic Keyphrase Extraction Corpus (abbr. AKEC).

1 https://www.ethnologue.com/statistics/size
2 http://www.crowdflower.com

2.2.1 *The Crowdsourcing task*

2.2.1.1 *Document Collection*

The document collection we used contains 160 documents selected from four general purpose, freely available corpora: 46 documents from the Arabic Newspapers Corpus (ANC) [5], 53 from the Corpus of Contemporary Arabic (CCA) [4], 31 from the Essex Arabic Summaries Corpus (EASC) [33, 34], and 30 from the Open Source Arabic Corpora (OSAC) [104].

The documents are categorized into the following topics:

- Art and music (18 documents)
- Environment (18 documents)
- Finance (18 documents)
- Health (19 documents)
- Politics (18 documents)
- Religion (17 documents)
- Science and technology (18 documents)
- Sport (17 documents)
- Tourism (17 documents)

After preprocessing the documents by eliminating unrelated text like headers, image captions, and corpus metadata, their lengths vary between 500 and 1000 words, with a median of 735.5 words.

The analysis process involves three different forms of Arabic text for both documents and selected KPs. The first one is the original form, which is the text without processing or removing any character. The second one, that we will call "pure form", includes only the basic Arabic alphabet and numbers. In other words, "pure" KPs are the selected phrases with Arabic diacritic signs and non-Arabic characters removed. To obtain this result, the text is cleaned by removing all unnecessary characters like special Arabic punctuation marks, diacritics, and Kashida[3]. In addition, some Arabic characters have various forms which we normalize into a single one to decrease the processing complexity.

The third one offers a more in-depth analysis of the documents and the selected KPs, since we extracted the root form of the words, i.e. we removed any information about gender, number, pronouns, etc. attached to the word. Usually for the English language, a stemming algorithm is used to perform this task. In Arabic, however, stemming

3 Also called Tatweel, it is a form of Arabic text justification that, instead of adding whitespace, adds an horizontal, slightly curvilinear stroke between certain letters.

Word	Meaning	Lemma	Meaning
مدرستنا	Our School	مدرسة	A School
المدرسون	The Teachers	مدرس	A Teacher
الدارسون	The Students	دارس	A Student
دروس	Lessons	درس	A Lesson
دراستهم	Their Study	دراسة	A Study

Table 2.1: Examples of Arabic words and their lemmatized forms.

is much less effective than in other languages, since it simply removes the derivational affixes of the word and is often not able to get its true root (stem). Thus, many words with different meanings would be reduced to the same stem.

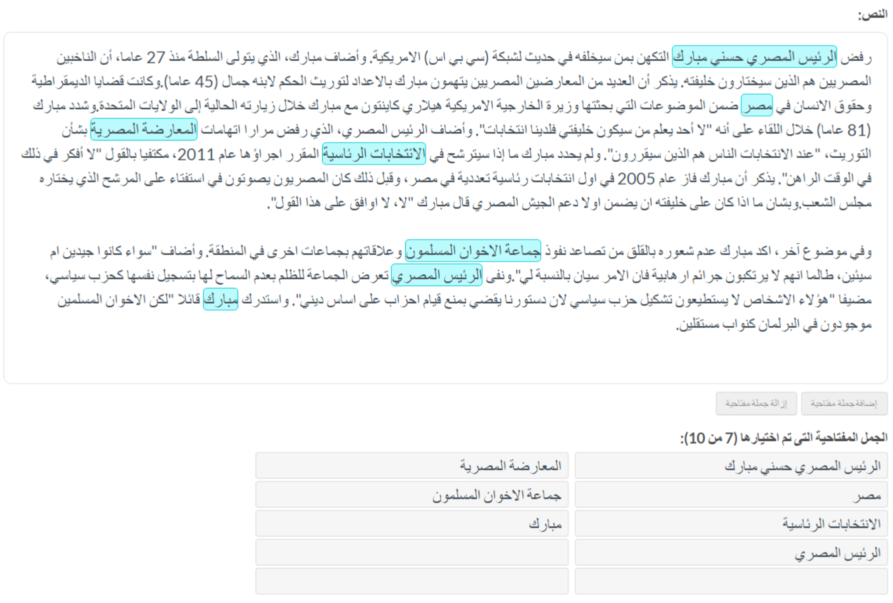
To tackle this problem, [32] used a more aggressive, custom-made stemmer. A similar approach was used by [9], which combined two different softwares to get to an acceptable root form of each word. We did not want to use custom-made solutions, so we decided to use lemmatization instead of stemming. Lemmatization, performed with the AraMorph lemmatizer [24], applies vocabulary and morphological analysis on the word to get its dictionary form (lemma), resulting in a much more precise "cut" of the originally selected words. This way we could obtain a standard root form of each selection, which allowed us ultimately to merge similar selections together. Table 2.1 provides an example of such technique for a set of Arabic words. All of these words have the same Arabic stem which is (درس, translated as "study"), but have different lemmas.

2.2.2 *Pilot Experiment*

First of all, we launched a pilot experiment on the Crowdflower platform with 10 documents, to tune our task for the whole corpus. Each worker was shown one document and had to assign 5 different KPs. Each documents was analyzed by 5 different workers that had to assign 5 different KPs to that document. This way, we had 250 total KP assignments. The workers were simply asked to select from the document phrases that they considered important inside the text without any guidance on how these phrases were composed, i.e., if the selected phrases should contain or not nouns, verbs, etc. A worker's work was discarded if the time spent on the document was lower than 90 seconds. Each worker could read and annotate up to five documents. The worker could select phrases from one to five words long.

Count	Selected	Pure	Lemmatized
1	171	169	165
2	24	25	27
3	5	5	5
4	4	4	4
Total	204	203	201

Table 2.2: Frequency of uniquely selected KPs in the pilot experiment.



النص:

رفض الرئيس المصري حسني مبارك التكهن بمن سيخلفه في حديث لشبكة (سي بي اس) الامريكية. وأضاف مبارك، الذي يتولى السلطة منذ 27 عاما، أن الناخبين المصريين هم الذين سيختارون خليفته. يذكر أن العديد من المعارضين المصريين يتهمون مبارك بالاعداد لتوريث الحكم لابنه جمال (45 عاما).وكانت قضايا الديمقراطية وحقوق الانسان في مصر ضمن الموضوعات التي بحثتها وزيرة الخارجية الامريكية هيلاري كاينتون مع مبارك خلال زيارته الحالية إلى الولايات المتحدة.وشدد مبارك (81 عاما) خلال اللقاء على أنه "لا أحد يعلم من سيكون خليفتي فلدينا انتخابات". وأضاف الرئيس المصري، الذي رفض مرارا اتهامات المعارضة المصرية بشأن التوريث، "عند الانتخابات الناس هم الذين سيقررون". ولم يحدد مبارك ما إذا سيترشح في الانتخابات الرئاسية المقرر اجراؤها عام 2011، مكتفيا بالقول "لا أفكر في ذلك في الوقت الراهن". يذكر أن مبارك فاز عام 2005 في اول انتخابات رئاسية تعددية في مصر، وقبل ذلك كان المصريون يصوتون في استفتاء على المرشح الذي يختاره مجلس الشعب.وبشان ما اذا كان على خليفته ان يضمن اولا دعم الجيش المصري قال مبارك "لا، لا اوافق على هذا القول".

وفي موضوع آخر، اكد مبارك عدم شعوره بالقلق من تصاعد نفوذ جماعة الاخوان المسلمون وعلاقاتهم بجماعات اخرى في المنطقة. وأضاف "سواء كانوا جيدين ام سيئين، طالما انهم لا يرتكبون جرائم ارهابية فان الامر سيان بالنسبة لي".ونفى الرئيس المصري تعرض الجماعة للظلم بعدم السماح لها بتسجيل نفسها كحزب سياسي، مضيفا "هؤلاء الاشخاص لا يستطيعون تشكيل حزب سياسي لان دستورنا يقضي بمنع قيام احزاب على اساس ديني". واستدرك مبارك قائلا "لكن الاخوان المسلمين موجودون في البرلمان كنواب مستقلين.

الجمل المفتاحية التى تم اختيارها (7 من 10):

الرئيس المصري حسني مبارك	المعارضة المصرية
مصر	جماعة الاخوان المسلمون
الانتخابات الرئاسية	مبارك
الرئيس المصري	

Figure 2.1: The user interface that the workers used to select keyphrases in Arabic documents.

The experiment was completed by 33 distinct workers. The average number of documents analyzed by each worker was 1.5. Table 2.2 shows the analysis of the KPs selected by the users.

The KPs were manually inspected to analyze the behavior of the workers. On a total of 204 distinct KPs, we marked 45 phrases as "wrong", i.e. we found that workers selected 28 phrases which included a verb, 12 phrases that were meaningless, and 5 phrases that likely contained a selection error, i.e. first or last character missing. After removing diacritics and lemmatizing phrases, we obtained an average of 20.1 KPs per document, which lowers to 3.6 taking only the KPs selected by at least two workers.

2.2.2.1 *Main Experiment*

Following what we learned from the pilot experiment, we launched the full experiment adjusting some parameters. First, we decided to use 10 workers per document, and to ask each worker to select 10 KPs,

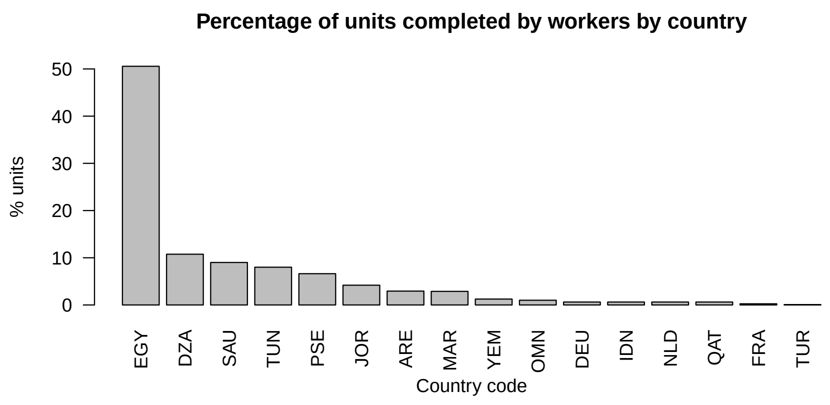


Figure 2.2: Percentage of units completed on the main experiment by country. Country names are abbreviated using ISO 3166 3-letter Codes.

while in the pilot experiment we required just 5 KPs by 5 workers. Moreover, we adjusted the task instructions to guide workers to not select phrases beginning with stopwords, verbs, or adjectives. Each unit was discarded if the worker did not spent at least 120 seconds on the document. Each worker could read and annotate up to ten documents, i.e., could complete up to ten units. Moreover, we required Crowdflower to select only medium and highest quality workers.

Figure 2.1 shows the user interface we developed to let crowd workers select the KPs in the Arabic documents. Workers were allowed to select entire words only, to prevent selection errors; wrong selections were automatically expanded to the closest full word (e.g., if a worker selected "*Crowsourci*", the selection was automatically extended to "*Crowdsourcing*"). Moreover, workers were prevented to manually insert KPs in the text fields, forcing them to select only KPs that appear in the document. Finally, workers were prevented to continue if they didn't select at least 10 KPs.

2.2.2.2 *Descriptive Statistics*

The experiment was launched and completed by a total of 226 workers, for a mean of 7.07 documents per worker. The time spent reading a document had an average of 302 seconds (5 minutes) and a median of 222 seconds (less than 4 minutes). We collected a total of 10,646 distinct KPs, i.e., 66.5 per document on average.

We performed the whole task in two sessions, and each of them required about 12 hours. The sessions were carried out respectively the 3rd and the 6th of February, 2016. The mean time spent by a worker for extracting 10 keyphrases from a document is 302 seconds (5 minutes and 2 seconds) while the median time is 222 seconds (3 minutes and 42 seconds).

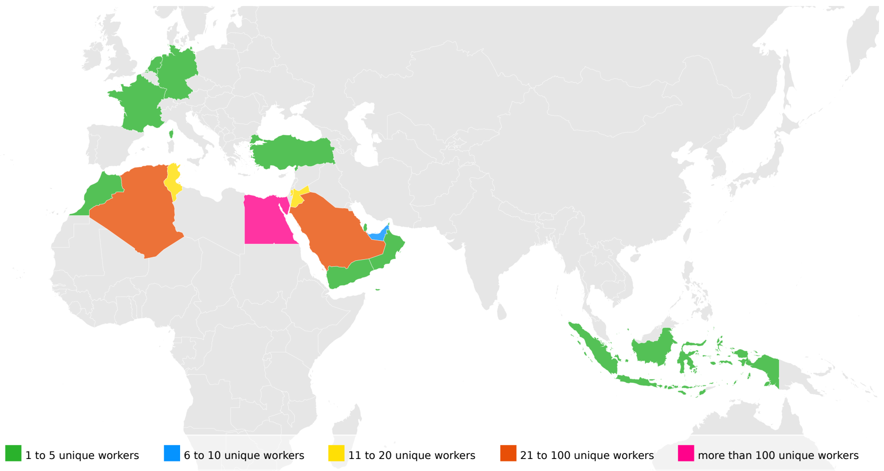

🟩 1 to 5 unique workers
🟦 6 to 10 unique workers
🟨 11 to 20 unique workers
🟧 21 to 100 unique workers
🟪 more than 100 unique workers

Figure 2.3: The geographic distribution of the crowd workers that completed the main keyphrase collection experiment.

Figures 2.2 and 2.3 show that, geographically, the crowd workers came prominently from North Africa. In fact, more than 75% of the workers were based in one of four countries, namely Egypt, Algeria, Saudi Arabia and Tunisia. Only 2.2% of the workers came from countries where Arabic is not an official language, i.e., Germany, Indonesia, Netherlands, France, and Turkey. However, since Islam is the main religion of Indonesia and Turkey, and since there are strong German, Dutch and French Muslim minorities, we assume that these workers must know Arabic as a religious language.

Figure 2.4 shows the frequency of the lemmatized phrases, when selected by at least n workers. It is noteworthy that some phrases, when lemmatized, have a frequency higher than 40, and that in at least half of the documents, more than five of the lemmatized keyphrases is selected by at least three workers, backing up our aim of building an high-quality dataset.

2.2.3 *The Arabic Keyphrase Collection*

2.2.3.1 *Selecting High Quality Keyphrases*

Taking all the data as is makes the average number of KPs per document dramatically high. For example, the SEMEVAL 2010 collection provides an average of only 14.81 KP per document, while if we took all the crowd collected phrases without any kind of filtering, we would end up with a big low-quality dataset including badly selected text, invalid KPs such as stopwords, verbs, etc.

To generate a high-quality collection, we reduced the total number of 16,000 KPs by using the approaches described in Section 2.2.1.1. Considering only pure KPs (i.e., without diacritics and forbidden symbols) leads to a total number of 10,602 different KPs; if we further

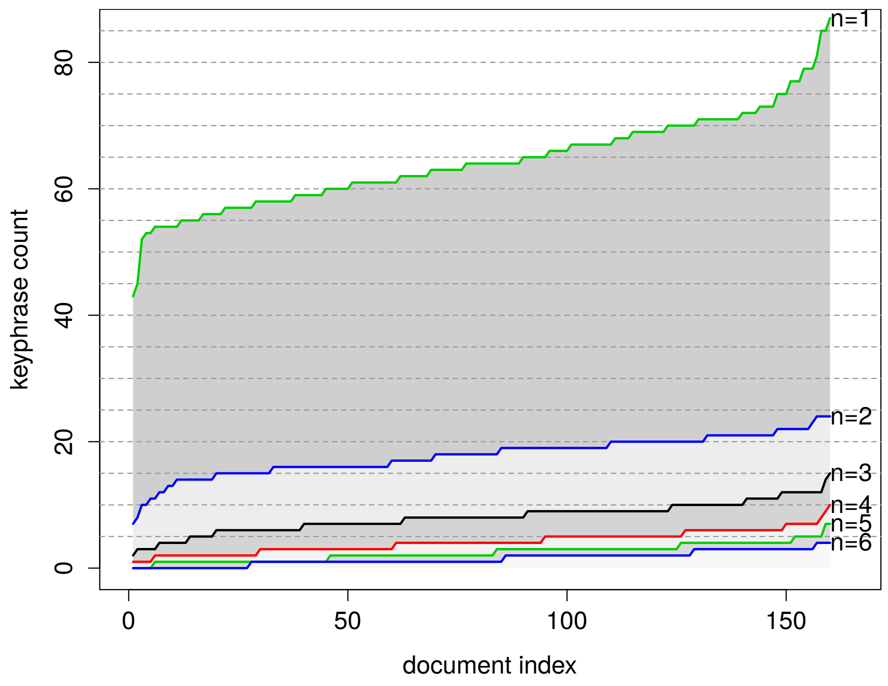


Figure 2.4: Number of *lemmatized* phrases selected by at least n out of 10 workers.

apply lemmatization we obtain 10,286 KPs, for an average of 64.2 lemmatized phrases per document. To improve the collection quality, we adopted two additional selection approaches:

- *Frequentist*: we order KPs by the number of times that they have been selected by workers, then we discard all the KPs that have not been selected at least twice.

- *Linguistic*: we build a language model and sort the KPs using that model; then, we keep the best 15 ranked phrases per document and discard the others. Note that we keep all phrases that are at tied the 15^{th} position of the raking, so the actual number of KPs per document will be variable.

Looking at Figure 2.4, we can see that almost any document is associated with at least 15 KPs selected by at least two workers and with a KP that has been selected by at least 5 workers. Table 2.3 offers a more detailed analysis of these data. We see that some documents will have very high-quality KPs, as they were selected by at least 8 workers. Table 2.3 also shows that the number of KPs selected by at least $n \geqslant 2$ workers are pretty similar to the SEMEVAL 2010 dataset.

The linguistic approach is used to discriminate phrases which are substrings of other KPs. We built a language model (LM) for each document for a total of 160 LMs. The corpus of each LM consisted in the set of the crowd-assigned KPs and its features are the n-grams (with $n = 1 \ldots 5$) generated by these KPs. For each LM, we calculated

n Worker	Median	Mean	Min	Max
2:	18	17.8	10	24
4:	4	4.1	1	10
6:	1	1.5	0	4
8:	0	0.5	0	2
10:	0	0.1	0	1
Linguistic cut:	15	15.5	15	19
SEMEVAL:	14	15.1	8	37

Table 2.3: Number of KPs which have been selected by at least n crowd workers for each document.

the sum of the columns of the relative frequency table, obtaining a score for each feature, which was dependent from how many times that feature occurred as a (or as part of a) worker selected KP. Then, we excluded the features which were not crowd-assigned KPs, and ranked them by score, obtaining our final linguistic ranking.

The difference between the two approaches is that the language model favors conceptually similar phrases. For example, suppose we have three phrases selected from a document: "computer science teacher", "science teacher" and a completely unrelated word, which is most probably an error, e.g., "foo". Assuming that the first word has been selected three times, while the second and the third one have been selected once, "science teacher" and "foo" have exactly the same score and will be discarded by the frequentist model. The language model, instead, is able to say that "science teacher" is similar to a more frequent KP, pushing it higher in the ranking.

Obviously, each model has its pros and cons; the LM may promote phrases which are actually not important, while the frequentist model may suffer from worker error. Nevertheless, we claim that they are both good ways of ranking worker selected KPs, so we release both rankings in the final dataset.

2.2.3.2 *Data Validation*

To validate our approaches, we selected a subset of 56 documents from the corpus and had an expert (an Arabic native speaker attending a Ph.D. on AKE) manually assess the quality of the KPs that the crowd selected. The expert was shown the KPs in random order to avoid any bias.

Since we have ranked KPs, it is natural to use classical "top-heavy" ranking metrics, well studied in the IR community. In particular we use the classical ones: Average Precision (AP) and Mean AP (MAP), as well as MAP@5, MAP@10, and MAP@15 to show the quality of the first ranked KPs. Indeed, we show boxplots of AP values (i.e., a representation of the distribution of AP values over the various documents — one dot, one document) to see the variability over documents.

Figure 2.5 shows that, for all documents and for both linguistic and frequentist approaches, AP@5 and AP@10 values are very high: median values are around 0.9 or higher. AP@15 values are similarly high. MAP@5, MAP@10, and MAP@15 (i.e., the mean AP values, represented as red dots in the figure), are all above 0.8. This means that the sets of the first 5, 10, or 15 KPs in both ranks (i.e., linguistic and frequentist) are very good and sometimes almost perfect. The leftmost boxplot pair in figure shows that AP and MAP values are slightly lower but still well above 0.6: although some of the KPs are not good ones according to our expert, even when using all of them we get a reasonable quality.

We are confident that, when compared with the quality of the other similar KP datasets in the literature, our dataset is at least as reliable. For example, the authors of the SEMEVAL 2010 dataset recognize that only 85% and 81% of their reader- and author-assigned KPs, actually appear in the text and, in contrast to our approach, they simply trust that their readers assigned correct KPs, without any kind of post-collection quality control process like the one we perform.

2.2.3.3 *Applying a Baseline KPE System on the Dataset*

Various KPE systems employ TF-IDF as a numerical and statistical method to extract and rank KPs to measure the importance of keywords to a document in a dataset or corpus [32, 76, 90]. Therefore, to further validate our dataset, an Arabic TF-IDF based testbed system was implemented as a baseline KPE to evaluate the quality of the dataset KPs and assess workers performance.

For each document, two lists of words have been generated. The first list contains words of all KPs extracted by the workers excluding stopwords while the second one is a sorted list of the important words generated and ranked by the testbed system. After that, the two lists were compared and the precision of the workers selections against the TF-IDF generated list was calculated. The precision was about 0.6, which means that 60% of workers KPs words are recognized by the system; we take this as a good result, especially if compared with the scores of the aforementioned KP extraction systems which rely on this feature.

