Introducing Distiller: a unifying framework for Knowledge Extraction

Marco Basaldella, Dario De Nart, Carlo Tasso

Artificial Intelligence Lab
Department of Mathematics and Computer Science
University of Udine, Italy
basaldella.marco.1@spes.uniud.it, {dario.denart,carlo.tasso}@uniud.it

Abstract. The Digital Libraries community has shown over the last years a growing interest in Semantic Search technologies. Content analysis and annotation is a vital task, but for large corpora it's not feasible to do it manually. Several automatic tools are available, but such tools usually provide little tuning possibilities and do not support integration with different systems. Search and adaptation technologies, on the other hand, are becoming increasingly multi-lingual and cross-domain to tackle the continuous growth of the available information. So, we claim that to tackle such criticalities a more systematic and flexible approach, such as the use of a framework, is needed. In this paper we present a novel framework for Knowledge Extraction, whose main goal is to support the development of new applications and to ease the integration of the existing ones.

1 Introduction

Automatic Knowledge Extraction (herein KE) from natural language documents is a critical step to provide better access and classification of documents by means of semantic technologies. However, due to the current size of digital archives, one cannot expect human experts to annotate such data manually. Several tools have been developed over the past years to address this issue. However four critical issues in state-of-the-art Knowledge Extraction systems can be identified:

- *Multilinguality*: roughly half of the available Web pages include non-English text[1]. The large majority of Web users are non-English native speakers, and tasks like multilingual search, adaptation, and personalization are likely to be key features of future information access. Unfortunately KE tools show, with a few notable exceptions, a general lack of multilingual support.
- *Knowledge Source Completeness*: KE systems mostly rely on a specific knowledge source (such as DBpedia or Freebase) acting in a closed-world fashion and assuming that such knowledge source is complete. This assumption is in contrast with the open-world assumption of semantic Web technologies and shows off its limitations when applied to texts where new concepts are

[1] http://w3techs.com/technologies/overview/content_language/all

often introduced, such as scientific papers. Therefore a more flexible approach open to more than one external knowledge source and compliant to the open-world assumption seems more appropriate.
- *Knowledge Overload*: long texts such as scientific papers, may include a lot of named entities, but not all are equally relevant inside the text. State-of-the-art KE systems currently provide Named Entity Recognition but do not filter relevant entities nor include relevance measures. On the other hand Keyword and Keyphrase extraction systems usually do filter entities but do not disambiguate nor link them to DBpedia or other authoritative ontologies.
- *Flexibility*: state-of-the-art systems tend to provide a "one-size-fits-all" solution that is generally a domain independent application and, to the best of our knowledge, none of them can be easily tailored by non-KE-experts to fit specific domain requirements, assumptions, or constraints of each digital library.

To overcome this issues in this paper we introduce *Distiller*, a KE framework whose aim is to overcome these limitations by providing a complete, yet easily understandable, KE pipeline, allowing quick development of custom applications and integration of heterogeneous KE technologies.

The rest of the paper is organized as follows: in Section 2 we present some related work, in Section 3 we introduce the key concepts of the Distiller framework as well as the built-in modules, and in Section 4 we explain how to obtain and use the Distiller. Finally, Section 5 and 6 conclude and present the future extensions of our work.

2 Related Work

Named entity recognition and automatic semantic data generation from natural language text has already been investigated and several knowledge extraction systems already exist [3] such as OpenCalais[2], Apache Stanbol[3], TagMe [2], BabelNet, Babelfy [12], and so on. All these systems are tailored to a specific domain and work well in that specific domain. On the other hand several authors in the literature have addressed the problem of filtering document information by identifying keyphrases (herein KPs) and a wide range of approaches have been proposed. Different techniques of KP extraction have been identified in literature [5]. These techniques can be divided mainly in supervised techniques based on statistical, structural, and syntactic features of a document, and unsupervised techniques, which employ graph clustering and ranking techniques to find the most relevant KPs in a document.

As far as we know by now these efforts, even when they brought a significant step forward in the KP research field, rarely brought to a *systematic* and *replicable* development approach to the KP problems. At the time we write, we are not aware of the existence of an 'out of the box' solution able to offer a developer,

[2] `http://www.opencalais.com/`
[3] `https://stanbol.apache.org/`

or even a less technical-minded researcher, a solution which is both easy to use and easy to *configure* for the KP extraction problem. Moreover, while there is a wide body of state of the art algorithms, just few of them are freely available to the research community. So in this section we focus only on the KP extraction software that is available for download on the Internet.