2.3 CROSS-LANGUAGE SUPERVISED KEYPHRASE EXTRACTION

2.3.1 *Multilanguage Keyphrase Extraction*

Now that we have our Arabic dataset, we can proceed on building the multilanguage AKE pipeline introduced in Section 2.1.

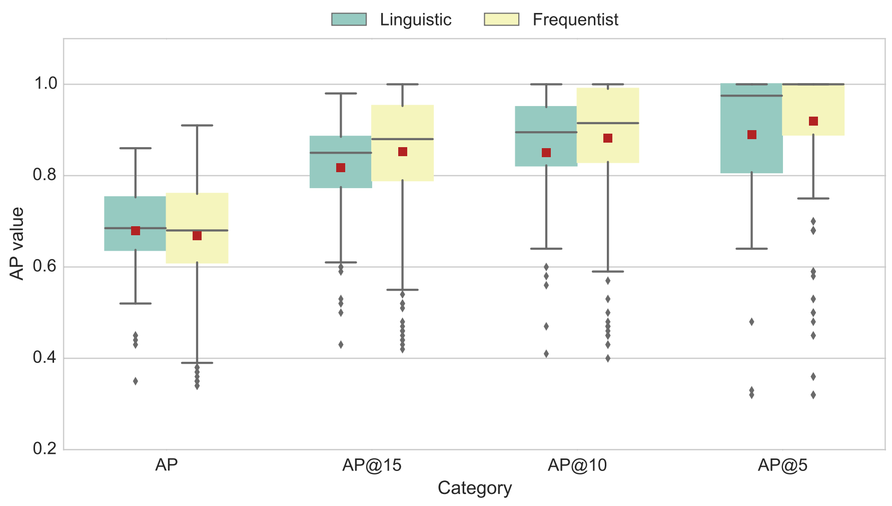


Figure 2.5: AP of the two approaches at different cuts.

To build a multilanguage AKE system the first step is to understand which parts of the pipeline need to be customized to work on more languages. Therefore, we divide the AKE pipeline in modules and we identify the language-dependent ones:

1. Low-level NLP: sentence and word segmentation, part-of-speech tagging and stemming;
2. Candidate generation: selection of the possible keyphrases in the document. It is usually performed by detecting the phrases which match certain known part-of-speech tags patterns;
3. Feature extraction: candidates are assigned some features, like position in the text, frequency, etc.
4. Candidate scoring: the feature extracted in the previous step are used to assign a score to the candidates; then, the top ranked candidates are usually used to evaluate the pipeline.

In this Chapter we focus on the first, second and fourth steps, which are the ones which rely mostly on language-specific knowledge, by implementing several AKE pipelines in the Distiller keyphrase extraction framework described in Appendix A.

2.3.2 *Low-level NLP*

The first step of the AKE pipeline consists in preparing the document to identify the potential keyphrases, by splitting the text into sentences and the sentences into tokens and, finally, performing part-of-speech tagging, stemming and/or lemmatization.

We use off-the-shelf libraries to perform these tasks. For the English and Arabic languages, we used the Stanford CoreNLP library

[75] for all the tasks except stemming, since it offers state-of-the-art performance. Due to the limited availability of languages available for CoreNLP, for the Portuguese and Italian languages we used the the Apache OpenNLP[4] library with the default models for Portuguese and Ciapetti's models for the Italian language[5]. For the Romanian language, the models were not available, so we built them ourselves; we describe this process in Section 2.3.2.1.

To stem the tokens, we used the Porter stemmer [98] for all languages but Arabic, where lemmatization was used instead of stemming, as we already explained in Section 2.2.1.1. Thus, the lemmatizer used for the Arabic language is the AraMorph lemmatizer [24].

2.3.2.1 *Romanian*

We were not able to find any suitable CoreNLP or OpenNLP models for the Romanian language to perform sentence splitting, tokenization, and PoS tagging. Thus, we decided to build our own models for Apache OpenNLP using the ROMBAC[6] dataset [59], which was collected by the Research Institute for Artificial Intelligence of the Romanian Academy.

The corpus is divided in five domains: journalism (news and editorials), pharmaceutics and medicine, law, biographies of Romanian literary personalities, and fiction. The documents are tokenized, morpho-syntactically tagged, lemmatized, shallow parsed (chunked) and encoded in an XCES compliant format. The tagset used in the ROMBAC corpus is quite large, with about 614 MSD tags [35]. The corpus contains about 41,000,000 words, including punctuation. We tested different training/testing split to obtain the best possible performance. For the training of the sentence detector and tokenizer, we used the journalism domain, while to train the POS tagger we used the journalism, medicine, and fiction domains.

We tested the models on the remaining genres. The sentence detector obtained a F1-score of 72.0, the tokenizer a F1-score of 98.61 and the PoS tagger obtained a F1-Score 94.41.

While the performance for the tokenizer and the PoS tagger is satisfactory, the sentence detector model we trained offers a somewhat underwhelming performance. We tried to obtain better performance by tweaking the parameters offered by the OpenNLP library or different training/testing split but, after careful analysis, we concluded that the problem lies in the input data. The ROMBAC dataset, in fact, is constituted by files which contain all the original page formatting data, such as title, chapter name, author, page number, and so on, so the library is not able to fit a better sentence splitting model.

4 https://opennlp.apache.org/
5 https://github.com/aciapetti/opennlp-italian-models
6 The Romanian Balanced Annotated Corpus

Anyway, we evaluated the model on the set of documents we used to perform AKE in Romanian (see Section 2.3.4). After a manual inspection of the results we decided to continue to use this model, since for the scope of our experiments the performance was good enough.

2.3.2.2 *Arabic*

Being very different from the Western languages we previously described [36, 46], the Arabic language documents needed additional text preprocessing steps before we were able to extract candidate keyphrases successfully.

To make our pipeline able to work on arbitrary documents, and still be compatible with the AKEC dataset we introduced in Section 2.2, we had to apply the same preprocessing step we used in the corpus. Thus, in order to align the documents with the "pure text" representation we introduced in Section 2.2.1.1 we use the same preprocessing techniques, i.e. we remove again all unnecessary characters like the special Arabic punctuation marks, diacritics, and Kashida, and we normalize Arabic characters.

Another issue about Arabic is that punctuation is used differently than in English and other Western languages. In fact, Arabic has traditionally no punctuation, and it is still usual to find modern Arabic books written in this way [31]. In the other languages we analyzed, we trivially assume that all the words of a keyphrase have to appear within the same sentence. Since it may be not possible to distinguish sentences in an Arabic input text, we follow the approach described by Helmy et al. [53], assuming that the tokens of a keyphrase in the Arabic language should appear in the same syntactic noun phrase.

2.3.3 *Candidate Generation*

Candidate generation requires more domain knowledge than simply using an off-the-shelf library. To generate the candidate keyphrases, we scan the text for phrases that match certain part-of-speech tag sequences (we will call such sequences *PoS patterns* from now on). For example, for the present Chapter, a valid keyphrase may be "multilingual keyphrase extraction", which PoS pattern is "`(adjective, noun, noun)`".

These patterns, as introduced in Section 1.2, are the most common way of generating the candidates in supervised AKE and they require to be engineered by a domain expert, since they are significantly different from language to language. For example, the English phrase "*software engineering*" is translated in Italian as "*ingegneria del software*", where *del* is an *articulated preposition*, i.e. the union of an article with a preposition, a part of speech which does not exist in the English language. This example shows also that in some languages we will look for shorter n-grams, while other, more verbose languages may require a larger n. For example, in English we look typically up to 3-grams [63, 99], while in Italian our system will look for 1 to 5-grams.

For the English and Arabic languages we used known PoS patterns from the literature [53, 62]. For the other languages, we constructed or own PoS patterns. For this purpose, we collected about 500 author assigned keyphrases in Italian, Portuguese and Romanian from scientific repositories and we performed PoS tagging on them. Then, we selected the most frequent patterns, after manually removing or correcting the erroneous ones (e.g. pattern with only conjunctions, etc.).

2.3.3.1 *Features Extraction*

After the identification of the candidate keyphrases we assign to each of them seven features. Most of the features assigned to candidate keyphrases rely simply on statistical and structural information (like the position in the text, frequency, TF-IDF, and so on) [52]. As already pointed out in Section 1.2.1, researchers have developed language-dependent features, based e.g. on part of speech tags [57], or based on external knowledge like Wikipedia [77], but this kind of features requires long computational times and specialized tools which may not be available for all languages. Thus, for the sake of simplicity, we decided to leave them out of our pipeline.

To describe the features we implemented we use the notation we defined in Section 1.5. In our system, we take four features from Pudota et al. [99] and we use them for *unsupervised* keyphrase extraction only. These features are:

NORMALIZED FREQUENCY i.e. the number of times a candidate appears in the document normalized to the number of sentences, i.e. given a candidate kp for a document d,

$$norm\_frequency(kp, d) = \frac{|\{i | kp \in s_{i,d}\}|}{|S_d|}$$

HEIGHT i.e. the relative position of the first occurrence of the candidate in the document, i.e. given a candidate kp for a document d,

$$height(kp, d) = \frac{min_i\{i | kp \in s_{i,d}\}}{|S_d|}$$

DEPTH i.e. the relative position of the last occurrence of the candidate in the document, i.e. given a candidate kp for a document d,

$$depth(kp, d) = \frac{max_i\{i | kp \in s_{i,d}\}}{|S_d|}$$

LIFESPAN i.e. the difference between the last and the first appearance of the candidate, i.e. given a candidate kp for a document d,

$$lifespan(kp, d) = depth(kp, d) - height(kp, d)$$

We add three other features to this set to perform supervised keyphrase extraction, namely:

FREQUENCY i.e. the number of times a keyphrase $\mathtt{kp}$ appears in a document $\mathtt{d}$, i.e. the function $\mathtt{freq}(\mathtt{kp}, \mathtt{d})$ defined in Section 1.5; While this feature may seem redundant since we already have the *normalized frequency* in our model , our experiments showed that using both features leads to better results. To see why, just consider the word "candidate" in the present document: it has been repeated many times in the same sentence, e.g. in the definition of the *lifespan* feature;

TF-IDF i.e. one of the first features used for AKE along with *height* [121]. A common statistic used in information retrieval to identify which candidates are peculiar of that particular document with respect to a corpus, is the product of the *term frequency* $\mathtt{tf}$ and the *inverse document frequency* $\mathtt{idf}$. This feature are some of the most popular ones in supervised AKE, so it's straightforward to include them in the model (see Section 1.2.1) detail, we define $\mathtt{tf}$ and $\mathtt{idf}$ as

$$\mathtt{tf}(\mathtt{kp}, \mathtt{d}) = \frac{\mathtt{freq}(\mathtt{kp}, \mathtt{d})}{|\{w|w \in \mathtt{d}|\}}$$

$$\mathrm{idf}(\mathtt{kp}, D) = \log \frac{|D|}{|\{\mathtt{d} \in D : \mathtt{kp} \in \mathtt{d}\}|}$$

so have that

$$\mathtt{tfidf}(\mathtt{kp}, \mathtt{d}, D) = \mathtt{tf}(\mathtt{kp}, \mathtt{d}) \times \mathtt{idf}(\mathtt{kp}, D)$$

DPM i.e. Document Phrase Maximality, is used to discriminate between overlapping keyphrases and it helped to reach new state-of-the-art performance in the AKE task on the SEMEVAL 2010 dataset [47]. Given a document $\mathtt{d}$ and the candidate keyphrase $\mathtt{kp}$, we define the set $\mathtt{sup}(\mathtt{kp}, \mathtt{d})$ of the *superterms* of $\mathtt{kp}$ candidate keyphrase, i.e. the set of the candidates that contain $\mathtt{kp}$ as a substring. For example, if we have a document $\mathtt{d}$ which talks about "*software engineering*", we have that "*software engineering*" $\in \mathtt{sup}($"*software*"$, \mathtt{d})$. Then, DPM is defined as

$$\mathrm{DPM}(\mathtt{kp}, \mathtt{d}) = 1 - max_{s \in \mathtt{sup}(\mathtt{kp}, \mathtt{d})} \frac{\mathtt{freq}(s, \mathtt{d})}{\mathtt{freq}(\mathtt{kp}, \mathtt{d})}$$

2.3.3.2 *Candidate Scoring*

To score the candidates, the most trivial approach in this kind of AKE pipeline is to assign heuristically crafted weights to the features, like in Pudota et al. [99]. Typically, however, one can train a machine learning algorithm over a dataset, like the SEMEVAL 2010 dataset [63], and

Feature Name	**Weight**
Frequency	0.10
Height	0.32
Depth	0.16
Lifespan	0.12

Table 2.4: Values for the weights of the features used in Section 2.3.3.3

use the generated model to identify keyphrases of new documents. Unfortunately, as pointed out in the introduction, datasets are available only for a minority of languages, so this is not always a viable option.

In our system we used three different scoring techniques. First, since we do not have training sets for all five languages, we assigned manual weights to four of our features, using values which have proven to work on the English language [99]. Then, we trained two models, one for English using the SEMEVAL 2010 dataset and one for Arabic using the AKEC dataset. Finally, we used the models trained on these languages to score keyphrases in Italian, Romanian and Portuguese as well.

2.3.3.3 *Manual Weights*

In this approach we follow a very simple technique. Given a candidate keyphrase kp, a feature f, and the set of the features F, we define $value(kp, f)$ as a function which returns the value of f for the candidate kp. For each feature, we also assign a weight w, whose value is defined below. Then, the score of a keyphrase kp is computed as follows:

$$score(kp) = \sum_{i=1}^{|F|} w_i \times value(kp, f_i)$$

As mentioned in Section 2.3.3.1, the features used in this step are *normalized frequency*, *height*, *depth*, and *lifespan*, and the values of their weights w_i are displayed in Table 2.4, as presented in Pudota et al. [99]. There, the authors used a set of 1000 keyphrases assigned to 215 documents from [90] to calculate them[7]. We did not re-compute the values of the weights by ourselves, because authors already proved its effectiveness, and since we just want to use them as a baseline.

7 Here we call Pudota et al. [99]'s *phrase last occurrence* and *depth* as *depth* and *height* respectively, but their values are computed in the same way. We do not use the *noun value* feature because it has proven ineffective in our experiments.

2.3.3.4 *Supervised Weights*

The manual weights technique is trivially limited by the fact that these weights are used to compute a simple linear function, so it can't cope with complex interactions of the features. Many machine learning algorithms such as multilayer neural networks, support vector machines, decision trees, and so on, are instead able to learn non-linear functions, and for this reason are commonly used for AKE in literature [52].

Thus, in our final step we train a multilayer neural network to extract keyphrases in English and Arabic, using two different training sets. For the English language, we use the dataset from the "SEMEVAL-2010 Task 5: Automatic Keyphrase Extraction from Scientific Articles" challenge [63] (herein, simply SEMEVAL 2010). For the Arabic language, the network has been trained on the AKEC corpus described in the previous Section. The corpora have 244 and 160 documents respectively, of which 144 are used for training and 100 for testing in the SEMEVAL 2010 dataset, and 100 are used for training and 60 for testing in the AKEC dataset (see Table 2.5) . Both datasets, as described in the previous Sections, have about 15 human-assigned keyphrases per document (see Table 2.3). Moreover, as described in Table 2.5, while the SEMEVAL dataset is composed by scientific papers, hence the mean length of its documents is of about 8000 words, the AKEC dataset is composed by much shorter documents from a variety of sources, from religion to news, with a mean length of about 750 words. For this reason, we expect that the two networks will have quite different weights, so in Section 2.3.5 we will use Garson's algorithm [41] to analyze the importance of each feature in both datasets.

The neural network has been trained using the `nnet` package in the R programming language using the *entropy* parameter. The network uses one neuron per input feature, a hidden layer with two times the input neurons, and one output neuron. The keyphrases are ranked according to the score assigned by the network, which is the value of the output neuron.

We use the models trained on these datasets to extract keyphrases in all our five languages. This is possible because, as already pointed out, the neural network ignores the text of the actual candidate keyphrase, since it does not receive any information about its words or about its meaning, but only statistical information about its appearance(s) in the input document.

2.3.4 *Experimental Evaluation*

While it is straightforward to analyze the performance of our model in English and Arabic since both the SEMEVAL 2010 and AKEC datasets provide test sets, for Italian, Portuguese and Romanian we

Dataset	Size	Mean length	σ length
Semeval 2010	244	8020	1946
AKEC	160	757	145
English	20	3717	1877
Italian	20	4699	3412
Portuguese	20	4335	2482
Romanian	20	802	730

Table 2.5: The datasets used to train and test the pipelines, with their number of documents, their mean length in words, and the standard deviation σ of the length in words.

are not aware of publicly available collections of keyphrase extraction datasets.

For this reason, we asked mother tongue speakers of these three languages to collect 20 documents per language and assign 15 keyphrases to each document, ranking them by importance. To have a further verification of our techniques, we repeated the same process for 20 documents in the English language as well. The collected datasets are described in Table 2.5, in which we show the mean number of words and the standard deviation of the number of words of the documents of each dataset. For English, Italian and Portuguese, we have collected similar datasets with a majority of scientific documents, which is reflected by mean of about 4000 words per document. For Romanian, we collected mainly newswire documents, so we have a mean of 800 words per document, close to the AKEC dataset. All the purpose-built datasets have a greater variability in the number of words with respect to the SEMEVAL 2010 and AKEC datasets, because while these datasets are composed by only one kind of documents with strict constraints on the length, our test datasets are mixed, containing scientific papers, newswire text, web pages, etc.

Note that for our own datasets, since they have only 15 expert-assigned keyphrases, the Precision score and the F1 score on the top 15 candidates are equal[8], so for these datasets we show only the former.

For all three approaches, we evaluate our algorithms using Precision computed on the top 5 extracted keyphrases (herein Precision@5 or P@5), and Precision and F1-score computed on the top 15 extracted keyphrases (herein P@15 and F1@15), which are the same metrics used to evaluate the SEMEVAL 2010 dataset [63]. Since these metrics do not take in account the order of the extracted keyphrases, we add Mean Average Precision (MAP) to this set, as anticipated in Section 1.6. Note that for our own datasets, since they have only 15

8 Because the size of the set of the retrieved documents is equal to the size of the set of the relevant documents, hence $\mathrm{Precision} = \mathrm{Recall}$.

Dataset	Pipeline	P@5	P@15	MAP	F1@15
English (SEMEVAL)	Manual weights	0.13	0.1	0.076	0.10
	English model	**0.29**	**0.21**	**0.151**	**0.21**
	Arabic model	0.25	0.18	0.117	0.18
Arabic	Manual weights	0.46	0.37	0.158	0.145
	English model	**0.63**	0.47	0.185	0.18
	Arabic model	0.61	**0.48**	**0.190**	**0.19**
English	Manual weights	0.33	0.26	0.232	
	English model	**0.51**	**0.35**	**0.320**	
	Arabic model	0.47	0.32	0.280	
Italian	Manual weights	0.37	0.23	0.182	
	English model	**0.44**	**0.26**	**0.209**	
	Arabic model	0.41	0.23	0.185	
Portuguese	Manual weights	0.40	0.27	0.226	
	English model	0.42	0.24	0.221	
	Arabic model	**0.45**	**0.31**	**0.250**	
Romanian	Manual weights	0.34	0.26	0.229	
	English model	**0.47**	**0.3**	**0.266**	
	Arabic model	0.39	0.28	0.248	

Table 2.6: The results obtained with the different 15 experiment we executed.

expert-assigned keyphrases, the Precision score and the F1 score on the top 15 candidates are equal[9], so for these datasets we show only the former.

2.3.5 *Experimental Results*

We present the results obtained in our experiments in Table 2.6. As expected, the manual weights method achieves the lowest performance. This is true in particular for the SEMEVAL 2010 dataset, where its performance by F1-Score would have been only the second-to-last in the original event. On the other datasets, though, the performance is higher, with 40% and 46% P@5 score on the Portuguese language and Arabic language respectively. These scores seem to be particularly good, since the best performing system in the SEMEVAL 2010 challenge obtained 40% in P@5.

Using the English model we obtained better performance on the SEMEVAL 2010 dataset with 21% F-Score, which would be enough

9 Because the size of the set of the retrieved documents is equal to the size of the set of the relevant documents, hence $\mathtt{Precision} = \mathtt{Recall}$, hence, since the F1-Score is the harmonic mean of Precision and Recall, we have $\mathtt{Precision} = \mathtt{Recall} = \mathtt{F1\text{-}Score}$.

to be placed 9th over 19 systems in the challenge. Since we were not interested in getting the best score but we want to get an average AKE system, this result looks acceptable. Moreover, the score obtained by our neural network greatly outperforms the manual baseline, so we were satisfied and we decided to use it on the other languages.

The Arabic model obtains similarly satisfactory results on the AKEC dataset. We have no other systems to compare our model with, since the dataset has been recently released, but the 61% P@5 Score is hugely outperforming the best system in the SEMEVAL 2010 challenge and a 15% improvement over the manual weights, so we're satisfied with this result. As shown in Table 2.6, Precision@15, MAP, and the F1-Score improve as well, so we were satisfied with the performance of this model.

After we validated the machine learning models, we proceeded to use them for the other languages. Using the English and Arabic models to extract keyphrases in Italian, Portuguese, and Romanian offers *always* an improvement with respect to the manual weights, and the same holds for our English countercheck collection.

Analyzing the results for each language, we see that the English model outperforms the Arabic one on the English, Italian and Romanian language, while the Arabic model performs better on the Portuguese language only. However, both models obtain good performance, with at least around 40% Precision on the top 5 keyphrases which, as we previously mentioned, would be a top score on the SEMEVAL 2010 challenge.

Looking at the Precision@15 score, of particular interest is the English model on the English and Romanian collections, and the Arabic model again on the English collection and on the Portuguese one, all of which reach and/or surpass 30%, outperforming again the best performing systems on the SEMEVAL 2010 dataset.

Anyway, a direct comparison of our results with the one obtained in the SEMEVAL challenges is clearly not fair, since the documents in our collections (other than written in different languages!) are significantly shorter, so we generated much less candidate keyphrases and the problem we had to solve was easier [52]. This is the reason why our English model performs better on our collections than on the SEMEVAL 2010 dataset, the reason why the Arabic model performs better on the AKEC collection than on the other datasets, and a probable reason of the poor performance of the unsupervised weights, since Pudota et al. [99] tailored them on documents with a different length (i.e. the already mentioned Inspec corpus [57]).

This difference is reflected in the weights of the two neural networks, as we see in the results of Garson's algorithm for the computation of the relative importance of the weights in a network [41] plotted in Figure 2.6. On the one hand, on the AKEC dataset, where the documents are short, presumably there are less repetitions and

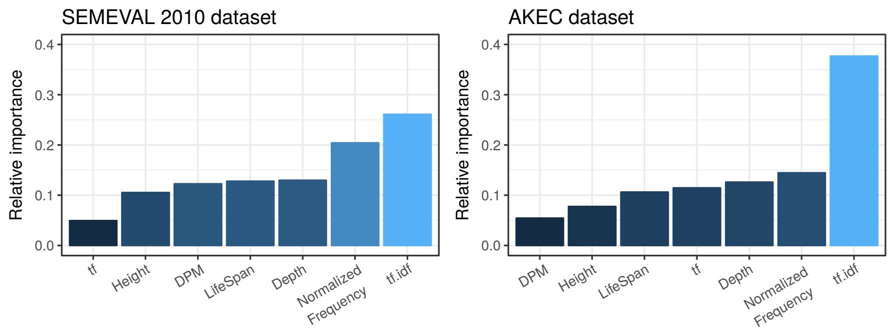


Figure 2.6: The result of Garson's weight analysis algorithm [41] on the models trained on the SEMEVAL 2010 and AKEC datasets.

terms are sparsely positioned in the document, so the most important feature used by the network is TF-IDF. On the other hand, for the SEMEVAL 2010 dataset, documents are much longer, there are more repetitions, and the documents are more structured, having an abstract, a body, and the conclusion, so important terms are expected to appear in a certain position of the document. The network seems to take into account these factors, since while TF-IDF is still the most important feature for the SEMEVAL 2010 dataset according to Garson's algorithm, normalized frequency is the second most important feature with a close gap, followed equally by the features which take into account the position of the candidate in the document.

2.4 CONCLUSIONS

In order to build the multilanguage AKE system, we first had to complete our first effort in building a new KP dataset for Arabic documents by means of crowdsourcing. Being our first take in building such a corpus, there is plenty of directions to explore in the future. It is possible that we will enlarge the corpus by including more documents; before doing so, however, we intend to study in more detail some issues. For example, we intend to try different approaches and variants to filter the high quality KPs besides those presented in Section 2.2.3.1. It will also be important to understand which is the ideal number of workers per document; we have used 10 in our experiment, and a first research direction may be to see if some sampling technique can lead to accurate KPs with lower numbers and, thus, lower cost.

Finally, on a related note, we also plan to try different experimental designs. For instance it would be interesting to try an approach similar to the well known ESP game [2], including the mechanism of taboo words to avoid the crowd to repeatedly select already known KPs.

The dataset is available at `https://github.com/ailab-uniud/akec`, and as anticipated is structured as follows: 100 randomly selected documents to be used as the training set, and 60 documents as test set. For both sets, for each document, we provide a list of all KPs selected by the workers, randomly ordered, and the two lists of good quality KPs generated with the approaches discussed in Section 2.2.3.1. This split is the one we used in the present Chapter to train our multilanguage pipeline.

Our approach showed that it is possible to build an effective supervised keyphrase extraction pipeline for an under-resourced language which lacks a keyphrase extraction gold standard by training it on another language. In fact, by training our AKE pipeline over English and Arabic, we were able to obtain good performance on Italian, Romanian and Portuguese as well. However, the English and Arabic pipeline still obtained better performance on their respective language corpora. Moreover, the actual performance of our AKE pipelines seems to depend more on the length and the domain of the document than on its language, since in general the accuracy of the extracted keyphrases seems to increase with the decrease of the length of our test document, confirming what Hasan and Ng [52] pointed out in their survey.

As a future work, to validate this hypothesis, it should be considered to perform further experiments on documents in several languages coming from several domains and of different lengths, to verify if a supervised AKE algorithm performs better if trained on the same language or if it is trained on the same domain.

3 IMPROVING SUPERVISED KEYPHRASE EXTRACTION WITH LINGUISTICS

After the investigation of how training and testing an AKE algorithm on different languages, in this Chapter our focus will be adding more contextual information using *linguistics* in AKE instead. As already discussed in the previous Chapters, most of the supervised algorithms used in AKE do not make use of linguistic information to train the models. The technique presented in this Chapter tries to invert this trend, by making use of the information conveyed by anaphoras.

The work discussed in this chapter has been presented in Basaldella, Chiaradia, and Tasso [11] at the 26th International Conference on Computational Linguistics held in Osaka, Japan in 2016.

3.1 INTRODUCTION

In this chapter we follow the path of exploring new features and new ways of using linguistic knowledge from anaphora resolution in order to detect more contextual information and to improve AKE performance. We started from the following hypotheses:

- If an n-gram is referenced many times inside a document, i.e. it has many *anaphors*, its relevance as a KP may increase;
- If a pronoun can be replaced with the noun (or noun phrase) that it substitutes, we may detect information about said noun that otherwise would be lost; this information could be used to detect better KPs.

To check if these hypotheses hold we used the following approach. First, we set a baseline to compare our hypotheses against by choosing a minimal set of features that defined a system behaving like an average SEMEVAL 2010 contestant, using a pipeline similar to the one used in the previous Chapter, and implemented it in the Distiller system described in Appendix A. Then, we designed two approaches, one based on the new linguistic features and the other based on a text preprocessing stage which applies anaphora-antecedent substitutions. Finally, we evaluated the performance of several ML algorithms using the SEMEVAL 2010 dataset and different feature sets combination which include the first hypothesis, the second one, or both.

3.2 ANAPHORA RESOLUTION

Anaphora resolution (hence AR) is the problem of resolving what a pronoun or a noun phrase refers to. For example, in the phrase "*His tail between his legs, the dog walked out the door*": the goal of an AR algorithm is to understand that the pronoun *his* refers to *the dog*.

AR is a problem with a long history in the NLP community. For example, to solve this problem, Lappin and Leass [66] proposed "an algorithm for identifying both intrasentential and intersentential antecedents of pronouns in text": they used syntactical and morphological filters to find out dependencies between pronouns and possible related noun phrases (herein NPs), scoring the candidates by considering salience to select an adequate antecedent for each pronoun not flagged as pleonastic.[1]

A close problem to anaphora resolution is *coreference resolution*. These two fields share similar information, so they overlap in a certain way: both problems, in fact, have the ultimate goal of detecting the "cohesive agency" [49] which points back to some previous item"[2].

We can define the process of binding an antecedent to an anaphora as anaphora resolution; coreference resolution instead is defined the process of detecting when anaphors and their antecedent(s) have in common the same referent in the real world. Consider this the example by Mitkov [84]:

> <u>This book</u> is about anaphora resolution. <u>The book</u> is designed to help beginners in the field and <u>its</u> author hopes that <u>it</u> will be useful.

Then the NP "the book" and both the pronouns "its" and "it" are anaphors referring to the antecedent "This book", and all three anaphors have the same real-word referent, which is "this book". So anaphors and their antecedent(s) are coreferential and form a chain, called *coreference chain*, in which all the anaphors are linked to their antecedent.

In our experiments we used the Stanford Coreference Resolution System module `dcoref` in the Stanford CoreNLP library [75] to retrieve anaphors and referents to implement our linguistics based features. We choose this software because of its good performance, since it is the direct descendant of the top ranked system at the CoNLL-2011 shared task. Moreover, both the Distiller and Stanford's system are Java-based, thus the integration of the two systems is easier. To resolve both anaphora and coreference, `dcoref` extracts the couples of anaphors and their relative referents, according to the matching of phrases' attributes, such as gender and number.

Other strategies to perform anaphora resolution, as the one introduced by Ge, Hale, and Charniak [42], use statistical information to

1 A pleonastic pronoun is tipically used in phrasal verbs as subject or object but without an effective meaning (i.e. it seems, it is known, etc.)

2 Cohesion will be discussed in detail in Section 3.3.1

resolve pronouns. They use the distance between pronoun and the proposed antecedent to check the probability of the connection between them, information about gender, number, and animacity of the proposed antecedent as hints for the proposed referent, and head information to make selectional restrictions and mention count.

3.3 ANAPHORA IN KEYPHRASE EXTRACTION

3.3.1 *Motivation*

To understand why we hope that AR can be helpful in detecting good keyphrases, it may be useful to understand the roots of *human communication*. In fact, when we (humans) communicate, whether in a spoken or written form, generally we express a coherent whole, i.e., a consistent and logical collection of words, phrases, and sentences. In many languages, including English, people often use abbreviated or alternative linguistic forms, such as pronouns, referring to or replacing some items that were previously mentioned in the discourse. Thus, to fully understand the discourse we need to interpret these elements, which *depend* on the elements they refer to. In linguistics, this process of interpretation is called *cohesion* [84].

On the other hand, when a document is processed for AKE, non influential words are usually removed. These words, commonly called *stop words*, are excluded from the text because they appear to be not significant, even if they are extremely frequent. Among them there are also pronouns such as *he*, *she*, *it*, *that*, *who*, and so on. Moreover, using the PoS pattern method we described in Section 2.3.3 to generate candidate keyphrase we automatically discard pronouns, since they are not included in the noun-phrase like patterns typically used for AKE [62].

The removal of such elements causes a loss of cohesion, both syntactically and semantically. This happens because pronouns have a relevant role in the discourse, since they allow the author to enrich his writing using a fuller vocabulary, composing more complex sentences, and so on. In fact, pronouns are parts of the text which typically have the function of a substitute: depending on the case, they can replace a subject or an object, they can indicate possession, places, or refer back to previously mentioned things or people. Given these premises, disregarding all pronouns without replacing them with a valuable substitute could lead to a loss of syntactical and/or semantical information. In fact, during the reading process we are able to decode the information conveyed by pronouns because we automatically replace them with the entity they refer to. In NLP this process is performed by anaphora resolution algorithms, thus our idea is to use this information, which would be otherwise lost, for AKE.

3.3.2 *Definitions*

We augment the terminology we introduced in Section 1.5 with the following definitions Given a document d and a keyphrase kp, $S_d(kp)$ is the set of sentences $s \in d$ in which kp appears, i.e.

$$S_d(kp) = \{s \in d | d \ \in D, kp \in s\}$$

We also define $|C_d|$ as the number of clauses in the document d, which are defined as a "simple sentences". In details, clauses are defined as the smallest grammatical units which can express a complete proposition[3], while sentences can be composed by more than one clause.

Finally, we remind that given a sentence S we denote with $|S|$ the number of words in S, with $|kp|$ the number of tokens in kp, with $|S_d(kp)|$ the number of sentences in the set S, and so on.

3.3.3 *First approach: Use of anaphora as a feature*

In our first approach we decided to use linguistic knowledge provided by anaphors to produce some new features. In detail, we designed a statistical feature that counts all the pronouns/pronominal anaphors which point to an entity (the *antecedent*), and a feature based on lexical noun phrase anaphors, which are realized as definite noun phrases and proper names [84]. We will call them *nominal anaphors* and *proper name anaphors* respectively.

For the first feature we follow this process: first, we use the Stanford CoreNLP Coreference Resolution System to find all the anaphors contained in a text and link them to their antecedent. Then, we select the pronominal anaphors, which are anaphors identified by personal pronouns *(he, she, ...)*, reflexive pronouns *(him, her, ...)*, possessive pronouns *(himself, itself, ...)*, demonstrative pronouns *(that, those, ...)*, and relative pronouns *(which, who, ...)*. Finally, we normalize the references counted for each antecedent dividing them by the number of clauses in the document.

Formally, we call $PA_{d,kp}$ the set of pronominal anaphors for which kp is the antecedent in the document d. We define:

$$\mathtt{numOfReference}(d, kp) = \frac{|PA_{d,kp}|}{|C_d|}$$

In our opinion, the use of sentences for normalization is not correct because within a sentence we could find more than one pronoun, thus skewing the normalization. If we choose the number of clauses to normalize the feature we are instead sure that $0 \leqslant \mathtt{numOfReference} \leqslant 1$. For clarity, consider the "this book" example from [84] we introduced

3 Here we use *clause* and *proposition* as defined in [65].

in Section 3.2: by normalizing over sentences, the value of the feature would be $\frac{2}{1} = 2$, while by normalizing over clauses the value of the feature is $\frac{2}{5} = 0.4$.

The other linguistic feature we implemented is based on *nominal* and *proper name* anaphors. A nominal anaphora instead arises when the referring expression has a non-pronominal noun phrase as its antecedent: it is the case of clauses in which anaphora and antecedent are implicitly related, i.e., they do not stand in a structural or grammatical relationship, but they are linked by a strong semantic one. Consider this example from Wikipedia[4]:

> Margaret Heafield Hamilton (born August 17, 1936) is a computer scientist, systems engineer and business owner. She was Director of the Software Engineering Division of the MIT Instrumentation Laboratory, which developed on-board flight software for the Apollo space program. In 1986, she became the founder and CEO of Hamilton Technologies, Inc. in Cambridge, Massachusetts. The company was developed around the Universal Systems Language based on her paradigm of Development Before the Fact (DBTF) for systems and software design.

Here "Margaret H. Hamilton" is the *antecedent* and the corresponding anaphors are the underlined words in the quote. "Computer scientist", "Director of the Software Engineering Division" are all examples of *nominal anaphors*.

The Wikipedia excerpt continues with this sentence:

> Hamilton has published over 130 papers, proceedings, and reports about the 60 projects and six major programs in which she has been involved.

Here, "Hamilton" is a proper name referring to "Margaret Heafield Hamilton", and realizes a *proper name anaphora*.

When we read these anaphoric patterns we automatically link, for example, the concept of being a "computer scientist" to a property of the subject of this sentence, i.e. Mrs. Hamilton, while in AKE this information is lost. Hence, the basic idea behind this feature is to reward all the candidate KPs which appear in a nominal or proper name anaphora because they implicitly refer to the mentioned subject, highlighting important aspects of it.

In details, we process the document as previously defined. Then, for each candidate KP, we count all the times it appears in the set of the lexical noun phrases, i.e., the set of nominal and proper name anaphors. Finally we normalize the obtained score by the total number of appearances of the candidate in the document.

---

4 https://en.wikipedia.org/wiki/Margaret_Hamilton_(scientist)

Formally, given a keyphrase kp and the set NPA_d of the lexical noun phrase anaphors in the document d, the *inAnaphora* feature can be computed as follows:

$$\mathrm{inAnaphora}(d, kp) = \frac{|\{a \in NPA_d | kp = a\}|}{|S_d(kp)|}$$

3.3.4 *Second approach: Use of anaphora for preprocessing*

While the previous approaches are able to capture some information about anaphora, they are not powerful enough to catch other knowledge that anaphora convey. For example, frequency-based features such as TF-IDF, which plays an important role in several AKE algorithms since its first introduction in [39], may be recalculated using the anaphors in the frequency count as well.

This leads us to a different strategy: transforming the text into something that resembles the original human reading process as described in Section 3.2. To achieve this goal, we add to the our system a pre-processing stage that receives the original text from the dataset and substitutes in it all the non pleonastic pronouns with their antecedent. After this preprocessing phase, we perform AKE as usual and then we evaluate the results.

Consider the example we introduced in Section 3.3.3. If we apply the pre-processing, the sentence becomes:

> Margaret Heafield Hamilton (born August 17, 1936) is a computer scientist, systems engineer and business owner. Margaret Heafield Hamilton was Director of the Software Engineering Division of the MIT Instrumentation Laboratory, MIT Instrumentation Laboratory developed on-board flight software for the Apollo space program. In 1986, Margaret Heafield Hamilton became the founder and CEO of Hamilton Technologies, Inc. in Cambridge, Massachusetts. The company was developed around the Universal Systems Language based on Margaret Heafield Hamilton paradigm of Development Before the Fact (DBTF) for systems and software design.

Unfortunately, the original articles from the SEMEVAL 2010 Task 5 are transformed into plain text using the UNIX tool `pdftotext`. The output of this tool is a very unstructured text, where not only information about title, sections, etc., is lost, but also figures and tables may be placed inside content paragraphs, sentences may be badly split, and so on. This caused problems with the anaphora resolution and substitution algorithm, whose precision was undermined by these conversion errors. Moreover, these formatting problems may cause an erroneous coreference chain where the effects of an early bad resolution are amplified while going further in the text.

Thus, to improve anaphora resolution (and then substitution) in the original text, we segment the original text into sections. In this way, we improve the reliability of the parsing tree for the sentences and so we obtain a more correct performance in searching the antecedent. To perform this segmentation we use some heuristics to distinguish the title, the authors, and the email addresses, and to detect the boundaries of sections, paragraphs, figures, and so on. As a result, the text to process becomes more similar to the original graphical appearance in the PDF format, it is more structured, and it contains also less errors.

Then, by using this structure, we work on single sections, generating and using the coreference chain with the Stanford CoreNLP Coreference Resolution System to collect all the pronominal anaphors. Finally, we go back to the antecendent of each pronoun detected in the chain and we replace the former with the latter. The text turns out to be simpler, maybe even grammatically wrong, but more informative for our AKE algorithm: by replacing the pronouns in the document we have no loss of information, while we are able to recover statistical and semantical information about the antecedents that would be otherwise lost.

The choice of substituting only pronominal anaphors is justified by the fact that nominal anaphors may not be just synonyms but also very different words, possibly with a different meaning. This happens because nominal anaphors have only the property of referring to the same entity in common, thus substituting a nominal anaphora with its antecedent could change the meaning of the sentence. For example, from the text above, "Computer scientist", "system engineer", "Director of the Software Engineering Division", are all references to "Margaret Heafield Hamilton", but if we substitute them with the head of chain (i.e. "Margaret Heafield Hamilton"), the meaning of the sentence is completely different (actually, it becomes nonsensical). Considering proper name anaphors is worthless as well: replacing a proper name anaphora with its antecedent could lead to a substitution that in our opinion could be useless or wrong. For example, in a biography there could be more people referenced by a common surname. Thus, an arbitrary substitution of all proper name anaphors could be wrong, because the anaphora resolution software may fail to identify the correct subject: in our example, if the head of the coreference chain is "Hamilton", we risk to replace "Margaret Heafield Hamilton" with her husband, whose surname is Hamilton too.

To summarize this approach, we follow this process: first, we parse the article from the dataset and use some heuristics to divide it into correctly formatted sections. Then, we process each section with the Stanford CoreNLP Coreference Resolution System, we collect all the pronominal anaphors, and we replace each anaphora with the correct antecendent. Finally, we submit the preprocessed text to the AKE process along with the value of the *InAnaphora* feature. Regarding the *numOfReference* feature, which concerns only pronominal anaphors, its use has to be taken into consideration as well because when documents are preprocessed with substitution, different coreference chains could be discovered.

3.4 METHODOLOGY

3.4.1 *Baseline algorithm*

In order to evaluate the impact of the proposed features, we used the Distiller framework to implement a baseline keyphrase extraction algorithm with few basic features. As usual, our baseline algorithm candidate KPs are n-grams selected from the text if they match a given set of part-of-speech patterns.

The baseline feature set for our experiment is a set of well-accepted features for AKE, i.e., given a candidate KP, we consider:

- TF-IDF;
- relative position of the first appearance of the candidate (*height*);
- difference between the position of the last and the first appearance (*lifespan*);
- number of appearances of the candidate in the text, normalized by number of sentences.

Then, we consider a new feature set, in which we add to the baseline a fifth feature called Document Phrase Maximality (DPM). We use this feature because it supposedly should help to discriminate between candidate keyphrases which often appear as substring of another candidate. We deem this feature as necessary because, by using our substitution algorithm, we usually substitute an anaphora with a longer antecedent, thus leading to an increase of frequency of all the words contained in the antecedent. In our example, we will substitute the anaphors with "Margaret Heafield Hamilton", thus increasing the frequency, e.g., of the word "Hamilton", but DPM allows us to assign a *low* score to the single words while assigning an *high* score to the full name of the scientist.

Note that we already defined how we calculate these features in Section 2.3.3.1, so we won't repeat such definitions here. However, we must note that here the feature set is slightly different because we tuned our experiment differently, in order to obtain the best performance with this feature set.

The machine learning algorithms we choose are logistic regression, neural networks, and boosted decision trees, since these algorithms have a reputation of being good algorithms in the AKE community

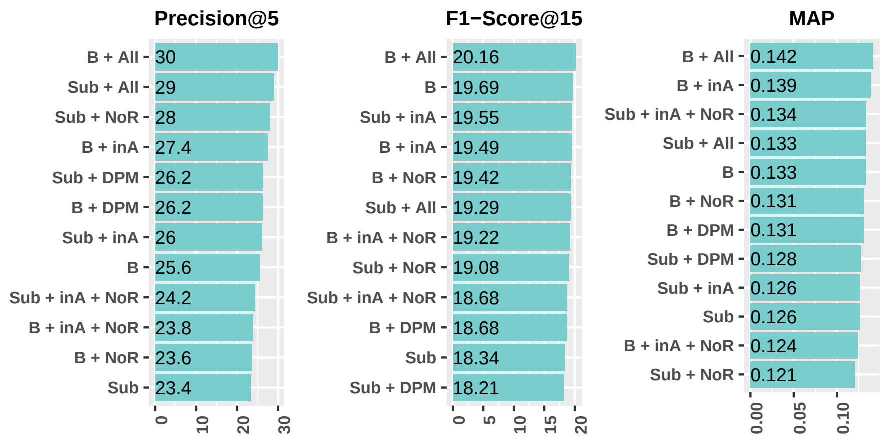


Figure 3.1: Scores obtained by running our keyphrase extraction algorithm over different feature sets. Note that "B" stands for "Baseline", "Sub" indicates the baseline features ran on the preprocessed documents, "inA" marks the `inAnaphora` feature, "NoR" marks the `numberOfReference` feature, "DPM" stands for the Document Phrase Maximality feature, and "All" indicates that the feature set contains all the aforementioned features.