An example of an available solution is RAKE [14]. While there is an open source implementation of the algorithm[4], it's a single purpose application with little or no configuration. There is also an open source implementation of the KEA algorithm [16] available online[5], but it seems that the project has not been updated since 2007. As for RAKE, this software is a single-purpose solution with very little customization options. The KEA algorithm is the basis for the MAUI software[6], which offers an open source implementation of an improved version of the KEA algorithm plus other tools for other common KE tasks such as Entity Recognition or Automatic Tagging [10]. Unfortunately the bulk of such useful features is part of a closed-source commercial product. Moreover, such software is not meant to be a framework, therefore extension with new modules and integration with existent systems are hard to develop. Finally, JATE[7] is a library that offers a set of KP extraction algorithms. Unfortunately, this library is not developed as a framework, but just as a collection of algorithms.

It is also important to stress that the KE domain itself lacks in standardization. Evaluation of KP extraction systems is difficult, since in the community there is little agreement on which metrics should be used: some scholars use Information Retrieval metrics [7], while others introduce new domain specific metrics like in [15]. Moreover, as we discuss in Section 3.4, there is still no shared terminology in the community.

Our work aims to be a step towards a wider, unifying direction: we want to provide to the KE and KP communities an open-source, simple, and flexible framework-based solution, which can be used for fast development and evaluation of KE and KP extraction techniques.

3 Framework Design

3.1 General Design

In order to overcome the shortcomings of state-of-the-art KE systems we extended the approach presented in [13] and [1] and formalized it in a framework named *Distiller*, whose main aim is to support research and prototyping activities by providing an environment for building testbed systems and integrating existing KE systems.

Distiller is implemented in Java, since such language is widespread among the research community and offers reasonable performance and multiplatform

[4] `https://github.com/aneesha/RAKE`
[5] `http://www.nzdl.org/Kea/download.html`
[6] `https://github.com/zelandiya/maui`
[7] `https://github.com/ziqizhang/jate`

support. Moreover, since it runs on the JVM, Distiller can be used with other popular languages such as Groovy, Scala, and Ruby[8]. Distiller relies on the Spring framework to handle dependency injection allowing easy Web deployment on Servlet containers such as Apache Tomcat.

The design of Distiller is guided by the key principle that several different types of knowledge are involved in the process of KE and should be clearly separated in order to design systems able to cope with multilinguality and multi-domain issues. For example, by now we consider four types of knowledge:

- *Statistical*: word distribution in the document and/or in a corpus of documents;
- *Linguistic*: Lexical and morphological knowledge;
- *Social-Semantic*: Knowledge derived from external sources such as Wikipedia, or more specific domain ontologies, possibly cooperatively developed;
- *Meta-Structural*: heuristics based on prior knowledge on text structure (e.g.: knowing that scientific papers have an abstract).

Linguistic knowledge is language dependant Meta-Structural knowledge is domain dependent, and Social-Semantic knowledge is both domain and language dependant. At a more practical level this principle implies that different types of knowledge must reside in distinct modules, for instance, statistical and linguistic analysis must be handled by different modules.

Distiller is organized in a series of single-knowledge oriented modules, where any module is designed to perform a single task efficiently, e.g. POS tagging, statistical analysis, knowledge inference, and so on. This allows a highly modular design with the possibility of implementing different pipelines (i.e. sequences of modules) for different tasks. All these modules are required to insert the knowledge they extract on a shared blackboard so that a module can use the knowledge produced by another module. For example an n-gram generator module can generate n-grams according to the POS tags produced by a POS tagger module. Since these modules work by *annotating* the text on the blackboard with new information, we call them *Annotators* in our framework.

Implementing Knowledge Extraction tasks with Distiller ultimately is reduced to specifying a pipeline including the right annotators. Consider for instance the task of KP Extraction introduced in Section 2. Usually such task is divided in the following steps: text pre-processing, candidate KP generation, and candidate KP selection and/or ranking. Distiller allows a quick deployment of such an application with the following annotators: a Sentence Splitter and a word Tokenizer to handle the pre-processing phase, a Stemmer, a POS Tagger and an optional Entity Linker to annotate the text, an N-Gram Generator to generate candidates, and Scoring a Filtering modules to filter the most relevant candidates according to the annotations produced in the previous steps. The resulting pipeline is shown in Figure 1. Since each Annotator provides only a specific kind of knowledge, tailoring the pipeline to specific needs requires little effort. For instance, switching to another language requires to replace only the language

[8] via the JRuby implementation.

dependant annotators, namely the POS Tagger, the Stemmer, and the Word Tokenizer. Other pipelines can be specified to implement different Knowledge Extraction and text mining tasks such as Sentiment Analysis, Summarization, or Authorship Identification.

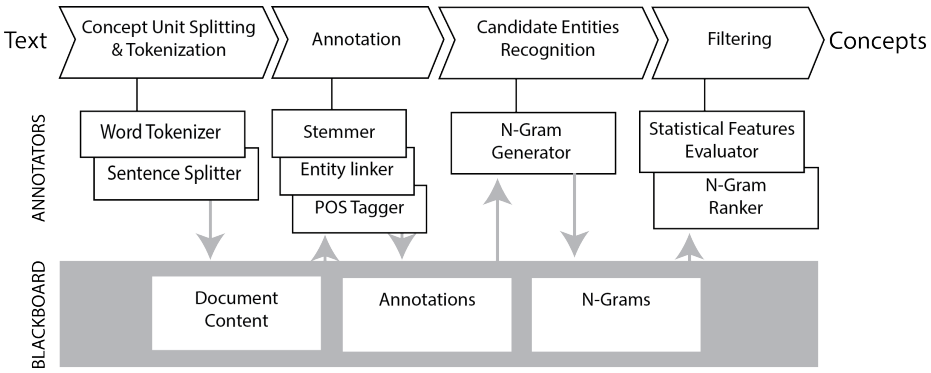

Fig. 1: Knowledge extraction pipeline. The downwards arrow indicates an annotator that writes on the blackboard, the upwards arrow indicates an annotator that reads from the blackboard.

3.2 Examples of Annotators

The framework provides out of the box a small set of annotators that allow to build a simple pipeline for the tasks of KP Extraction and Concept Inference. The pipeline we designed follows the feature-based approach which is widespread in the keyphrase extraction literature [5]. In this section, to showcase the capabilities of the framework, we present a set of annotations that the Distiller is already able to produce.

There are lots of features that can be found in literature that we have not implemented in the Distiller yet. This is not due to the fact that we don't consider them worthy or interesting enough, but, since the framework architecture offers the capability to quickly implement an Annotator that calculates a desired feature, our purpose is to provide a solid and reliable framework design rather than a simple collection of algorithms. We plan, to extend this feature set in the future, extending it also to other domains different from Knowledge Extraction such as, for example, Sentiment Analysis.

3.2.1 Linguistic Annotators We developed wrappers for two of the most popular natural language processing toolkits available in the Java language, namely the Stanford CoreNLP library [9] and the Apache OpenNLP library[9].

[9] `https://opennlp.apache.org/`

We use these tools to split, tokenize, and POS tag documents. These modules are usually the annotators at the beginning of the pipeline.

Moreover, we provide a simple n-gram generator used to generate candidate keyphrases. This module selects from the input documents the n-grams whose POS tag sequence corresponds to a typical keyphrase POS tag sequence; for example `NN NN` is a valid POS tag sequence for this module. These sequences are stored in a simple database in the shape of a JSON file. The developer can then give to the n-gram generator one database file per language, and the module is able to select the appropriate one at run time. Default pos-pattern databases that we obtained by running a POS tagger on a corpus of manually defined keyphrases, using the same approach as [6], are already available in the framework.

This n-gram generator is also used to compute what we call the Noun Value of a candidate keyphrase, i.e., given a n-gram g of length n,

$$noun\ value(g) = (number\ of\ nouns\ in\ g)/(n)$$

3.2.2 Statistical Annotators We include in the Distiller a statistical annotator that provides statistical information about n-grams generated by the n-gram generator mentioned above.