[52]. We used their implementation with the R software, using the `glm`, `nnet` and `C5.0` libraries to train the respective models. We used no particular tweaking on the algorithms; the neural network used was a simple Multi Layer Perceptron (MLP) with one hidden layer. Then, we ranked KPs using the raw scores assigned by the algorithms.

As in Chapter 2, we didn't choose a bigger feature set because there is no agreement on which are the "state of the art" features for AKE. As stated in Section 1.2.1, the top performing systems use many different strategies: for example, Lopez and Romary [72], use few features, but a custom "post-machine learning" ranking algorithm; You, Fontaine, and Barthès [122] use few features, but a different candidate generation algorithm; in the work by Haddoud et al. [48], we can find more than 20 features instead.

Moreover, this simple feature set is enough to get an average performance on the SEMEVAL task. With the MLP, our baseline system showed an F-score of 19.69% on the best 15 keyphrases, which is good enough to be ranked 11th out of 20 contestant in the SEMEVAL 2010 challenge. The same position would be achieved using logistic regression, with a score of 19.22%, while the use of decision trees causes a slip of one position down, with a score of 18.95%.

3.5 RESULTS

Combining the baseline features defined in Section 3.4.1 with our new features defined in Section 3.3.3 and the pre-processing technique we described in Section 3.3.4, we defined a total of 36 different AKE pipelines. These pipelines were built by running the three machine learning algorithms we choose on the baseline feature set first, then adding DPM and our anaphora-based features, then combining the features together, both on the original SEMEVAL 2010 dataset and the text preprocessed with our technique.

The neural network was the best performing algorithm overall, and we can see its results in Figure 3.1. The first impression is that there is generally a little improvement in F1-Score, with the baseline algorithm on the original documents still being the second best feature set with this metric.

Nevertheless, looking at the Precision@5 score, we see that our approach has a significant impact: as shown in Figure 3.1 (left column), the combination of the linguistic features with the statistical ones, both in the original documents and in the ones with preprocessing, the precision score is greater than the one obtained for the baseline set. In particular, starting from the result of 25.60% for the baseline, we reach a score of 27.60% in precision just by using *inAnaphora* feature and 30.00% adding also *numOfReference* and *DPM*. A similar behavior can be observed in the results when considering that the substitution technique is performed in the preprocessing. In fact, while on the preprocessed text the baseline features show no improvement, a more interesting result can be seen using the linguistic features, for which precision score raises from the baseline's 23.40%, to 26.00% adding *inAnaphora*, and to 29.00% combining all the features together.

Looking at the scores of MAP, we can see that using all features on the original dataset offers the best ranking, as it would be expected by the high Precision@5 score; this confirms our idea of combining the anaphora-based features with DPM. Interestingly enough, most of the other feature sets show a slight decrease in the quality of the ranking, probably because the gain in precision is not high enough to balance the decrease in recall.

Taking into account just the second approach, the results provide evidence of our initial assumptions on the importance of using *inAnaphora* feature over preprocessed text. Precision and F1-Score show a more significant increment when using preprocessed documents, and the reason can be found in a second parsing with more *coherent* coreference chains. In details, coreference resolution and so our features that depend from it improve because the substitution of pronouns with the common antecedent in the first chain produces a text with more noun phrases and less pronouns. This way, the parse tree of the preprocessed text is simpler, so the relationships between the "new" noun phrases are more clear. This allows the anaphora resolution library to find more anaphors and to better detect pleonastic pronouns, thus obtaining a more precise score for our feature.

The other ML algorithms (not shown in the figures) seem to prove the conclusions we obtain from the neural network: using either decision trees or logistic regression the behavior is similar to the one described for the neural network, with a relatively stable F1-Score on the top 15 extracted keyphrases, but a significant increase in Precision@5 and MAP score when adding linguistic features. It is interesting that for both algorithms, using all features on the original documents offers highest Precision@5 and MAP scores, confirming the results shown in Figure 3.1. In particular, with this features/dataset combination, with the `glm` library we see slight rises in Precision@5, F1-Score@15 and MAP from 24% to 24.4%, from 19.22% to 20.30% and from 0.127 o 0.136 respectively; for `C5.0`, while F1-Score rises from 18.95% to just 19.29%, Precision@5 and MAP shows a more significant improvement from 22.2% to 26.8% and from 0.123 to 0.137 respectively, thus supposedly showing a better ranking of the keyphrases found. On the other hand, using the same feature set over the preprocessed documents still shows an improvement from the baseline, but with slightly lower scores.

3.6 CONCLUSIONS

Our analysis shows that anaphora and coreference resolution can be used for AKE with significant results. Like in [57], we see that by exploiting linguistic knowledge in a keyphrase extraction algorithm it is possible to increase the precision of the results. We think that it is important to analyze the relationships which could arise when linguistic features are combined together with statistical features. For example, it is clear that preprocessing the input text by substituting the pronouns with the entity they refer to could increase the frequency of certain terms, thus statistical features like DPM can be useful to gain a performance boost.

A better result could be obtained by improving anaphora resolution performance, since the software we used was not always able to find all the correct anaphors, even if it is (or it is close to) the state of the art system for anaphora and coreference resolution at the time of writing. For example, looking at the example we introduced in Section 3.3.3, the algorithm was not able to detect the anaphora *director* from the sentence "*She was Director of the Software Engineering Division*", which means that we would not able to detect and replace correctly all the pronouns in the coreference chains or compute the value of our features correctly. This is confirmed by the fact that the *numOfReference* feature, which is based on the count of pronominal anaphors, had a positive impact on performance even after the text preprocessing, which *should* have had replaced all the pronouns with their antecedents.

As a future work, we consider the idea of using the outcomes of our preprocessing stage for improving anaphora resolution specifically for the task of keyphrase extraction, developing an ad-hoc mining algorithm for the parsing trees, with the goal of producing a better pre-processing algorithm and finding for each valuable pronoun a good candidate antecedent. Moreover, another interesting approach would be looking for statistical features other than DPM which are able to better interact with the anaphora-related ones.

Part III

DEEP SUPERVISED KEYPHRASE EXTRACTION

This Part describes the effort of applying Deep Learning techniques to Automatic Keyphrase Extraction. First, we will analyze which parts of DL can be useful for improving AKE. Then, we will show how we applied this concepts, to design several network architectures that have been able to outperform several other AKE algorithms on the well known Inspec Dataset.

4 DEEP LEARNING BASICS

This Chapter will provide a gentle introduction to the concepts and techniques we will face in Chapter 5. We didn't analyze deeply the machine learning algorithms we used in the previous Chapters because, as stated in Section 1.2.1, there is at present not a proper state-of-the-art algorithm used in the "classical" supervised AKE approach, since all the algorithms used by scholars seem to offer decent performance, thus the justification in choosing an algorithm over another can be even just boiled down to the training speed [72].

On the other hand, when using Deep Learning techniques, we won't simply try to attach a *weight* to a *feature*: all the algorithms presented in Chapter 5, in fact, will be based on *word embeddings*, whose goal is to encode the *meaning* of single words. The use of a different architecture dramatically affects how such embeddings are interpreted and used to *learn new features* by the networks.

For this reason, while we won't delve deeply into the concepts of Deep Learning, in the present Chapter we will introduce the mathematical and philosophical foundations of the concepts that will be later applied to AKE. For the reader interested in a full introduction on Deep Learning, the book by Goodfellow, Bengio, and Courville [43] provides an excellent introduction to the subject.

4.1 WHAT IS DEEP LEARNING?

To understand why Deep Learning KE needs a different analysis from "classic" supervised KE, we analyze the difference between DL and "classical" supervised algorithms used in KE. The performance of the machine learning algorithms used in KE depends heavily from the *features* that the scholars designed to identify relevant KPs, i.e., it depends on the *representation* of the input data. As we already stated by using, e.g., logistic regression, a decision tree, or a support vector machine, the algorithm is not endowed with the possibility to learn from raw input data, i.e. it does not "see" the actual words that compose the content of the document. Instead, we feed it some higher-level information (the features) about each potential KPs, e.g., the position in the document, the number of occurrences, and so on.

On the other hand, deep learning can be seen as a branch of *representation learning*. In representation learning, the algorithm does not only learn the mapping from the features (i.e. the *representation*) to the output, but also the feature themselves [43]. DL differs from simpler representation learning techniques because it allows to build addi-

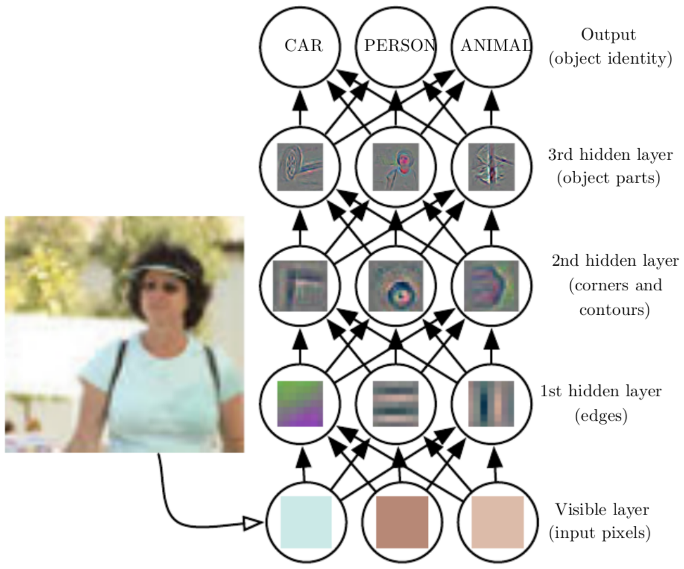


Figure 4.1: An example of representation learning: the first layer receives the input pixel and the subsequent *hidden layers* learn a more and more abstract representation that ultimately lets the network classify the image. Image reproduced from [43].

tional abstract representations of the data, by implementing neural networks with many layers (i.e. *deep networks*). Specifically, when we talk about DL, we are addressing the *multi-layer perceptron* (abbr. MLP) or its variants, which is a neural network with more than one hidden layer. In a MLP, the first hidden layer learns an abstract representation of the input values, e.g., the pixel that composing an image; the second layer, uses the information from the first layer to build a new representation; and so on. For example, in Figure 4.1 we see what happens when training a MLP to recognize if an image contains a person, a car, or an animal: the first hidden layer leans to recognize the edges, the second layer learns to recognize the contours, the third layer learns to recognize specific parts of the body (e.g. head, torso, eyes . . .), and the last layer uses such information to assign the input image to the correct class.

Deep Learning has been successfully applied to problems closely related to AKE as, for example, summarization [86, 102] or sentence classification [64], and some scholars, as already mentioned in Section 1.2.3, introduced deep learning Keyphrase Extraction algorithms [8, 78, 123] for extracting keyphrases from scientific papers or Twitter. The following Sections will introduce the DL concepts that we believe can aid the DLKE task.

4.2 NEURAL LANGUAGE MODELS

When working on images we have to learn from the pixels, as we have seen in the previous Section. On the contrary, when working on documents, we have to deal with *words*. For this reason, we have to find a representation of the language of the input document which is suitable to be handled by a MLP. MLPs take in input only numeric data, i.e., numbers or vectors of numbers, so we must transform the words in a numerical value accepted by the MLP.

The simplest solution is the *one-hot vector* representation. For example, suppose we have the input sentence:

```
The cat is on the table
```

We can build a dictionary Σ containing all the words of the documents, i.e.

$$\Sigma = \{\text{the}, \text{cat}, \text{is}, \text{on}, \text{the}, \text{table}\}$$

Then, we can assign each word a vector v, based on its position in the dictionary:

$$\begin{aligned} v_{\mathtt{the}} &= [100000] \\ v_{\mathtt{cat}} &= [010000] \\ &\dots \\ v_{\mathtt{table}} &= [000001] \end{aligned}$$

Now, all the records of each vector are zeros but the one marking the position in the dictionary, hence the name *one-hot*. Now we could give such one-hot representations in input to the MLP. However, this approach has several drawbacks: first, if the number of words in the input document (or corpus) is high, each one-hot vector can be very long (e.g., it can be even 100,000 elements long), with a consequent waste of space. Second, the one-hot representation does not carry any information about the words. For example, when two vectors are close (i.e. the "1" entry is close), this does not necessarily imply that their *meaning* is close. We can see this fact even in our example: $v_{\mathtt{the}}$ and $v_{\mathtt{cat}}$ have a completely different meaning, even if they are one next to the other.

Language models try to overcome the limitations about the context by modeling the probabilities of sequences of n words (i.e. n-grams), by computing the conditional probability of finding the n-th word given the preceding $n-1$-gram. Mathematically speaking, we have that

$$P(x_1, \dots, x_\sigma) = P(x_1, \dots, x_{n-1}) \prod_{t=n}^{\sigma} P(x_t | x_{t-n+1}, \dots, x_{t-1}) \quad (4.1)$$

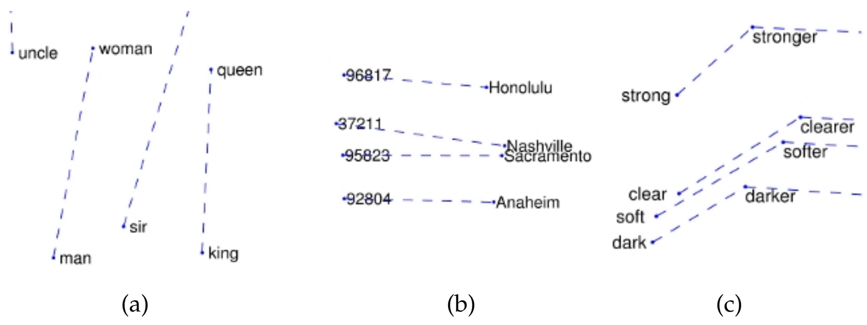


Figure 4.2: Example relations in the GloVe pre-trained word embeddings [97]. Here we see that the relation between “man” and “woman” is similar to the one between “king” and “queen” (a), that the NLM was able to associate each city to the corresponding ZIP code (b), and each word to its comparative (c). Images reproduced from `https://github.com/stanfordnlp/GloVe`.

However, we can see that modeling the joint probabilities of sequences of words leads to a problem called the *curse of dimensionality* [18]. For example, if we want to model 10-words long sequences from a vocabulary of size $|\Sigma| = 10^6$, we have $100000^{10} - 1 = 10^{50} - 1$ free parameters.

The solution for the curse of dimensionality is to build a distributed representation for using with *neural* language models (henceforth NLM), introduced by Bengio et al. [18]. In NLMs, a neural network builds a *vector* for each word of the dictionary, called *word vector* or *word embedding*. A popular way of calculating such vectors is the Skip-Gram architecture introduced by Mikolov et al. [83], where the neural network tries to predict a word based on its context, i.e. its previous and next words. Word vectors show interesting properties that make them very interesting for NLP applications. For example, it’s possible to *add* and *subtract* word vectors to find relations, as pointed out by [83]:

> To find a word that is similar to *small* in the same sense as *biggest* is similar to *big*, we can simply compute vector $X = \text{vector}(\text{“biggest”}) - \text{vector}(\text{“big”}) + \text{vector}(\text{”small”})$.

The solution is the word vector closest to X. Moreover, words having a close relationship between them are often organized by NLMs accordingly. For example, Mikolov et al. [82] found that the names of countries and their capital cities are organized efficiently by running a principal component analysis of a 1000-dimensional skip-gram vector space; in other words, the model seemed to learn what a “country” and a “capital city” are, since it associated each capital city to the correct country in the learned vector space. Figure 4.2 shows other examples of the self-organization of NLMs.

In practice, given a vocabulary Σ, to compute the word vectors we proceed in this way: we associate to each word of the vocabulary an index, e.g. in the example above we may assign `the` = 1, `cat` = 2, and so on. To limit the size of Σ, we often rank the words by frequency, and we keep only the n most frequent words. Then, we decide the size of our word vectors, and we compute them by running a NLM architecture (like the skip-gram introduced in [82, 83]) over a corpus. The larger the corpus, the higher will be the quality of the vectors. The result of the training is the NLM, where each word vector is a list of word indexes. For example, in our simple sentence, we may have that $v_{cat} = v(2) = 1, 3, 4$.

In DLKE, we represent the input documents using word vectors. However, since training word vectors is computationally very expensive, we use Stanford's GloVe pre-trained word vectors [97]. These vectors are trained using a vocabulary with 400 thousand uncased words and 6 billion tokens, and the resulting vectors are available in four different lengths (50, 100, 200, and 300).

Word vectors are the first step to introduce contextual information in AKE algorithms. However, to let the neural network learn from the context in a more efficient way, we will also use two particular architectures, which will be introduced in the following Sections.

4.3 RECURRENT NEURAL NETWORKS

The first architectures that we analyze are Recurrent Neural Networks (RNNs) and Long-Short Term Neural Networks (LSTMs). These architectures are more interesting for AKE than "regular" NNs because they take in input a *sequence* of words (represented by word embeddings) and at each step the weights of the network are updated taking in account the *previous* weights as well.

Formally, given an input sequence of word representations $x_1, \ldots,$ x_n, a Recurrent Neural Network computes the output vector y_t of each word x_t by iterating the following equations from t times, with t iterating from 1 to n:

$$h_t = \sigma(Ux_t + Vh_{t-1} + b_h) \tag{4.2}$$

$$y_t = \sigma(Wh_t + b_y) \tag{4.3}$$

where x_t and h_t are respectively the input and the hidden vectors at time t. U, V and W are respectively the weight matrices connecting the input layer and the hidden layer, the hidden layer with the hidden layer at the previous time step, and the hidden layer with the output layer. b_h and b_y are the bias vectors, and σ is the activation function of the hidden layer and output layers. The important bit in Equation 4.2 is that a RNN has connections between the previous hidden state h_{t-1} and the current state, thus RNNs can make use of previous context.

In practice, however, the RNN is not able to manage efficiently long sequences due to the *vanishing gradient* problem [56]. To overcome this problem, Hochreiter and Schmidhuber [55] proposed the Long Short-Term Memory (LSTM) architecture, a variation of the RNN that is supposed to effectively learn on broader input sequences.

LSTMs are defined similarly as RNNs, with exception for the fact that hidden layer updates are enhanced by specific units called "memory cells", designed to solve the vanishing gradient problem by keeping track of long dependencies.

The LSTM works by sequentially updating its internal state, according to the values of three sigmoid gates. Specifically, a LSTM is implemented using the following functions:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{4.4}$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{4.5}$$
$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{4.6}$$
$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \tag{4.7}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{4.8}$$
$$h_t = o_t \tanh(C_t) \tag{4.9}$$

where σ is the activation function, i, f, o, and C are respectively the input gate, the forget gate, the output gate and the cell activation vectors, and all bs are the learned biases. We won't go deep into the mathematical details here, but an high-level explanation of the LSTM is the following:

- the activation function σ is the logistic sigmoid or a similar function ranging from 0 to 1;

- the forget gate f_t decides if is worth "remembering" the old cell state C_{t-1}. To understand why, see how the value computed in Equation 4.4 is used in Equation 4.7;

- Equation 4.5 and 4.6 generate the new candidate cell state, that will be combined with the result of the forget state in Equation 4.7 to create the new cell state C_t;

- Equations 4.8 and 4.9 compute the actual output of the cell, i.e. the new hidden layer, using the cell state C_t and information from the previous hidden layer h_{t-1}.

One shortcoming of both RNNs and LSTMs is that they consider only the previous context. However, in the keyphrase extraction task, where the whole document is given in input, we want to exploit *future* context as well. For example, consider the phrase "John Doe is a lawyer; he likes fast cars". When we first encounter "*John Doe*" in the phrase, we still don't know whether if he's going to be an important entity; then, we find the word "*lawyer*" and the pronoun "*he*", which clearly refer to him, stressing his importance in the context.

In order to overcome the limitations of RNNs and to use information from the future context as well, in our approach we adopt a Bidirectional LSTM network [44] (Bi-LSTM). In fact, with this architecture we are able to make use of both past and future contexts of the word evaluated at each time step. The Bi-LSTM consists of two separate hidden layers. It first computes the forward hidden sequence $\overrightarrow{h_t}$, then it computes the backward hidden sequence $\overleftarrow{h_t}$. Finally, it combines $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ to generate the output y_t. It is implemented by the following composite functions:

$$\overrightarrow{h_t} = H(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}) \tag{4.10}$$

$$\overleftarrow{h_t} = H(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}) \tag{4.11}$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y \tag{4.12}$$

where the hidden states are represented with LSTM blocks.

4.4 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNN) are neural networks where matrix multiplication is replaced by convolution [43]. While the most common application for Convolutional Neural Networks is image processing, they have been successfully used on many NLP tasks, such as sentiment analysis [105], sentence classification [64], or answer selection [38]. CNNs are based on three important concepts: *sparse interaction*, *parameter sharing* and *equivariant representations*. Sparse interaction is a concept in contrast with traditional NNs, where matrix multiplication is used to make each input interact with each output unit. In a CNN instead we use a *filter* (also named *kernel*) which is much smaller than the input. This small filter is reused, i.e., is applied to each position of the input, hence the concept of parameter sharing. Having to store (and learn) fewer parameters, CNNs are usually faster and use less memory than traditional NNs [43].

This kind of parameter sharing leads to the concept of equivariant representation. A function $f(x)$ is defined equivariant to a function g if $f(g(x)) = g(f(x))$. Convolutional operators are *equivariants* with respect to translation, i.e., the position of a learned feature in the input is irrelevant [38], a property desirable in many NLP applications.