In order to illustrate how the statistical processing is performed, we introduce some definitions. Given D a document and g a gram, we denote with $|D|$ the number of sentences of the document, and $pos(D, g)$ as a function that, given a gram, returns a *list of positions* of the gram in the document. For example, suppose we have $pos(D, g) = \{1, 3, 3, 5\}$: this means that g appears in the first, third, and fifth sentence of the document, appearing two times in the third sentence. This module annotates n-grams with four features:

- **depth**: the (relative) position of the last occurrence of the n-gram, i.e.

$$\text{depth} = \frac{max(pos(D, g))}{|D|}$$

- **height**: the (relative) position of the first occurrence of the gram, i.e.

$$\text{height} = \frac{min(pos(D, g))}{|D|}$$

- **lifespan**: the part of the text in which the gram appears, i.e.

$$\text{lifespan} = \frac{max(pos(D, g)) - min(pos(D, g))}{|D|}$$

  or equivalently

$$\text{lifespan} = depth - height$$

- **frequency**: the relative frequency of the gram in the text, i.e.

$$\text{frequency} = \frac{|pos(D, g)|}{|D|}$$

These annotations provide us *positional knowledge* about the n-grams, helping us to discriminate potential keyphrases. This kind of knowledge is widely used in the keyphrase extraction field [5], albeit with different names or slightly different definitions. For example, what we call *height* is called *distance* in the KEA system [16], and it's computed on the basis of the number of words instead of sentences. The HUMB system [8] calls KEA's 'distance' simply *first position*. More recently, [4] also calls KEA's distance *first position*, and moreover it defines *first sentence* as we define *height* in this work.

We recognize that the difference in terminology may cause confusion to a reader coping with all these definitions but, since there's no standard terminology in the KP community itself, it's hard to come up with unambiguous definitions. This remarks may be indeed useful for the KP community in order to define a common corpus of definitions, eliminating the need for re-definition.

3.2.3 Knowledge-Based Annotators We built an annotator that relies on TagME [2] aimed at marking an n-gram with a boolean value if it appears on Wikipedia. We called this boolean value *Wikiflag*.

Using the information provided by this annotator, we're able to identify a set of *relevant entities* that appear in the document and a set of *suggested entities* that are related to the ones that appear in the document. This way, we provide a quick way for the reader to gather the relevant information of a document without the need of reading the whole document.

We thoroughly describe the process of filtering and suggesting entities in [11].

3.3 Multilinguality

It is simple to adapt the design of a pipeline to languages different from English. Since we use components that are quite standard in the NLP community one can use the resources that are already available online to port a pipeline from a language to another. Let's take again our Keyphrase Extraction pipeline as an example. The pipeline is already designed to support English and Italian but it's possible to support an arbitrary number of languages. In fact, the only annotators that are language-dependent are the linguistic annotators (POS tagger, splitter, and so on), the n-gram generator and the external knowledge annotators. We already mentioned that splitting, tokenization, and POS tagging are performed by external libraries such as Apache OpenNLP. To perform these tasks in languages different from English, we already offer the user a simple configuration parameter that allows him to use one of the many language models that are already available[10]. Listing 1.2 is an example of multi language support for the Apache OpenNLP wrapper in the Distiller. These models can be used to build the POS patterns for the n-gram generator, whose multilanguage capabilities we have already mentioned in Section 3.2.1

[10] `http://opennlp.sourceforge.net/models-1.5/`

Regarding the external knowledge annotators, while TagMe is available only in Italian and English, it is possible to use one of the many similar online services to perform the same task such, for example, Babelfy.

3.4 Evaluation

An important step of every scientific process is the evaluation of the results. For this reason the Distiller design allows to easily build an evaluation stage for every kind of pipeline that it can support.

As we already mentioned, the focus of the Distiller by now is on Knowledge Extraction and, more specifically, on KP Extraction, so we designed a simple evaluator process for this task. We have built an evaluation system for scientific articles based on the SEMEVAL 2010 dataset. In the near future we plan to integrate evaluation on the Inspec dataset to evaluate the pipeline on abstracts, and DUC-2001 dataset to evaluate news articles.

For the Keyphrase Extraction task, evaluation is performed by calculating the usual metrics of precision, recall and f-measure. Moreover, [7] recently introduced two metrics derived from the Information Retrieval community, namely the *binary preference measure* and the *mean reciprocal rank*, which are used to take the *ranking* of the extracted keyphrases into account. For the same reason, recently [15] proposed a new metric called *average standard normalised cumulative gain* which claims to offer a even better evaluation technique for keyphrase extraction. We use these three innovative metrics along with the usual precision, recall and f-measure in the Distiller. This way, we hope to provide a fast, accurate, and comprehensive evaluation of the KE task in our framework.