To understand how CNNs work on text, we propose an example adapted from Kim [64]. Suppose that we have a input sentence of length n $S = x_1, \ldots x_n$, where each word is represented by a k-dimensional word embedding; we denote a *slice* $x_{i:i+j}$ the concatenation of the words ranging from index i to index j. The convolution operator consists in a filter $W \in \mathbb{R}^{hk}$, applied to h k-dimensional word embeddings to produce a feature. For example, a feature c_i is generated from a slice $x_{i:i+h-1}$ as:

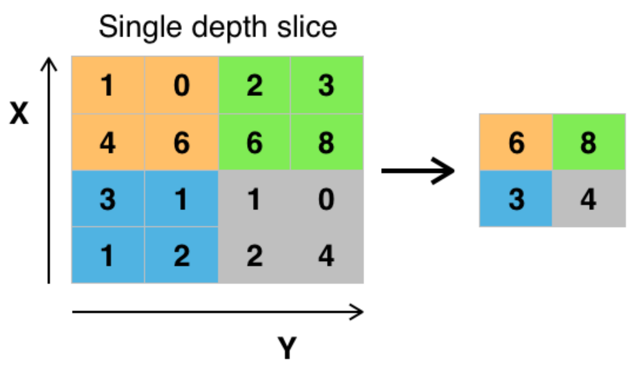


Figure 4.3: An example of max-pooling over a 4×4 matrix, with a 2×2 filter and stride $= 2$. Image reproduced from https://commons.wikimedia.org/wiki/File:Max_pooling.png and licensed with Creative Commons BY-SA 4.0 license.

$$c_i = f(W \cdot x_{i:i+h-1} + b)$$

where b is the bias term an f is a nonlinear function, such as tanh.

After the convolutional filter, almost all CNNs employ an operation called *pooling* [28, 43], which replaces the output of the convolutional layer with a statistical summary of nearby outputs; we see an example of pooling in Figure 4.3. The two most popular approaches are *max-pooling* and *average polling*, which take respectively the maximum value of an output window (in our example $\tilde{c} = \max\{c\}$) or the mean of the values of the output windows. In our architectures, however, we will stick to max-pooling, since it better deals with different sentence lengths in a more efficient way [64].

4.5 FINAL REMARKS ON DL

This Chapter introduced the main deep learning concepts that we believe are helpful in the field of keyphrase extraction. While we didn't deeply investigate the mathematical details of DL, we introduced three concepts that can, at least in principle, be useful for AKE:

- Word vectors, which can help a machine learning model to better understand the topics of the input document. In fact, by *embedding* the meaning of each word in a vector, they should help to overcome the limitations of "classic" supervised ML AKE algorithms, which do not "see" the words but only their features.
- Long-Short Term Memory networks, which can help us to write algorithms that mimic how humans read documents. In fact, we can use LSTMs to build a DL algorithm that scans a document, word by word, keeping *memory* of the what it has previously

read. Bidirectional LSTMs can further enhance this ability by reading the document twice; one, from the beginning to the end, and one from the end to the beginning.

- Convolutional Neural Networks, which, while intuitively seem more useful for image processing, have been successfully used in many NLP tasks closely related to AKE, like summarization of answer selection. They show some properties that are desirable when working on AKE, like the translation-invariance of the learned features, which may help to understand *if* a concept appears in a document regardless of *where* it appears.

In the next Chapter we will apply all these concepts to build and evaluate different DL architectures in order to improve the AKE task.

5 ARCHITECTURES FOR DEEP KEYPHRASE EXTRACTION

In this Chapter we will apply the concepts we introduced in Chapter 4 to AKE, in order to explore different network architectures for Deep Keyphrase Extraction. We will evaluate such architectures on a well-known dataset: the Inspec abstract dataset introduced by Hulth [57].

Each of these architectures will follow the path, introduced in Chapter 3, of exploiting the context of each potential keyphrase in order to build a more efficient AKE algorithm. Moreover, we will take advantage from the neural language models and architectures we introduced in Chapter 4.

A seminal part of the work presented in this Chapter will be presented in Basaldella et al. [17] at the 14th Italian Research Conference on Digital Libraries (IRCDL 2018), which will be held in Udine, January 25-26, in 2018.

5.1 INTRODUCTION

By now, to the best of our knowledge, the proposed deep learning architectures for AKE have been fairly conventional. For example, Zhang et al. [123] proposed a joint-layer RNN; Meng et al. [78] proposed a bidirectional Gated Recurrent Unit, an architecture fairly similar to the LSTM layer we described in Section 4.3, and an encoder-decoder architecture to generate KPs not present in the input document. Ammar et al. [6] proposed a more complex model, with a LSTM followed by a Conditional Random Field (hence CRF), but with the caveat that the they were competing for SEMEVAL 2017, where the challenge was not only to *recognize* keyphrase but also to *categorize* them, hence the use of the CRF. Tsujimura, Miwa, and Sasaki [114], who participated to the SEMEVAL 2017 challenge as well, used a LSTM stacked with a fully connected layer, obtaining the third place overall. All these architectures use word embeddings to represent the input words.

The techniques we propose in this Chapter will try to enhance these architectures. After developing a simple LSTM-based network, in fact, we will introduce new architectures inspired from Paragraph Vectors [67] and Document Embeddings [29]. In short, we will develop several architectures which will be based all on the same concept, i.e., creating a *summary* of the document inside the network and using such summary to extract better keyphrases.

As anticipated, we will evaluate our architectures on the Inspec dataset introduced by Hulth [57]. The choice of this dataset is motivated by two reasons: first, it's an historical dataset in the AKE community, with several results to compare with. Second, differently from many other datasets, it is quite large[1], with 1000 training documents, compared with the only 144 provided by the SEMEVAL 2010 challenge and 350 provided by the SEMEVAL 2017 challenge.

Differently from what presented in Chapters 2 and 3, here we will evaluate our system using not only Precision, Recall and F1-Score on the top 5,10, and 15 KPs (see Section 1.6), but also on *all* the KPs extracted by the system. In this way, we will be able to compare our results with the ones obtained by Hulth, by the TextRank system [80], and by the RAKE system [101].

Moreover, we will compare our results with the ones obtained by the only other[2] DLKE algorithm evaluated on the Inspec corpus [78], i.e. the F1-Score@10 obtained by their Copy-RNN algorithm and by TF-IDF, KEA [121], TextRank [80] and SingleRank [117]. CopyRNN obtained the best result of 34.2 F1-Score on the Inspec dataset outperforming all other algorithms, and we use this result as our baseline.

5.2 NETWORK ARCHITECTURES

All the network architectures presented in the current Chapter are written in Python, using the Keras [25] and Theano [3] libraries. The code is collected in a repository called `deepkeyphraseextraction`, which is described in detail in Appendix B. We leave the technical details in the Appendix; for the scope of the present Chapter, it's worth noting that as word embeddings, we use Stanford's GloVe pretrained embeddings [97] with a dimensionality of 300, and to perform all low-level NLP operations (splitting, tokenizing, etc.) we use Python's NLTK library [19].

5.3 ARCHITECTURE I: BIDIRECTIONAL LONG-SHORT TERM MEMORY NETWORKS

The first architecture, depicted in Figure 5.1, is a somewhat basic deep neural network, similar to the one presented in other DL-based AKE algorithms [113, 123]. In this architecture the contextual information is given simply by the Bi-LSTM layer, which "scans" the input document back and forth, and learns how to label the *sequences* corresponding to keyphrases in the input document.

Borrowing from the field of NER, in fact, we represent each document using the list of the words composing it, where each word

1 The Inspec dataset is large in *keyphrase extraction terms*. Common training sets for image recognition problems, for example, can have *millions* [43] of training examples.
2 to the best of our knowledge

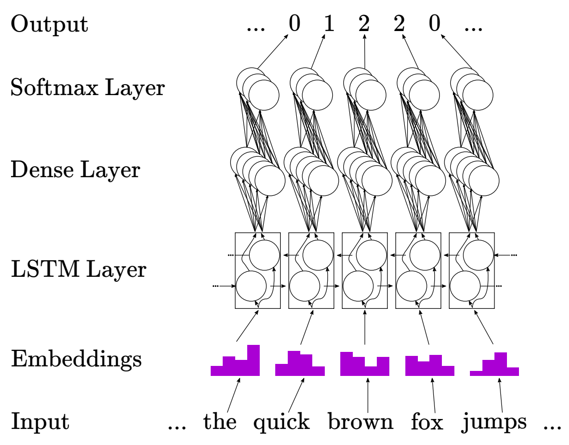


Figure 5.1: The first DL architecture: a Bi-LSTM layer followed by a fully connected layer a softmax output layer; after the LSTM layer and the dense layer we apply Dropout [110] to prevent overfitting.

is mapped into three possible output classes: `NO_KP`, `BEGIN_KP`, and `INSIDE_KP`, which respectively mark tokens that are *not* keyphrases, the *first* token of a keyphrase, and the other tokens of a keyphrase.

For example, if our input sentence is "*We train a neural network using Keras*", and the keyphrases in that sentence are "*neural network*" and "*Keras*", the tokens' classes will be the following:

```
We/NO_KP train/NO_KP a/NO_KP neural/BEGIN_KP
network/INSIDE_KP using/NO_KP Keras/BEGIN_KP
```

The network will be then trained to recognize such sequences. In detail, the steps of the AKE algorithm are the following:

- First, we split the input document into tokens using NLTK, and we prepare the input by associating each token to its pre-trained word vector, and the output by associating each token to its class. Please note that we will omit this step in further descriptions of DLKE architectures, as it will be common for all of them.
- The sequence of word embeddings is passed to a bidirectional LSTM layer with 600 neurons, built as described in Section 4.3;
- The output of the Bi-LSTM layer is passed to a fully connected layer with 150 neurons for each input word, using as activation function a rectified linear unit (ReLU) [85];
- The output of the fully connected layer is then passed to a layer performing a three-neuron softmax for each word, i.e. assigning each input token to the probability that it belongs to each of the three output classes. We pick the class with the higher probability assigned to each word as the output value of the network.

Note that, between the Bi-LSTM layer and the fully connected layer, and between the fully connected layer and the softmax layer, we apply 0.25 dropout [110] to help prevent overfitting, and that the number of neurons, the activation functions, and so on, have been tuned to maximize the performance of the network.

5.4 ARCHITECTURE II: TWIN READER NETWORKS

A Bi-LSTM is an effective way of using context to enhance AKE and is, apparently, enough to obtain a convincing performance, as we will see in Section 5.8. However, we believe that it's possible to train models which extract better KPs by using an even broader context than the one provided by a single recurrent neural network. For example, [67] introduced *paragraph vectors*, which are able to learn "fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents", which were later used by [29] to create *document embeddings*, which they found to be similar to word embeddings when performing vector operations (e.g. cosine similarity) to evaluate their semantics.

In this Section, we propose two different architectures inspired to such concepts. In these architectures, we build a vector representation of the input document using either a Bi-LSTM or a Convolutional Neural Network. This representation is then concatenated to each pre-trained word vector and fed to another Bi-LSTM network using these "combined" vector as in the Bi-LSTM network presented in Section 5.3. We will call these networks "reader" networks, to better identify them with respect to the other twin architectures that will be presented in the following Sections.

5.4.1 *Twin LSTM*

The second DLKE design is depicted in Figure 5.2. It is composed by two Bi-LSTM networks, where the "left" one (in Figure) is used to produce a neural representation of the input document, and the "right" one uses such representation along the word embeddings to extract the KPs. In details, this DLKE algorithm works as follows:

- The sequence of word embeddings is passed to a bidirectional LSTM layer with 300 neurons;
- The output of the Bi-LSTM layer is passed to a fully connected layer with 300 neurons, i.e., the size of the input word embeddings;
- The output of the fully connected layer is concatenated to each word embedding, and it is fed to the "right" part of the network, which is similar to the one presented in Section 5.3;

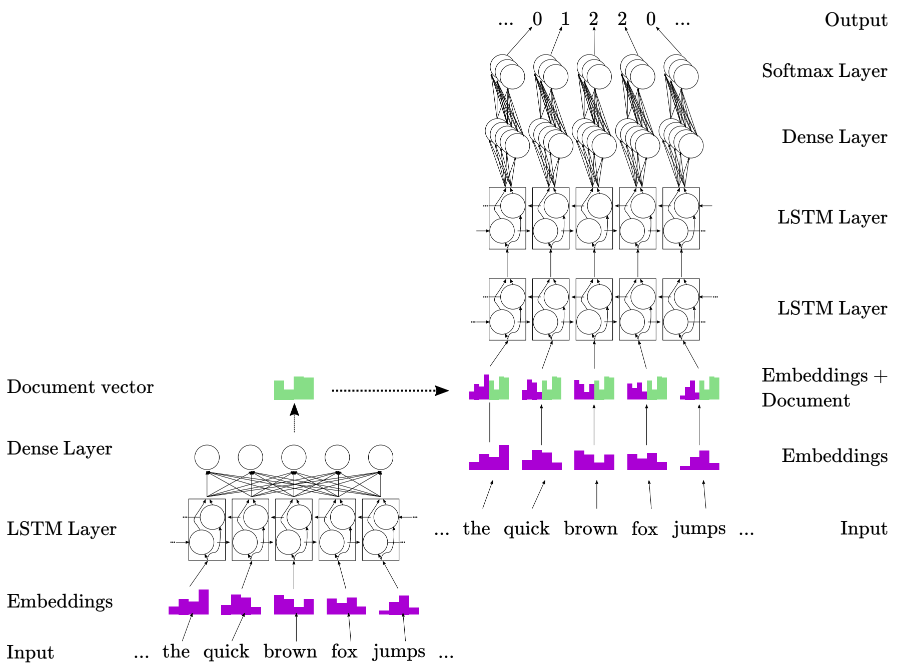


Figure 5.2: The first twin DLKE "reader" architecture: a Bi-LSTM layer is used to produce a document vector, which is then concatenated to the word vectors and fed to another Bi-LSTM network. The example input and output is the same as in Figure 5.1.

- The "right" part of the network consists in two Bi-LSTM layers with 300 and 150 output neurons each, followed by a fully-connected layer;
- Finally, the output of the twin network is again a three-neuron softmax layer built as in the previous section.

Again, here we use Dropout between each layer to prevent overfitting, and the activation functions are the ones used in the previous network.

5.4.2 *Twin LSTM/CNN*

In this third architecture, the "left" part of the twin network is composed by a Convolutional Neural Network (see Section 4.4). In short, we employ CNNs since they have been successfully used in other NLP tasks, with the advantage of being faster to train. The network architecture is the same as in the previous network, with the difference that the "left" network learning the document representation is composed by three convolutional layers. These layers use 128, 128, and 60 filters respectively, a kernel size of 32, 8 and 4 respectively, with strides of 4, 2 and 1. Each convolutional layer is followed by a max pooling layer, which is fully connected to the next convolutional

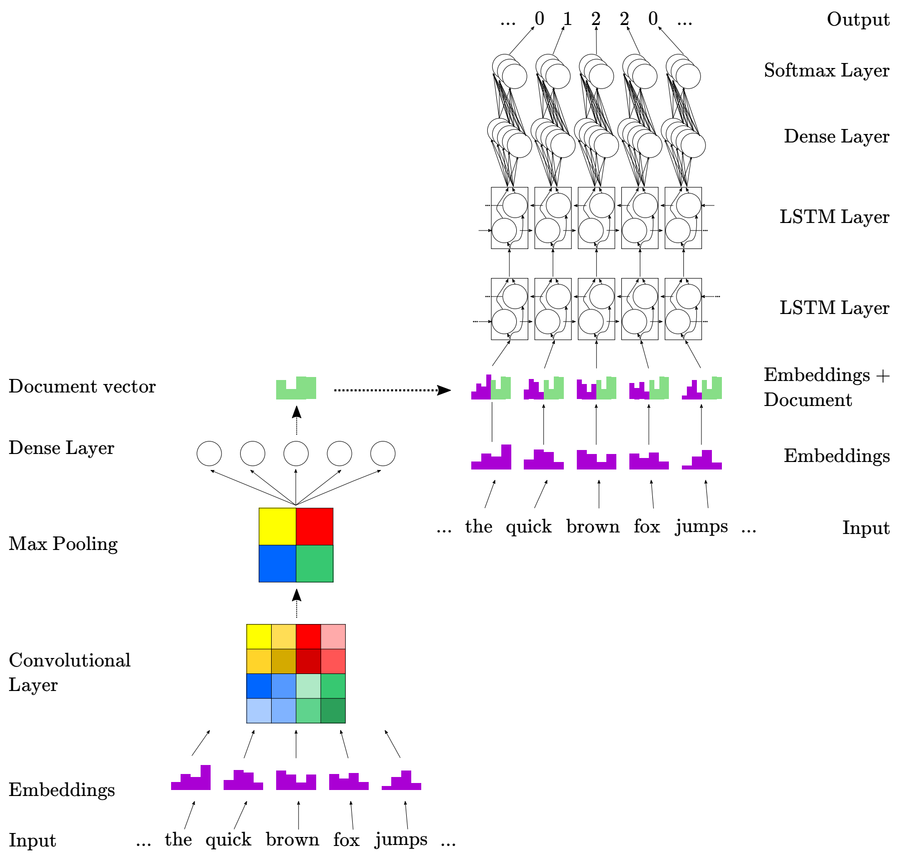


Figure 5.3: The second twin DLKE "reader" architecture: the left convolutional network is used to produce a document vector, which is then concatenated to the word vectors and fed to a Bi-LSTM network. The example input and output is the same as in Figure 5.1.

layer. The output of the last layer is flattened and then, as in the previous case, concatenated to the word embedding and fed to the "right" part of the network.

5.5 ARCHITECTURE III: KEYPHRASE SELECTION WITH TWIN NEURAL NETWORKS

The first three architectures we presented, are inspired from the NER task, and they are no novelty in the deep learning field. For example, Collobert and Weston [27] proposed a unified neural framework for performing many NLP tasks, such as part-of-speech tagging or NER, and all the top architectures presented in the SEMEVAL 2017 competition actually solve a task similar to ER, but there are many earlier examples of DL applications for NER. Differently from such approaches, for our last family of networks we will use some of the techniques for AKE that were developed during the last 20 years, and we will combine them with techniques from the field of Question Answering (herein QA).

In particular, we will develop a variation of the task of Answer Selection, which is defined as follows [38]:

Definition 5.5.1 (Answer Selection)**.** Given a question q and an answer candidate pool $A = \{a_1, \dots, a_n\}$ for that question, the goal of an Answer Selection algorithm is to select the correct answer $a_k \in A$.

We can thus define our new AKE task this way:

Definition 5.5.2 (Keyphrase Selection)**.** Given a document d, we define the candidate keyphrases pool for d as: $C^d = C^d_{correct} \cup C^d_{wrong}$, where $C^d_{correct} = \{kp_1^{(D,t)}, \dots, kp_n^{(D,t)}\}$ is the set the of correct keyphrases of d, $C^d_{wrong} = \{kp_1^{(D,f)}, \dots, kp_m^{(D,f)}\}$ the set of the wrong keyphrases of d and $C^d_{correct} \cap C^d_{wrong} = \emptyset$.

The goal of a Keyphrase Selection (herein KS) algorithm is, given in input d and C^d, to select from the candidate pool only the correct keyphrases, i.e. to select the subset $\{kp \in C^d | kp \in C^d_{correct}\}$.

Note that this problem is exactly the binary classification problem solved by "classic" supervised AKE algorithms such as the ones we analyzed in Chapters 2 and 3. However, it's useful to introduce this definition to differentiate the network we propose in this Section, which will be identified as "KS" networks, from the networks proposed in Sections 5.3 and 5.4, which we called "Reader" networks.

Feng et al. [38] and Tan, Xiang, and Zhou [111] analyzed several architectures for QA, all based on the same concept: they proposed many twin networks, where the first half of the network takes in input the question, and the second half of the network in input takes a candidate answer; one or more layers "read" the input and build a neural representation of both the question and the answer, which are then compared using cosine similarity. The layers of the network can be Bi-LSTMs, CNNs, or a combination of both of them. Such models obtained interesting results in the QA domain and are similar to the ones we used in Section 5.4.

5.5.1 *Twin LSTM*

The first KS architecture we propose is based on a twin Bi-LSTM network design, and it's shown in Figure 5.4. This network works similarly to the network presented in Section 5.4.1, with the difference that the "right" LSTM does not receive the whole input document, but only the candidate keyphrase. In detail, the network design is the following:

- Two Bi-LSTM layers receive respectively the input document and the candidate keyphrase, represented as sequences of word embeddings;

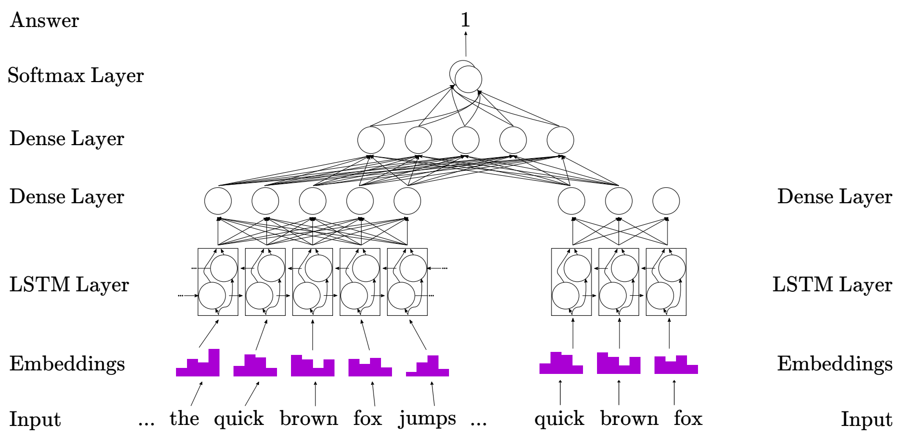


Figure 5.4: The first KS network: two Bi-LSTM networks receive the input document d and the candidate keyphrase kp; a dense layer receives their output, which is then connected to a softmax layer returns the probability that kp is a keyphrase for d.

- The output of the two Bi-LSTMs is then passed to two fully connected layers;
- The output of the two fully connected layers is merged into another fully connected layer;
- Finally, a softmax layer with two neurons assigns the probability that the candidate is an actual keyphrase.

We use the usual dropout rate between the layers, the same activation functions as before for the Bi-LSTM layers, and tanh as activation function for the dense layers. However, differently from the networks presented in Section 5.3 and 5.4, here the Bi-LSTM layer does *not* return a temporal sequence. In other words, the fully connected layers after the Bi-LSTMs, are connected to all neurons inside it, while in the previous networks we had a dense layer connected to each time step (word).

5.5.2 *Twin CNN*

The second KS network, depicted in Figure 5.5, is more similar to the ones proposed for the QA task. In fact, with this model we explicitly ask the network to compare the *similarity* between the input document and the candidate keyphrase. This is the process followed by this architecture:

- Exactly as in the previous architecture, two convolutional layers receive the input document and the candidate keyphrase, represented as sequences of word embeddings;
- The output of the two convolutional layers is then passed to a max pooling layer;

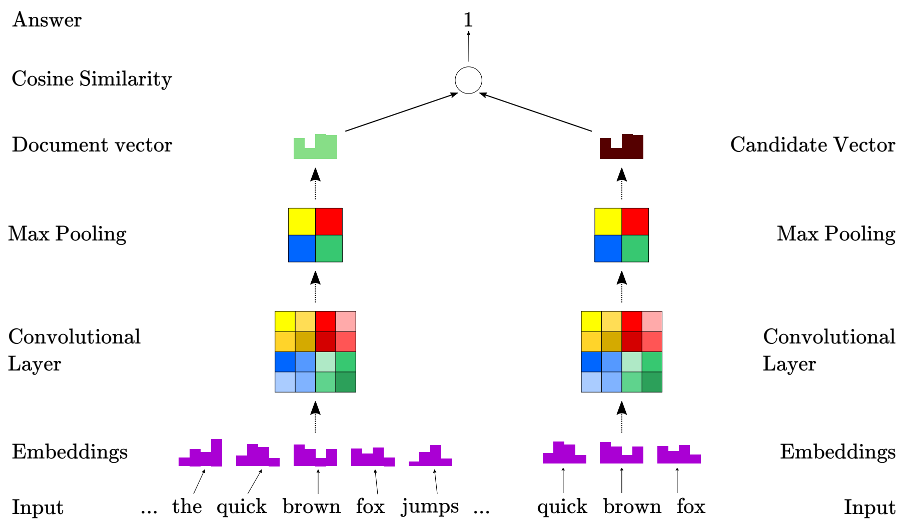


Figure 5.5: The second KS network: two CNNs receive the input document d and the candidate keyphrase kp; their output is flattened into two vectors, which are compared with cosine similarity. Note that on the left side of the network there are actually three convolutional and max pooling layers, but they are omitted for simplicity.

- On the "left" side of the networks, other convolutional filters and max pooling layers are applied, until the last max-pooling layer output is the same size as the output of the "right" max-pooling layer applied on the candidate keyphrase;
- The output of the two branches is merged and then we evaluate their similarity, i.e., the output of the max-pooling layers is flattened into two vectors, which are then compared using cosine similarity.
- The cosine similarity is compared with the ground truth, using the mean squared error as loss function.

The loss function that we use is much simpler than the one proposed by Feng et al. [38]; however, as we will see in Section 5.8, it is enough to obtain satisfactory results. Note that in the "left" side of the network we apply three convolutional layers with 128 filters and kernel sizes of 32, 8 and 4, and strides of 4, 2 and 1 respectively. On the other hand, the "right" side of the network convolutional layer has 128 filters, a kernel size of 4 and stride 1, since it has to deal with a much smaller input. These values are tuned specifically on the Inspec dataset in order to obtain two vectors of the same size to perform cosine similarity. For this reason, these vector would have to be calculated again if one wants to test this architecture on another dataset containing longer documents (e.g., on the SEMEVAL 2010 dataset, which contains full papers instead of only the abstracts).

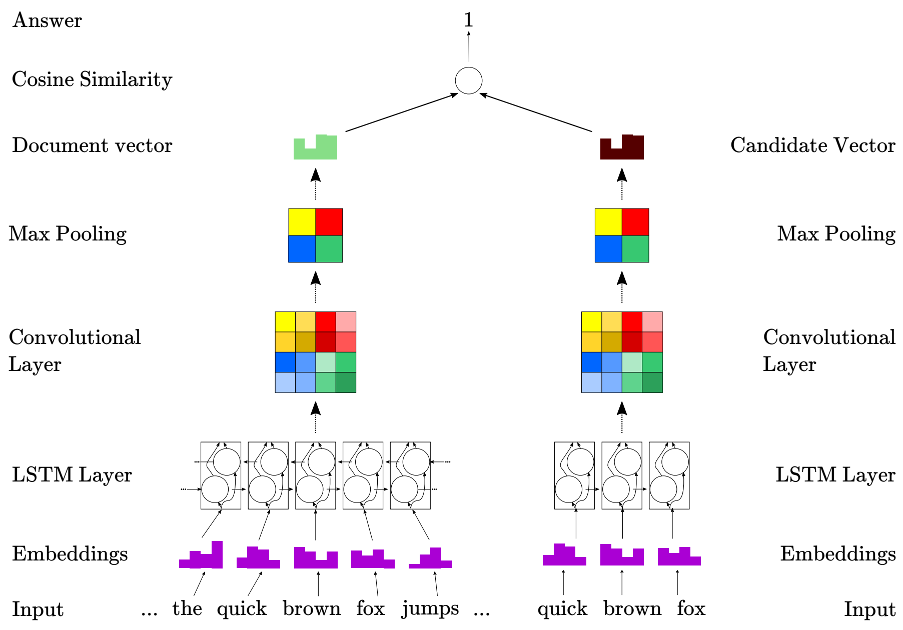


Figure 5.6: The third KS network: two Bi-LSTM layers receive the input document d and the candidate keyphrase kp; then, two CNNs receive their input, and the output of the CNNs is in turn flattened into two vectors, which are compared with cosine similarity. Note that on the left side of the network there are actually three convolutional and max pooling layers, but they are omitted for simplicity.

5.5.3 *Twin LSTM/CNN*

The LSTM/CNN last architecture, described in Figure 5.6, is very similar to the previous one, with the only difference that the CNN layers are preceded by a Bi-LSTM layer, similarly to the networks proposed for QA [38, 111]. We won't describe this network in detail as the only difference from the architecture presented in Section 5.5.2 is the aforementioned Bi-LSTM layer.

5.6 CANDIDATE KEYPHRASE GENERATION

Before delving in the details of the results obtained by our DL architectures for AKE, we will analyze how we perform the candidate generation phase. We covered the generation of candidate keyphrases in "classical" AKE algorithms extensively in Chapters 1, 2 and 3; the straightforward solution to build a training set could be then to use these candidate generation techniques for KS as well, i.e. building positive and negative example using part-of-speech patterns. However, the use of word embeddings offers new, different possibilities when training DLKE algorithms.

In fact, when developing "classic" supervised AKE algorithms, the training set must be composed only by candidate keyphrases appearing in the input document. This is straightforward, since phrases not appearing in the document have zero value in most of their features, e.g, they have zero depth, zero height, zero frequency, zero TF-IDF, and so on[3]. Finally, even if these candidate keyphrases had a non-zero feature vector, they would be impossible to generate, because by definition they don't appear in the input document. Thus, when training our DLKE networks for KS, we used two different candidate generation techniques.

The first technique is the same as the one used in "classical" supervised AKE algorithms: we select from the input documents the phrases matching certain part-of-speech patterns, and we use them as candidate keyphrases. In this case, the positive examples are the generated phrases that match with the gold keyphrases. This technique produces an heavily unbalanced set; for this reason, in Chapter 2 and 3 we used a random sampling technique, forcing a $1 : 9$ ratio between correct samples and wrong samples (i.e., out of ten samples, one is correct) and weighing them accordingly when training the model. In this Chapter, since we use the Inspec dataset, the documents are shorter, hence we have less candidate keyphrases. However, the number of negative samples still outcounts the number of positive samples. For this reason, we weigh our samples proportionally, e.g., if the number of negative samples is 7 times the number of positive samples, each positive sample will have weight 7 and each negative sample will have weight 1[4].

The second technique we address uses *all* the gold keyphrases as training examples. We are able to do this because the nature of the networks we developed does not require the presence of the candidate phrase in the input document. Thus, if we have a phrase which does not appear in the document, but we still know that it represents its contents well because it has been inserted in the gold standard, we can use it to train our model anyways. Hence, our training set will be composed, for each document, by all the correct gold keyphrases *even if they do not appear in the document* and by the same number of part-of-speech generated wrong keyphrases. Thus, the resulting training set is smaller, with 19539 samples, compared to 37007 samples generated by the PoS pattern technique; however, it has a perfect balance between correct and wrong samples and it contains a higher number of

3 Candidate KPs not appearing in the input document may still take non-zero values on other features, like for example the features designed to incorporate external knowledge (e.g., if a candidate appears in Wikipedia, its *Wikiflag* [12] feature can be set to 1). However, as we have covered in Chapter 1, such features are the minority in AKE domain, so having a feature vector with almost all zeros it's not useful when training a model.

4 Note that this is just an example; the actual weights are calculated automatically using the `sklearn` library.

System	P	R	F1	Epochs	Time (s)
Hulth [57]	25.2	51.7	33.9		
TextRank [80]	31.2	43.1	36.2		
RAKE [101]	33.7	41.5	37.2		
"Reader" LSTM	34.7	52.3	41.7	28	34
"Reader" Twin LSTM	37.3	**54.1**	**44.2**	29	110
"Reader" LSTM/CNN	37.5	50.7	43.1	43	70
"KS" LSTM	**42.5**	42.8	42.7	8	317
"KS" CNN	38.2	44.7	41.2	9	4
"KS" LSTM/CNN	39.3	46.1	42.4	8	61

Table 5.1: Overall results of the DLKE algorithms we proposed compared with results obtained by other systems over the same dataset. The last two columns mark the number of epochs required by the corresponding network to converge and the duration, in seconds, of each training epoch.

correct samples. As expected, with this technique we obtained better results than using the PoS pattern generated candidates only, hence we present only the scores obtained with the balanced training set.

5.7 HARDWARE AND SOFTWARE

All the DLKE networks were trained using the same hardware, i.e., a workstation using Ubuntu Linux 16.04.2 LTS, with an in Intel Xeon E5-1620 processor, 16 GB RAM and a Nvidia Titan X Pascal. The software used is described in Appendix B and it will be published as open source software on GitHub, correlated both with the specific versions of the libraries we used and some scripts we wrote to ensure the reproducibility of the results we obtained.

5.8 RESULTS

We present our results in Table 5.1, comparing them with the results obtained by Hulth [57], TextRank [80], and RAKE [101]. It's important to note that, for evaluation purpose, a keyphrase is considered correct if its stemmed string matches with a stemmed gold keyphrase. In order to perform stemming, we use the Porter stemmer [98], as the authors of the other AKE algorithms do. It's straightforward to see that our algorithms are the best performing ones overall. The best overall system is the "Reader"[5] Twin LSTM, with 44.2 F1-Score; however, the "KS" LSTM network obtains the best precision with 42,5%.

5 Remember that we label "Reader" networks the ones presented in Sections 5.3 and 5.4, while we call "KS" networks the one presented in Section 5.5.

System	F1@5	F1@10
TF-IDF	22.1	31.3
TextRank	22.3	28.1
SingleRank	21.4	30.6
ExpandRank	21.0	30.4
KEA	9.8	12.6
CopyRNN	27.8	34.2
"Reader" LSTM	31.8	40.1
"Reader" Twin LSTM	**34.5**	**43.1**
"Reader" LSTM/CNN	33.3	41.5
"KS" LSTM	32.3	40.3
"KS" CNN	30.2	38.2
"KS" LSTM/CNN	31.8	40.1

Table 5.2: F1@5 and F1@5 scores of our DLKE algorithms compared to other systems in literature: TF-IDF, TextRank [80], SingleRank and ExpandRank [117], and CopyRNN [78]. All the values of the systems are taken from Meng et al. [78]'s work.

Comparing the F1@5 and F1@10 scores obtained by our system with the results obtained by TF-IDF, TextRank [80], SingleRank and ExpandRank [117], and CopyRNN [78], we can still see that the "Reader" Twin LSTM architecture described in Section 5.4.1 holds the best results, outperforming both the unsupervised methods (TF-IDF, TextRank, SingleRank and ExpandRank) and the supervised ones. We show these results in Table 5.2.

However, it's worth noting that the maximum recall that KS algorithm can obtain is 59,33%, since the PoS pattern candidate generation technique is *not* able to generate all the possible keyphrases, while the "Reader" network can obtain a maximum recall of 76,91%, which is the number of gold keyphrases actually present in the test documents. In this work, we did not focus on "smart" techniques to generate candidates, so we suppose that these results can be improved. For example, Hulth [57] showed that selecting all the n-grams not starting or ending with a stopword as candidate keyphrases can improve the precision.

It's also interesting to look at the different times required by the networks to converge. All the twin networks take more time to train than the first architecture proposed by us, except for the twin CNN, which takes only 4 seconds per epoch to converge, for the motivations we already explained in Section 4.4. It's also not surprising that the "KS" LSTM network takes the longest time per epoch since, as we already mentioned in Section 5.6, the training set size of the "KS" networks is much bigger with 19539 samples, compared with the only

500 input documents that we can use with the "Reader" architecture. The bigger training set size of the "KS" networks, however, helps such networks to converge faster in terms of number of epochs required.

5.9 CONCLUSIONS

In this Chapter we presented six different deep learning architectures for keyphrase extraction. We proposed two different families of networks: the first one works similarly to Named Entity Recognition by labeling sequences, while the second one is inspired from the Answer Selection task, by comparing candidate keyphrase with his reference document. We found that, using the Inspec corpus [57] as training and evaluation set, our models outperform several known algorithms, with the architecture presented in Section 5.4.1 obtaining the best F1-score overall.

However, we believe that our results can be further improved. For example, it may be useful to implement *attention* techniques [60] which have already been proven useful, for example, in Question Answering [111]. Moreover, we believe that it is also possible to further improve the performance of the second family of networks by employing a smarter technique of candidate keyphrase generation. In the future, we plan to investigate these problems and to extend the evaluation of our approach to other datasets, to further prove its effectiveness.

Part IV

CONCLUSIONS AND APPENDICES

6 CONCLUSIONS

6.1 SUMMARY OF THE PRESENTED WORK

In this thesis we presented some of the weaknesses and strengths of the main algorithms for Supervised Automatic Keyphrase Extraction. In Chapter 2, we showed how building a supervised multilanguage keyphrase extraction algorithm is relatively easy, even when for a specific language there is a lack of a corpus of documents with their human selected keyphrases to train it. In order to build such algorithms, we provided the description of the process to develop a new dataset for keyphrase extraction in Arabic. In fact, in order to build the dataset, Internet users from all around the world were asked to assign keyphrases to Arabic document, instead of using expert knowledge. This technique allowed us to collect the dataset quickly and to train a supervised AKE algorithm using English and Arabic, and to test it on Italian, Romanian and Portuguese as well, obtaining satisfactory performances. However, this also showed how the most popular features used to train algorithms for supervised keyphrase extraction are quite "shallow", since they are typically based only on positional or statistical knowledge to distinguish between correct and wrong keyphrases.

In Chapter 3 we tried to overcome this limitations, by developing linguistics-based features for AKE in order to introduce more contextual information. Such features were based on the concept of *anaphora*, i.e., on exploiting the dependencies within certain parts of the discourse. We developed an algorithm which replaces pronouns with the entity they refer to, and uses two novel anaphora-based features. This algorithm obtained a better performance than a baseline algorithm developed using statistical and positional information only; however, it was impaired by the performance of current *anaphora resolution* techniques, which are slow and often not precise enough.

For this reasons, in Chapters 4 and 5 we turned to deep learning, an emerging field of supervised learning [43]. We developed six different deep learning architectures for keyphrase extraction which show two strong advantages with respect to classical supervised AKE algorithms. First, neural language models allow the machine learning algorithm to grasp the meaning of each word of the input document. Second, the network architectures we used (bidirectional long-short term memory networks, and convolutional neural networks) are able to use the whole content of the document to decide whether a candidate keyphrase is an actual keyphrase or not. With these architectures, we were able to outperform several other AKE algorithms when evaluating keyprase extraction performance on the Inspec dataset originally proposed by Hulth [57].

6.2 FUTURE WORK

The results presented in this work seem to show that, like many other fields, deep learning techniques may have a strong impact on the field of AKE. However, while even other scholars have presented DL algorithms for AKE with convincing results, the author of the present dissertation believes that the research in this field is still in an embryonic stage. First, the architectures we presented in Chapter 5 should be tested on several datasets to confirm their effectiveness. Moreover, as stated in Section 5.9, attentional techniques could further improve the performance of such algorithms, and there are other network architectures that could be evaluated as well: for example, Gated Recurrent Units, a variation of LSTMs, have already been proven effective in keyphrase extraction [78].

An interesting work could be to repeat the experiments of Chapter 2 by using deep learning to train a multilingual DLKE algorithm which is able to extract KPs in "new", unseen languages. For example, it has been already proven possible to train DL algorithms to translate between language pairs they have never seen: Johnson et al. [61] showed, e.g., that a network built to translate from English to Korean, and to Korean to Japanese, can also translate from English to Japanese even if not trained to do it.

Another path to explore could be to further enhance deep learning algorithms with techniques inspired by "classical" supervised AKE. In the present dissertation, taking inspiration from the design of DL algorithms for Question Answering, we used just one candidate generation technique, but for example Hulth [57] showed that carefully tuning this phase can lead to a better performance. Another interesting direction may be that of appending the features used by "classical" supervised AKE to the word embedding vector. The techniques we described in Chapter 3, for example, may be particularly useful, since even if by using neural language models we are able to inform the DL algorithm about the meaning of the words, the information conveyed by pronouns it's still lost.

However, it's also possible that deep learning techniques can be helpful to "classical" keyphrase extraction techniques: for example, topic-based algorithms like, e.g., TopicRank [23] could benefit from the use of neural language models to detect the different topics in a document.

6.3 CONCLUDING REMARKS

The work we presented in this dissertation has shown that incorporating more contextual information in supervised automatic keyphrase extraction improves its performance. The early algorithms for keyphrase extraction already used some sort of contextual information, since they were all based on TF-IDF [115, 121], which gives information about the frequency of a phrase in a corpus. In this work, after showing the strengths and the limitations of such approach in Chapter 2, from Chapter 3 onwards we restricted the context to the single input document. First, we have shown that exploiting the relations between candidate keyphrases and the information conveyed by pronouns we can improve the performance of AKE; however, this approach still relied on TF-IDF. Then, we employed Deep Learning techniques, which allowed to develop AKE algorithms using the whole content of the document as information to better detect keyphrases by using only the words of the input documents, conveniently transformed using a neural language model, in input.

The early works that apply deep learning on keyphrase extraction, including the one presented in the present document, seem to show that such algorithms are able to obtain better performance than older supervised ones, that employed in particular statistical and positional features [52] to detect keyphrases. However, performance is not the only advantage. The ability of DL algorithms to get rid of TF-IDF, a corpus-based metric, to successfully perform keyphrase extraction, makes them them able to fill the gap with unsupervised AKE algorithms. In fact, after training an unsupervised AKE algorithm, it is possible to extract a keyphrase from a document without the need of a whole corpus; this was not possible with supervised algorithms, since they were all based on TF-IDF, so they required a corpus to extract keyphrases from a document even *after* being trained. Now, with DLKE algorithms, this is no longer the case.

However, the performance of state-of-the-art algorithms for AKE is still far from being satisfactory, with the best performing algorithm presented in the current dissertation having just over 50% F1-Score. Nevertheless, we already listed a long list of research directions that could still be explored to improve the performance of AKE algorithms, with deep learning being the most promising field of research, in the opinion of who's writing this document. Moreover, even if AKE datasets are still small and often not precise, the new possibilities offered by Crowdsourcing make the collection of new, better datasets an easy task, even in currently under-resourced languages.

A THE DISTILLER FRAMEWORK

In 2015, we introduced a novel knowledge extraction framework called the Distiller Framework, with the goal of offering the research community a flexible, multilingual information extraction framework [12]. Two years later, the project has significantly evolved, by supporting more languages and many machine learning algorithms. In this Appendix we present the current state of the framework and some of its applications.

The work described in this section is contained in Basaldella, Nart, and Tasso [12], which has been presented at the 1st AI*IA Workshop on Intelligent Techniques At Libraries and Archives, held the 22nd September 2015 in Ferrara, and in Basaldella et al. [17] at the 14th Italian Research Conference on Digital Libraries (IRCDL 2018), which will be held in Udine, January 25-26, in 2018.

A.1 INTRODUCTION

Today digital document archives contain a tremendous amount of documents of various types, such as books, articles, papers, reports etc. Therefore, there is a urgent demand to adequate tools to process semantically documents in order to support the user needs.

Based on this demand, in this Chapter we present the current state of the Distiller framework, an open source information extraction framework written in Java at the University of Udine. Developed in the Artificial Intelligence Laboratory, the Distiller framework allows to annotate any document with linguistic, statistical, semantic or any kind of information. We present the history of the framework and related research in Section A.2; in Section A.3, we describe the design of the framework; then, in in Section A.4, we explain how to download and run the Distiller. In Section A.5, we briefly present research performed using the Distiller framework in the fields of Keyphrase Extraction and Named Entity Recognition in the biomedical domain. Finally, Section A.6 presents the challenges that we will have to face in future in the development of the framework.

A.2 RELATED WORK

The roots of the Distiller framework are in the AKE system DIKpE presented by Pudota et al. [99]. Originally, the system was part of a content recommendation framework, and performed AKE using five features and heuristically selected weights. Later, [87] and [30] extended the approach, offering the possibility of inferring keyphrases not contained in the original document and of processing documents in Italian as well. However, the software used in these projects was adapted using a series of ad-hoc solutions, hence becoming difficult to maintain and to further extend it with new functionality. For these reasons, we introduced the Distiller framework in [12], with the goal of building a more maintainable system which could be also used for tasks different than AKE.

Other open source AKE systems exist in academia. KEA [121], one of the first AKE algorithms, is available online as open source software[1], but the project seems abandoned since 2007. A free implementation of the RAKE [101] algorithm is available online as well[2], but with little or no possible customization. PKE [22] is an open source[3] implementation of many AKE algorithms, such as KEA, TopicRank [23], WINGNUS [91] and others. However, it is focused on keyphrase extraction only and it cannot be used for other NLP tasks. The MAUI software[4] seems the closest system to the Distiller framework, offering an source implementation of an improved version of the KEA algorithm, and algorithms for Named Entity Recognition or Automatic Tagging [77]. However, many of these features are only available buying a commercial license, and the end user is left with no or little possibility of customizing the pipelines.

A.3 DESIGN

The Distiller framework is written in Java 8, due to the robustness of the language, its strong object-oriented paradigm, and due to the availability of a large number of NLP and machine learning tools already available for this language, such as the Stanford CoreNLP library [75], Apache OpenNLP[5], Weka [103], and so on. Moreover, Java gives the possibility of writing wrappers for other software, adding flexibility to the framework. Many wrappers are already available and developed by the open source community, both for generic tools like R or Matlab, and for specialized tools, e.g., CRFSuite [93].

The design of the framework is somewhat similar to the Stanford CoreNLP system [75]. In fact, like in Stanford CoreNLP, we offer the possibility to annotate the text with a sequence of `Annotator` objects. When the developer of an information extraction pipeline is working with the Distiller, he will work by mainly using these classes of the framework:

1 http://www.nzdl.org/Kea/download.html
2 https://github.com/aneesha/RAKE
3 urlhttps://github.com/boudinfl/pke
4 https://github.com/zelandiya/maui
5 http://opennlp.apache.org

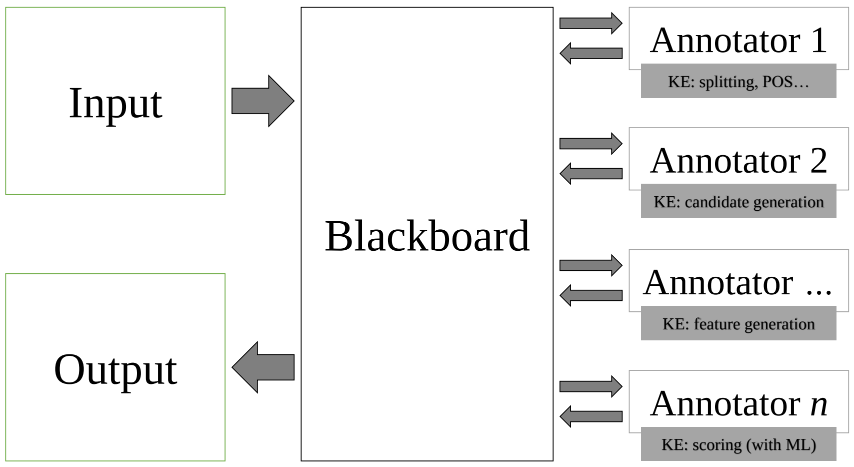


Figure A.1: The high-level architecture of the framework. The workflow is the following: first, the document is written on the blackboard. Then, a sequence of `Annotators` annotate the document, eventually using previously produced annotations. When all `Annotators` finish their job, the produced annotations are returned as output. In this case, we put some example annotators used for Keyphrase Extraction.

`DocumentComponent`: this class represents a unit of information within a document. Such unit may be a chapter, a paragraph, a sentence, or just text. It is designed using the Composite pattern [40], where the composite object (i.e. a sentence, a chapter, a section...) is represented by the `DocumentComposite` class and the smallest component is represented by the `Sentence` class, which in turn is an aggregation of `Tokens`.

`Blackboard`: this is the class that contains the original document and all the information produced by the pipeline. It contains a pointer to the root `DocumentComponent` of the document and a dictionary of `Annotations` that can be filled at the developer's will.

`Annotation`: the class that represents an annotation. It can be added to the Blackboard or any `Annotable` object. Example of annotable objects are any `DocumentComponent`, `Tokens`, `Grams`, etc.

`Annotator`: an `abstract` class that has to be extended by any class that produces annotations. An annotator can be, e.g., a part-of-speech tagger, it can count the occurrences of a word in a document, it can call an external knowledge base (e.g. Wikipedia) to get more information, it can be a machine learning algorithm, and so on.

Figure A.1 shows an example of workflow of the Distiller framework, applied to the case of AKE. In this case, the pipeline will follow the supervised AKE steps we defined in Section 1.2.1, by running the following annotators:

1. **Language detection**: first, we detect the language of the document;
2. **Low-level NLP**: then, we perform low-level NLP operations on the document, such as tokenization, part-of-speech tagging, stemming, and so on;
3. **Candidate Generation**: using the information produced at the previous step, we generate the candidate keyphrases that match certain part-of-speech patterns [11, 99];
4. **Candidate Annotation**: we annotate the candidate with information from different domains, such as statistics (e.g. number of occurrences of the candidate, length of the candidate...), linguistics (number of nouns in the candidate [99], anaphors that have the candidate as antecedent [11], . . .), or from external knowledge (e.g. Wikipedia [30]), and so on;
5. **Candidate Scoring**: we score the candidates using the annotations produced at the previous steps. The score can be calculated using simple, handcrafted techniques [12, 99] or using machine learning algorithms [11].

Some annotators are already provided out-of-the-box. For example, we provide two wrappers for Stanford CoreNLP and ApacheOpenNLP that offer sentence segmentation, word tokenization, and PoS tagging in many languages (as we've seen Chapter 2), a wrapper for the Porter's stemmer algorithm [98], a module that calculates statistical information about n-grams contained in the document, and so on.

A.4 OBTAINING AND RUNNING THE DISTILLER FRAMEWORK

Distiller is available as an open source project under the GPLv2 license. It is available online at the following URL:

`https://github.com/ailab-uniud/distiller-CORE`

After building it with Maven, it is possible to run the keyphrase extraction pipeline described in Section A.3 by writing the following code:

Listing A.1: Example Java code to call and run the Distiller.

```java
String document = ... // load your document
Distiller d = distiller = DistillerFactory.
    loadFromPackagedXML("pipelines/defaultKE.xml");
```

```
Blacboard b = d.distill(document);
Collection<Keyphrase> keyphrase =
    b.getGramsByType(Keyphrase.KEYPHRASE);
```

This code will load the default keyphrase extraction pipeline, run it, and store the results in the `keyphrase` collection. The `defaultKE` pipeline is based on the algorithm we described in Chapter 2, hence it supports keyphrase extraction in Arabic, English, Italian, Romanian and Portuguese. However, it's worth noting note that this pipeline requires R installed on your system, since it uses a MLP to score the keyphrases; if R is not available, it is possible to run the implementation of [99] AKE algorithm, called `fastKE`, which does not need any particular additional software.

As an example of the ease of configuration of the Distiller, in Listing A.1 we show (part of) the XML configuration file of the `fastKE` pipeline. As you can see, the code is very compact and allow for easy customization.

Listing A.2: The configuration XML code for the Distiller that implements the pipeline presented in [99]

```
<bean id="fastKE"
      class="it.uniud.ailab.dcore.annotation.Pipeline">
    <property name="annotators">
        <list>
            <!-- split the document -->
            <ref bean="openNLP"/>
            <!-- generate the n-grams -->
            <ref bean="nGramGenerator"/>
            <!-- annotate the n-grams -->
            <ref bean="statistical" />
            <!-- evaluate the keyphraseness -->
            <ref bean="linearEvaluator"/>>
            <!-- filter the non-interesting output -->
            <ref bean="skylineGramFilter"/>
        </list>
    </property>
</bean>
```

As we already mentioned, each module of the pipeline must implement the `Annotator` interface. An example of Annotator is the `OpenNLPBootstrapper`, a module that uses the Apache OpenNLP library to split, tokenize, and POS tag the document. This annotator is defined as a bean, as in Listing A.3, in the XML file and then passed to the pipeline as in Listing A.1. All the annotators must be defined in the same XML file as the one that contains the definitions of the pipeline.

Listing A.3: A configuration snippet

```
<bean id="openNLP"
```

```
      class="it.uniud.ailab.dcore.wrappers.external.OpenNlpBootstrapperAnnotator">
    <property name="modelPaths">
        <map key-type="java.lang.String" value-type="java.lang.String">
            <entry key="en-sent" value="/opt/distiller/models/en-sent.bin"/>
            <entry key="en-token" value="/opt/distiller/models/en-token.bin"/>
            <entry key="en-pos-maxent" value="/opt/distiller/models/en-pos-maxent.bin"/>
            <entry key="it-sent" value="/opt/distiller/models/it/it-sent.bin"/>
            <entry key="it-token" value="/opt/distiller/models/it/it-token.bin"/>
            <entry key="it-pos-maxent" value="/opt/distiller/models/it/it-pos-maxent.bin"/>
        </map>
    </property>
</bean>
```

This way, one can easily add the support for a language in the pipeline, by just adding a reference to the new models in the `openNLP` bean; on the other hand, it is also possible to replace OpenNLP with for example the Stanford NLP library, by defining a bean and calling it in the `fastKE` pipeline; it is possible to add a new annotator that implements some custom feature for AKE, and so on.

A.5 APPLICATIONS

Since the beginning of the development of the framework we immediately started to use it in actual research tasks, in order to gain experience about the challenges that developers face in designing tools for the academic world.

In Chapter 2, we used the Distiller to implement a five-language keyphrase extraction pipeline, working in English, Arabic, Portuguese, Romanian and Italian. In Chapter 3, we successfully used the Distiller framework to demonstrate the possibility of extracting better keyphrases using more linguistic knowledge than in the classic statistics based approaches.

However, the framework was also used in [14, 15] for entity recognition and linking, demonstrating its flexibility. In this case, we used Distiller along OntoGene [100], a text mining framework developed by the University of Zurich, in order to build an hybrid dictionary based - machine learning system for the detection and linking of technical terms in the biomedical domain. The system, based on CRFs, obtained promising results on the CRAFT corpus, with increased F1-Score when compared to the current state-of-the art system.

A.6 CONCLUSIONS AND FUTURE WORK

In the last years, deep learning techniques are attacking "classic" approaches for solving many problems, outperforming them in many tasks. For example, in the Machine Translation domain, the WMT 2016 task saw a surge of Neural Machine Translation systems, which vastly outperformed the systems presented the syntax-based system presented in the previous edition [20, 21]. The same happened in the ImageNet competition, where the introduction of DL techniques brought the error rate down to 3,6% from the previous, pre-"Deep Learning era" 26,1% state-of-the art [43]. DL approaches also improved the state of the art in many other fields, such as speech recognition and image segmentation, and are currently regarded of obtaining "superhuman" performance in traffic sign classification [43].

Deep learning techniques have been developed also for AKE (as we did in Chapter 5, Named Entity Recognition [106], and many other NLP tasks with promising results. This will prove a challenge for systems designed to be knowledge-based like the Distiller framework. However, there are tasks where the "older" knowledge-based approach is still dominant, as NER and Concept Recognition in specialized fields, e.g. when there is the necessity of binding concepts to specialized ontologies [14]. In these cases, where algorithms need to take into account rare words that DL models often fail to recognize, we believe systems like the Distiller still has much to offer to the research community. In addition, we believe that, due to the extensible framework architecture of Distiller, our system will continue to be useful in future, e.g. by integrating deep learning libraries inside it. For example, the popular Tensorflow [1] library offers Java APIs, so it would be easy to integrate it in the framework.

B DEEPKEYPHRASEEXTRACTION REPOSITORY

In Chapter 5, we described the achievement obtained in Keyphrase Extraction using Deep Learning techniques. As for Appendix A, where we described the Distiller framework used in Chapters 2 and 3, here we describe the software used to implement DLKE techniques.

However, this description will be much more compact. This is because, while the Distiller is a full-fledged framework that takes his roots in a decade of software written in the Artificial Intelligence Laboratory of the University of Udine, the software used in Chapter 5 is a simple *proof of concept* implementation of the algorithms we presented.

B.1 INTRODUCTION

Each Deep Learning KE pipeline is composed by a sequence of common steps, much like the "classic" supervised AKE pipelines we described many times throughout this Thesis. The DLKE pipeline is the following:

1. **Low-level NLP**: the input document is split into sentences, the sentences are split into tokens, and (in the case of "KS" the networks we described in Section 5.5) the same holds for the candidate keyphrases;
2. **Word embedding**: each token is assigned a pre-trained word embedding, and the same (eventually) holds for the candidate keyphrases (see Section 4.2);
3. **Test/Validation Sets Generation**: using the information from the previous steps, documents and candidate keyphrase are processed to generate the training and (optionally[1]) the validation sets;
4. **Model training**: the model is trained using the sets generated in the previous step.
5. **(Optional[2]) Model tuning:**: using the information provided by the validation set (i.e. validation set accuracy, recall, etc.), the model is fine tuned to obtain the best performance;

Steps 3 and 4 require to be customized for each network (recall the different network architectures we presented in Sections 5.3, 5.4, and

1 Actually, while using a training set it's optional, it's *highly* recommended to use one.
2 see above.

5.5). Steps 1,2, and 5, on the other hand, can be implemented in the exact same way for all DLKE pipelines. For this reason, we decided to implement a small library, called `deep keyphrase extraction`, where each pipeline we implemented shares the same code for (at least) steps 1,2, and 5. The benefit of such choice is two-folded: one the one hand, the code is more maintainable, having only one code base for all the experiments; on the other hand, the results of experiments are perfectly comparable, as they differ only in minimal (albeit essential) part.

B.2 DESIGN

Unfortunately, reusing the code of the Distiller framework, as presented in Appendix A, was not possible. In fact, while there are some deep learning libraries available for Java (the language used for the Distiller framework), the majority of the software developed by both the academic and commercial deep learning communities is written in Python.

One particular reason for choosing Python as a language is the availability of the Keras library [25], which was the library we used to develop our DLKE algorithms. Keras allows to quickly deploy a deep learning algorithm without actually taking care of the underlying implementation, both mathematical, as the most common layers are already implemented, and "practical", as Keras already offers three different backends: Theano [3], developed by the University of Montreal, Tensorflow [1], developed by Google, and Microsoft's Cognitive Toolkit (CNTK)[109]. All these backends offer the possibility of training the network using the GPU using NVIDIA's CUDA library [92].

The design of the system is very simple. As mentioned above, most parts of the KE pipeline are shared across the different DLKE algorithms. For this reason, we can roughly describe the structure of a DLKE script with the following pseudo-code:

```
# select the dataset
dataset = KP_EXTRACTION_DATASET
# load the data
train_data, val_data, test_data = dataset.load_data()
# preprocess data: tokenize the documents, load embeddings, and
    so on
train_data, val_data, test_data = preprocess_data(train_data,
    val_data, test_data)

# train the network
model.fit(...)

# run the network on the test set
results = model.predict(...)

```

```
# prepare results for evaluation
results = postprocess_data(results)

# evaluate
evaluate(results)
```

To implement this simple structure, the following source code files are provided:

- `data/dataset.py`: this file contains the loaders for the different datasets used in the experiments. While in Chapter 5 we experimented only with the Inspec corpus, the code is already able to load different datasets. Each loader inherits from an abstract `Dataset` class that offers common facilities to load training, testing and validation sets for each dataset.

- `utils/preprocessing.py`: this file contains the utilities to prepare the dataset in a format that is accepted by Keras' `model.fit()` function. These utilities take the documents, tokenize them, load the pre-trained word embeddings in GloVe format [97] and outputs the training, testing and validation sets ready to be fed to Keras.

- `utils/postprocessing.py`: this file contains the utilities that convert the output of Keras' `model.predict()` function in a format accepted by the evaluation methods;

- `eval/metrics.py`: this file contains the methods to calculate some common metrics for keyphrase extraction (see Section 1.6).

All these scripts depend on some other files to perform their operations. For example, a `nlp` folder is provided, with scripts for tokenizing, chunking or cleaning the output using NLTK [19]. The `utils` folder contains other utilities, e.g. for plotting the training graphs, and the `eval` folder contains other utility scripts for evaluation purposes.

B.3 REPRODUCIBILITY

All the scripts that implement a DLKE algorithm begin by setting seed for the random number generators used by the different libraries involved to ensure the reproducibility of the results over different machines. Moreover, we provide some scripts that ensure that certain CUDA features are disabled, to force deterministic training. However, please note that due to different hardware or software versions, running the scripts on different machines may yield slightly different results.

B.4 OBTAINING THE DLKE SOFTWARE

The source code software described in the present Appendix and used in Chapter 5 is available for download at https://github.com/basaldella/deepkeyphraseextraction. It is licensed under the Apache 2.0 License.

B.5 CONCLUSIONS AND FUTURE WORK

The software described in the present Appendix is an embryonic implementation of what a more complex framework for AKE may be. However, by being open source software, by offering simple methods to load and evaluate algorithms on many of the most common AKE tasks, and by using the high level DL library Keras, we believe it may be a useful testbed for developing new AKE algorithms in the future for the academic community. As stated in Section A.6 and in the Conclusions of the present dissertation, it may be interesting to integrate it with the Distiller framework or other similar KE libraries, to develop AKE algorithms that use both the newer, deep learning based approach to AKE, and the old, statistics based approach.

BIBLIOGRAPHY

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[2] Luis von Ahn and Laura Dabbish. “Labeling images with a computer game.” In: *Proceedings of the 2004 Conference on Human Factors in Computing Systems, CHI 2004, Vienna, Austria, April 24 - 29, 2004*. Ed. by Elizabeth Dykstra-Erickson and Manfred Tscheligi. ACM, 2004, pp. 319–326.

[3] Rami Al-Rfou et al. “Theano: A Python framework for fast computation of mathematical expressions.” In: *arXiv e-prints* abs/1605.02688 (May 2016).

[4] Latifa Al-Sulaiti and Eric Steven Atwell. “The design of a corpus of contemporary Arabic.” In: *International Journal of Corpus Linguistics* 11.2 (2006), pp. 135–171.

[5] Abdulmohsen Al-Thubaity, Marwa Khan, Manal Al-Mazrua, and Maram Al-Mousa. “New language resources for arabic: corpus containing more than two million words and a corpus processing tool.” In: *International Conference on Asian Language Processing (IALP)*. IEEE. 2013.

[6] Waleed Ammar, Matthew Peters, Chandra Bhagavatula, and Russell Power. “The AI2 system at SemEval-2017 Task 10 (ScienceIE): semi-supervised end-to-end entity and relation extraction.” In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, 2017, pp. 592–596.

[7] Germán Osvaldo Aquino, Waldo Hasperué, César Armando Estrebou, and Laura Cristina Lanzarini. “A Novel, Language-Independent Keyword Extraction Method.” In: *XIX Congreso Argentino de Ciencias de la Computación*. 2013.

[8] Isabelle Augenstein, Mrinal Das, Sebastian Riedel, Lakshmi Vikraman, and Andrew McCallum. “SemEval 2017 Task 10: ScienceIE - Extracting Keyphrases and Relations from Scientific Publications.” In: *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*. Ed. by Steven Bethard, Marine Carpuat, Marianna Apidianaki, Saif M. Mohammad, Daniel M. Cer, and David Jurgens. Association for Computational Linguistics, 2017, pp. 546–555.

[9] Arafat Awajan. "Keyword Extraction from Arabic Documents Using Term Equivalence Classes." In: *ACM Transactions on Asian and Low-Resource Language Information Processing* 14.2 (2015), 7:1.

[10] Ken Barker and Nadia Cornacchia. "Using Noun Phrase Heads to Extract Document Keyphrases." In: *Proceedings of the 13th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*. AI '00. London, UK, UK: Springer-Verlag, 2000, pp. 40–52. ISBN: 3-540-67557-4.

[11] Marco Basaldella, Giorgia Chiaradia, and Carlo Tasso. "Evaluating anaphora and coreference resolution to improve automatic keyphrase extraction." In: *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*. Ed. by Nicoletta Calzolari, Yuji Matsumoto, and Rashmi Prasad. ACL, 2016, pp. 804–814.

[12] Marco Basaldella, Dario De Nart, and Carlo Tasso. "Introducing Distiller: A Unifying Framework for Knowledge Extraction." In: *Proceedings of 1st AI*IA Workshop on Intelligent Techniques At LIbraries and Archives co-located with XIV Conference of the Italian Association for Artificial Intelligence, IT@LIA@AI*IA 2015, Ferrara, Italy, September 22, 2015*. Ed. by Stefano Ferilli and Nicola Ferro. Vol. 1509. CEUR Workshop Proceedings. CEUR-WS.org, 2015.

[13] Marco Basaldella, Giuseppe Serra, and Carlo Tasso. "The Distiller Framework: current state and future challenges." In: *14th Italian Research Conference on Digital Libraries (IRCDL 2018), Udine, Italy, January 25-26, 2018 (Accepted for Publication)*. 2018.

[14] Marco Basaldella, Lenz Furrer, Nico Colic, Tilia Ellendorff, Carlo Tasso, and Fabio Rinaldi. "Using a Hybrid Approach for Entity Recognition in the Biomedical Domain." In: *Proceedings of the 7th International Symposium on Semantic Mining in Biomedicine, SMBM 2016, Potsdam, Germany, August 4-5, 2016*. 2016, pp. 11–19.

[15] Marco Basaldella, Lenz Furrer, Carlo Tasso, and Fabio Rinaldi. "Entity recognition in the biomedical domain using a hybrid approach." In: *Journal of Biomedical Semantics (Accepted for Publication)* (2017).

[16] Marco Basaldella, Muhammad Helmy, Elisa Antolli, Mihai Horia Popescu, Giuseppe Serra, and Carlo Tasso. "Exploiting and Evaluating a Supervised, Multilanguage Keyphrase Extraction pipeline for under-resourced languages." In: *Recent Advances In Natural Language Processing 2017 (RANLP 2017), Varna (Bulgaria), September 4-6 2017*. 2017.

[17] Marco Basaldella, Elisa Antolli, Giuseppe Serra, and Carlo Tasso. "Bidirectional LSTM Recurrent Neural Network for Keyphrase Extraction." In: *14th Italian Research Conference on Digital Libraries (IRCDL 2018), Udine, Italy, January 25-26, 2018 (Accepted for Publication)*. Udine, 2018.

[18] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. "A Neural Probabilistic Language Model." In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155.

[19] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596516495, 9780596516499.

[20] Ondřej Bojar et al. "Findings of the 2015 Workshop on Statistical Machine Translation." In: *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Lisbon, Portugal: Association for Computational Linguistics, 2015, pp. 1–46.

[21] Ondřej Bojar et al. "Findings of the 2016 Conference on Machine Translation." In: *Proceedings of the First Conference on Machine Translation*. Berlin, Germany: Association for Computational Linguistics, 2016, pp. 131–198.

[22] Florian Boudin. "pke: an open source python-based keyphrase extraction toolkit." In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*. Osaka, Japan: The COLING 2016 Organizing Committee, 2016, pp. 69–73.

[23] Adrien Bougouin, Florian Boudin, and Béatrice Daille. "TopicRank: Graph-Based Topic Ranking for Keyphrase Extraction." In: *Proceedings of the Sixth International Joint Conference on Natural Language Processing*. Nagoya, Japan: Asian Federation of Natural Language Processing, 2013, pp. 543–551.

[24] Tim Buckwalter. *Buckwalter Arabic Morphological Analyzer Version 2.0 LDC2004L02*. Web Download. hiladelphia: Linguistic Data Consortium. 2004.

[25] François Chollet et al. *Keras*. `https://github.com/fchollet/keras`. 2015.

[26] Jason Chuang, Christopher D. Manning, and Jeffrey Heer. ""Without the Clutter of Unimportant Words": Descriptive Keyphrases for Text Visualization." In: *ACM Trans. on Computer-Human Interaction* 19 (3 2012), pp. 1–29.

[27] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: deep neural networks with multitask learning." In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*. Ed. by William W. Cohen, Andrew McCallum, and

Sam T. Roweis. Vol. 307. ACM International Conference Proceeding Series. ACM, 2008, pp. 160–167.

[28] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. "Natural Language Processing (Almost) from Scratch." In: *Journal of Machine Learning Research* 12 (2011), pp. 2493–2537.

[29] Andrew M. Dai, Christopher Olah, and Quoc V. Le. "Document Embedding with Paragraph Vectors." In: *CoRR* abs/1507.07998 (2015).

[30] Dante Degl'Innocenti, Dario De Nart, and Carlo Tasso. "A New Multi-lingual Knowledge-base Approach to Keyphrase Extraction for the Italian Language." In: *Proc. of 6th International Conference on Knowledge Discovery and Information Retrieval (KDIR)*. 2014, pp. 78–85.

[31] James Dickins, Sándor Hervey, and Ian Higgins. *Thinking Arabic translation: A course in translation method: Arabic to English*. Routledge, 2016.

[32] Samhaa R El-Beltagy and Ahmed Rafea. "KP-Miner: A keyphrase extraction system for English and Arabic documents." In: *Information Systems* 34.1 (2009), pp. 132–144.

[33] Mahmoud El-Haj, Udo Kruschwitz, and Chris Fox. "Using Mechanical Turk to Create a Corpus of Arabic Summaries." In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*. In the Language Resources (LRs) and Human Language Technologies (HLT) for Semitic Languages workshop held in conjunction with the 7th International Language Resources and Evaluation Conference (LREC 2010). Valletta, Malta, 2010.

[34] Mahmoud El-Haj, Udo Kruschwitz, and Chris Fox. "Creating language resources for under-resourced languages: methodologies, and experiments with Arabic." In: *Language Resources and Evaluation* 49.3 (2015), pp. 549–580.

[35] Tomaz Erjavec. "MULTEXT-East Version 4: Multilingual Morphosyntactic Specifications, Lexicons and Corpora." In: *Proceedings of the International Conference on Language Resources and Evaluation, LREC 2010, 17-23 May 2010, Valletta, Malta*. Ed. by Nicoletta Calzolari, Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias. European Language Resources Association, 2010.

[36] Ali Farghaly and Khaled Shaalan. "Arabic natural language processing: Challenges and solutions." In: *ACM Transactions on Asian Language Information Processing (TALIP)* 8.4 (2009), p. 14.

[37] Tom Fawcett. "An Introduction to ROC Analysis." In: *Pattern Recogn. Lett.* 27.8 (June 2006), pp. 861–874. ISSN: 0167-8655.

[38] Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. "Applying deep learning to answer selection: A study and an open task." In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*. IEEE, 2015, pp. 813–820.

[39] Eibe Frank, Gordon W. Paynter, Ian H. Witten, Carl Gutwin, and Craig G. Nevill-Manning. "Domain-Specific Keyphrase Extraction." In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Ed. by Thomas Dean. Morgan Kaufmann, 1999, pp. 668–673.

[40] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[41] G. David Garson. "Interpreting Neural-network Connection Weights." In: *AI Expert* 6.4 (Apr. 1991), pp. 46–51. ISSN: 0888-3785.

[42] Niyu Ge, John Hale, and Eugene Charniak. "A Statistical Approach to Anaphora Resolution." In: *In Proceedings of the Sixth Workshop on Very Large Corpora*. 1998, pp. 161–170.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[44] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." In: *Neural Networks* 18.5 (2005), pp. 602–610.

[45] Carl Gutwin, Gordon Paynter, Ian Witten, Craig Nevill-Manning, and Eibe Frank. "Improving Browsing in Digital Libraries with Keyphrase Indexes." In: *Decis. Support Syst.* 27.1-2 (Nov. 1999), pp. 81–104. ISSN: 0167-9236.

[46] Nizar Y Habash. "Introduction to Arabic natural language processing." In: *Synthesis Lectures on Human Language Technologies* 3.1 (2010), pp. 1–187.

[47] Mounia Haddoud and Said Abdeddaim. "Accurate keyphrase extraction by discriminating overlapping phrases." In: *Journal of Information Science* 40.4 (2014), pp. 488–500.

[48] Mounia Haddoud, Aïcha Mokhtari, Thierry Lecroq, and Saïd Abdeddaïm. "Accurate Keyphrase Extraction from Scientific Papers by Mining Linguistic Information." In: *Proceedings of the First Workshop on Mining Scientific Papers: Computational Linguistics and Bibliometrics co-located with 15th International Society of Scientometrics and Informetrics Conference (ISSI 2015), Istanbul,*

Turkey, June 29, 2015. Ed. by Iana Atanassova, Marc Bertin, and Philipp Mayr. Vol. 1384. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 12–17.

[49] Michael Alexander Kirkwood Halliday and Ruqaiya Hasan. *Cohesion in english*. Routledge, 2014.

[50] Khaled M. Hammouda, Diego N. Matute, and Mohamed S. Kamel. "CorePhrase: Keyphrase Extraction for Document Clustering." In: *Machine Learning and Data Mining in Pattern Recognition, 4th International Conference, MLDM 2005, Leipzig, Germany, July 9-11, 2005, Proceedings*. 2005, pp. 265–274.

[51] Juhyun Han, Taehwan Kim, and Joongmin Choi. "Web Document Clustering by Using Automatic Keyphrase Extraction." In: *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*. WI-IATW '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 56–59. ISBN: 0-7695-3028-1.

[52] Kazi Saidul Hasan and Vincent Ng. "Automatic Keyphrase Extraction: A Survey of the State of the Art." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 1262–1273.

[53] Muhammad Helmy, Dario De Nart, Dante Degl'Innocenti, and Carlo Tasso. "Leveraging Arabic morphology and syntax for achieving better keyphrase extraction." In: *Proc. of 20th International Conference on Asian Language Processing (IALP)*. IEEE. 2016, pp. 340–343.

[54] Muhammad Helmy, Marco Basaldella, Eddy Maddalena, Stefano Mizzaro, and Gianluca Demartini. "Towards building a standard dataset for Arabic keyphrase extraction evaluation." In: *2016 International Conference on Asian Language Processing, IALP 2016, Tainan, Taiwan, November 21-23, 2016*. 2016, pp. 26–29.

[55] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[56] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*. 2001.

[57] Anette Hulth. "Improved automatic keyword extraction given more linguistic knowledge." In: *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics. 2003, pp. 216–223.

[58] Anette Hulth. “Enhancing Linguistically Oriented Automatic Keyword Extraction.” In: *Proceedings of HLT-NAACL 2004: Short Papers*. HLT-NAACL-Short ’04. Boston, Massachusetts: Association for Computational Linguistics, 2004, pp. 17–20. ISBN: 1-932432-24-8.

[59] Radu Ion, Elena Irimia, Dan Stefanescu, and Dan Tufis. “ROMBAC: The Romanian Balanced Annotated Corpus.” In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. Istanbul, Turkey: European Language Resources Association (ELRA), 2012, pp. 339–344.

[60] Laurent Itti and Christof Koch. “Computational Modelling of Visual Attention.” In: *Nature Reviews Neuroscience* 2.3 (2001), pp. 194–203.

[61] Melvin Johnson et al. “Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation.” In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 339–351. ISSN: 2307-387X.

[62] Su Nam Kim and Min-Yen Kan. “Re-examining Automatic Keyphrase Extraction Approaches in Scientific Articles.” In: *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications*. MWE ’09. Suntec, Singapore: Association for Computational Linguistics, 2009, pp. 9–16. ISBN: 978-1-932432-60-2.

[63] Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. “SemEval-2010 Task 5 : Automatic Keyphrase Extraction from Scientific Articles.” In: *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*. Ed. by Katrin Erk and Carlo Strapparava. The Association for Computer Linguistics, 2010, pp. 21–26.

[64] Yoon Kim. “Convolutional Neural Networks for Sentence Classification.” In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1746–1751.

[65] Paul R Kroeger. *Analyzing grammar: An introduction*. Cambridge University Press, 2005.

[66] Shalom Lappin and Herbert J Leass. “An algorithm for pronominal anaphora resolution.” In: *Computational linguistics* 20.4 (1994), pp. 535–561.

[67] Quoc V. Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents.” In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 1188–1196.

[68] Marina Litvak, Mark Last, and Abraham Kandel. "Degext: a language-independent keyphrase extractor." In: *Journal of Ambient Intelligence and Humanized Computing* 4.3 (2013), pp. 377–387.

[69] Zhiyuan Liu, Peng Li, Yabin Zheng, and Maosong Sun. "Clustering to Find Exemplar Terms for Keyphrase Extraction." In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*. EMNLP '09. Singapore: Association for Computational Linguistics, 2009, pp. 257–266. ISBN: 978-1-932432-59-6.

[70] Zhiyuan Liu, Wenyi Huang, Yabin Zheng, and Maosong Sun. "Automatic keyphrase extraction via topic decomposition." In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2010, pp. 366–376.

[71] Patrice Lopez and Laurent Romary. "GRISP: A Massive Multilingual Terminological Database for Scientific and Technical Domains." In: *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*. Ed. by Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias. Valletta, Malta: European Language Resources Association (ELRA), 2010. ISBN: 2-9517408-6-7.

[72] Patrice Lopez and Laurent Romary. "HUMB: Automatic Key Term Extraction from Scientific Articles in GROBID." In: *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*. Ed. by Katrin Erk and Carlo Strapparava. The Association for Computer Linguistics, 2010, pp. 248–251.

[73] Eddy Maddalena, Marco Basaldella, Dario De Nart, Dante Degl'Innocenti, Stefano Mizzaro, and Gianluca Demartini. "Crowdsourcing Relevance Assessments: The Unexpected Benefits of Limiting the Time to Judge." In: *Proceedings of the 4th AAAI Conference on Human Computation and Crowdsourcing (HCOMP 2016)*. 2016.

[74] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[75] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. "The Stanford CoreNLP Natural Language Processing Toolkit." In: *Association for Computational Linguistics (ACL) System Demonstrations*. 2014, pp. 55–60.

[76] Luis Marujo, Anatole Gershman, Jaime Carbonell, Robert Frederking, and Joao P. Neto. "Supervised Topical Key Phrase Extraction of News Stories using Crowdsourcing, Light Filtering and Co-reference Normalization." In: *Proceedings of LREC 2012*. Istanbul, Turkey: ELRA, 2012.

[77] Olena Medelyan, Eibe Frank, and Ian H Witten. "Human-competitive tagging using automatic keyphrase extraction." In: *Proc of Conference on Empirical Methods in Natural Language Processing*. 2009.

[78] Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. "Deep Keyphrase Generation." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, 2017, pp. 582–592.

[79] Jean-Baptiste Michel et al. "Quantitative Analysis of Culture Using Millions of Digitized Books." In: *Science* 331.6014 (2011), pp. 176–182. ISSN: 0036-8075. eprint: http://science.sciencemag.org/content/331/6014/176.full.pdf.

[80] Rada Mihalcea and Paul Tarau. "TextRank: Bringing Order into Text." In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing , EMNLP 2004, A meeting of SIGDAT, a Special Interest Group of the ACL, held in conjunction with ACL 2004, 25-26 July 2004, Barcelona, Spain*. 2004, pp. 404–411.

[81] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. "Recurrent neural network based language model." In: *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*. Ed. by Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura. ISCA, 2010, pp. 1045–1048.

[82] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality." In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger. 2013, pp. 3111–3119.

[83] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." In: *CoRR* abs/1301.3781 (2013).

[84] Ruslan Mitkov. *Anaphora resolution*. Routledge, 2014.

[85] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 807–814.

[86] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Ça glar Gulçehre, and Bing Xiang. “Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond.” In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 2016, p. 280.

[87] Dario De Nart and Carlo Tasso. “A Domain Independent Double Layered Approach to Keyphrase Generation.” In: *WEBIST 2014 - Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 2, Barcelona, Spain, 3-5 April, 2014*. Ed. by Valérie Monfort and Karl-Heinz Krempels. SciTePress, 2014, pp. 305–312.

[88] Dario De Nart, Dante Degl’Innocenti, Marco Basaldella, Maristella Agosti, and Carlo Tasso. “A Content-Based Approach to Social Network Analysis: A Case Study on Research Communities.” In: *Digital Libraries on the Move - 11th Italian Research Conference on Digital Libraries, IRCDL 2015, Bolzano, Italy, January 29-30, 2015, Revised Selected Papers*. 2015, pp. 142–154.

[89] Dario De Nart, Dante Degl’Innocenti, Andrea Pavan, Marco Basaldella, and Carlo Tasso. “Modelling the User Modelling Community (and Other Communities as Well).” In: *User Modeling, Adaptation and Personalization - 23rd International Conference, UMAP 2015, Dublin, Ireland, June 29 - July 3, 2015. Proceedings*. 2015, pp. 357–363.

[90] Thuy Dung Nguyen and Min-Yen Kan. “Keyphrase Extraction in Scientific Publications.” In: *Proceedings of the 10th International Conference on Asian Digital Libraries: Looking Back 10 Years and Forging New Frontiers*. ICADL’07. Hanoi, Vietnam: Springer-Verlag, 2007, pp. 317–326. ISBN: 3-540-77093-3, 978-3-540-77093-0.

[91] Thuy Dung Nguyen and Minh-Thang Luong. “WINGNUS: Keyphrase Extraction Utilizing Document Logical Structure.” In: *Proceedings of the 5th International Workshop on Semantic Evaluation, SemEval@ACL 2010, Uppsala University, Uppsala, Sweden, July 15-16, 2010*. 2010, pp. 166–169.

[92] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. “Scalable Parallel Programming with CUDA.” In: *Queue* 6.2 (Mar. 2008), pp. 40–53. ISSN: 1542-7730.

[93] Naoaki Okazaki. *CRFsuite: a fast implementation of Conditional Random Fields (CRFs)*. 2007.

[94] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, 1999.

[95] Mari-Sanna Paukkeri and Timo Honkela. "Likey: unsupervised language-independent keyphrase extraction." In: *Proceedings of the 5th international workshop on semantic evaluation (SemEval '10)*. ACL. 2010, pp. 162–165.

[96] Mari-Sanna Paukkeri, Ilari T Nieminen, Matti Pöllä, and Timo Honkela. "A Language-Independent Approach to Keyphrase Extraction and Evaluation." In: *Proceedings of the 22nd International Conference on Computational Linguistics (COLING): Companion Volume: Posters*. ACL. 2008, pp. 83–86.

[97] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[98] Martin F Porter. "An algorithm for suffix stripping." In: *Program* 14.3 (1980), pp. 130–137.

[99] Nirmala Pudota, Antonina Dattolo, Andrea Baruzzo, Felice Ferrara, and Carlo Tasso. "Automatic keyphrase extraction and ontology mining for content-based tag recommendation." In: *Int. J. Intell. Syst.* 25.12 (2010), pp. 1158–1186.

[100] Fabio Rinaldi. "The ontogene system: an advanced information extraction application for biological literature." In: *EMBnet.journal* 18.B (2012). ISSN: 2226-6089.

[101] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. "Automatic Keyword Extraction from Individual Documents." In: *Text Mining*. John Wiley & Sons, Ltd, 2010, pp. 1–20. ISBN: 9780470689646.

[102] Alexander M. Rush, Sumit Chopra, and Jason Weston. "A Neural Attention Model for Abstractive Sentence Summarization." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, 2015, pp. 379–389.

[103] Ingrid Russell and Zdravko Markov. "An Introduction to the Weka Data Mining System (Abstract Only)." In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, Seattle, WA, USA, March 8-11, 2017*. 2017, p. 742.

[104] Motaz K Saad and Wesam Ashour. "Osac: Open source Arabic corpora." In: *The 6th International Symposium on Electrical and Electronics Engineering and Computer Science (EEECS 2010), European University of Lefke, Cyprus*. 2010, pp. 118–123.

[105] Cicero dos Santos and Maira Gatti. "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts." In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, 2014, pp. 69–78.

[106] Cicero dos Santos and Victor Guimarães. "Boosting Named Entity Recognition with Neural Character Embeddings." In: *Proceedings of the Fifth Named Entity Workshop*. Beijing, China: Association for Computational Linguistics, 2015, pp. 25–33.

[107] Adam Schenker, Abraham Kandel, Horst Bunke, and Mark Last. *Graph-theoretic techniques for web content mining*. Vol. 62. World Scientific, 2005.

[108] Natalie Schluter. "A critical survey on measuring success in rank-based keyword assignment to documents." In: *22eme Traitement Automatique des Langues Naturelles, Caen* (2015).

[109] Frank Seide and Amit Agarwal. "CNTK: Microsoft's Open-Source Deep-Learning Toolkit." In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 2135–2135. ISBN: 978-1-4503-4232-2.

[110] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[111] Ming Tan, Bing Xiang, and Bowen Zhou. "LSTM-based Deep Learning Models for non-factoid answer selection." In: *CoRR* abs/1511.04108 (2015).

[112] Takashi Tomokiyo and Matthew Hurst. "A Language Model Approach to Keyphrase Extraction." In: *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18*. MWE '03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 33–40.

[113] Yuen-Hsien Tseng. "Multilingual keyword extraction for term suggestion." In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1998, pp. 377–378.

[114] Tomoki Tsujimura, Makoto Miwa, and Yutaka Sasaki. "TTI-COIN at SemEval-2017 Task 10: Investigating Embeddings for End-to-End Relation Extraction from Scientific Papers." In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, 2017, pp. 985–989.

[115] Peter D Turney. "Learning algorithms for keyphrase extraction." In: *Information Retrieval* 2.4 (2000), pp. 303–336.

[116] Peter D. Turney. "Coherent Keyphrase Extraction via Web Mining." In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 434–439.

[117] Xiaojun Wan and Jianguo Xiao. "Single Document Keyphrase Extraction Using Neighborhood Knowledge." In: *AAAI*. Vol. 8. 2008, pp. 855–860.

[118] Jiabing Wang and Hong Peng. "Keyphrases Extraction from Web Document by the Least Squares Support Vector Machine." In: *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*. WI '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 293–296. ISBN: 0-7695-2415-X.

[119] Wikipedia. *Wikipedia — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-September-2017]. 2017.

[120] Ian H. Witten. "Browsing Around a Digital Library." In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '03. Baltimore, Maryland: Society for Industrial and Applied Mathematics, 2003, pp. 99–99. ISBN: 0-89871-538-5.

[121] Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. "KEA: Practical Automatic Keyphrase Extraction." In: *Proceedings of the Fourth ACM conference on Digital Libraries, August 11-14, 1999, Berkeley, CA, USA*. 1999, pp. 254–255.

[122] Wei You, Dominique Fontaine, and Jean-Paul Barthès. "An automatic keyphrase extraction system for scientific documents." In: *Knowledge and Information Systems* 34.3 (2013), pp. 691–724. ISSN: 0219-3116.

[123] Qi Zhang, Yang Wang, Yeyun Gong, and Xuanjing Huang. "Keyphrase Extraction Using Deep Recurrent Neural Networks on Twitter." In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, 2016, pp. 836–845.

[124] Yongzheng Zhang, A. Nur Zincir-Heywood, and Evangelos E. Milios. "World Wide Web site summarization." In: *Web Intelligence and Agent Systems* 2.1 (2004), pp. 39–53.

[125] Yongzheng Zhang, Nur Zincir-Heywood, and Evangelos Milios. "World Wide Web Site Summarization." In: *Web Intelli. and Agent Sys.* 2.1 (Jan. 2004), pp. 39–53. ISSN: 1570-1263.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LYX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of January 26, 2018 (`classicthesis` version 4.2).