4 Using the Distiller

4.1 Distribution and Licensing

All the code of the Distiller is available online under the Apache 2 License. The full source code can be found on GitHub[11]. Due to license constraints we can't include GPL licensed code in our framework. For this reason we will not include the Stanford CoreNLP wrapper in the default release but we will release in the future a set of GPL2 licensed annotators to overcome this limit.

4.2 A practical example

Being a Spring application, Distiller can be configured with a XML configuration file. Each module can be specified and configured in such file and the system configuration can be changed with no need to recompile the code. It's also possible to configure the Distiller using Java code, but since the result is the same as the XML configuration, we cover only the latter in this paper. Listing 1.1 shows a sample configuration snippet where the KE pipeline is defined. This pipeline is injected into the Distiller using the facilities that the Spring framework provides.

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

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

Listing 1.1: A configuration snippet

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[12] to split, tokenize, and POS tag the document. This annotator is defined as a bean, as in Listing 1.2, in the XML file and then passed to the pipeline as in Listing 1.1 above.

```
<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" />
```

[12] `http://opennlp.apache.org/`

```
            <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>
```

Listing 1.2: A configuration snippet

Listing 1.2 is also useful to show how a single module can be configured. Here again we use the facilities provided by the Spring framework to set the model file paths that the OpenNLP framework is going to use in this configuration to split, tokenize and POS tag text.

Once configured, Distiller offers a simple and minimal interface to allow programmers to instantiate and run the application. Listing 1.3 shows how to build a Distiller application according to the configuration file and to launch extraction from a text. It is also possible to use the Spring framework (or the wrappers for the framework provided in the `DistillerFactory` class) to load and use any custom pipeline for the distiller.

```
Distiller d = DistillerFactory.getDefault();
DistilledOutput output = d.distill(''Text to distill'');
```

Listing 1.3: Running Distiller with the default configuration

The output format is an object containing ranked concepts, links to external knowledge sources (if any) and other annotations generated along the KE pipeline.

5 Conclusions

With respect to the four issues of KE presented in Section 1, Distiller allows the development of applications able to overcome such shortcomings. The issue of multilinguality is eased by the possibility of specifying a wide array of annotators and to dynamically link them at runtime on the basis of the considered language. The issue of Knowledge Source Completeness is eased by the possibility of integrating heterogeneous knowledge sources as different annotators, such as TagME or Babelfy. The issue of Knowledge Overload, finally, is eased by the presence of a filtering phase in which entities are evaluated with respect to their relevance in the text. Currently we are releasing the Distiller framework as an open source project and providing, by request, a RESTful API to access a sample application with multilingual support. Finally, we believe that the Distiller is flexible enough to tackle complex and diverse tasks, provided that the right annotators for these tasks are available. If an annotator for a specific problem does not exists, however, it is possible to implement it and easily plug it in a custom KE pipeline.

6 Future Work

Since the keyphrase ranking phase is based on heuristically calculated weights for the features we discussed in this paper, we plan to build a keyphrase ranking module with the possibility to use different machine learning techniques for this task. This work is out of the scope of this paper and will be discussed in a future work.

We also plan to include support for other languages in the Keyphrase Extraction task. We're currently working on Portuguese, Arabic, and Romanian.

Other future work will include the development of a different kind of pipelines in the Distiller, such as a Sentiment Analysis oriented pipeline. In order to demonstrate this possibility we already built a simple module, which is a Java port of the Syuzhet R package[13], which is used to detect the emotional intensity of a text.

References

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

[2] Paolo Ferragina and Ugo Scaiella. "TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entities)". In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. CIKM '10. Toronto, ON, Canada: ACM, 2010, pp. 1625–1628. ISBN: 978-1-4503-0099-5. DOI: 10.1145/1871437.1871689.

[3] Aldo Gangemi. "A comparison of knowledge extraction tools for the semantic web". In: *The Semantic Web: Semantics and Big Data*. Springer, 2013, pp. 351–366.

[4] Mounia Haddoud et al. "Accurate Keyphrase Extraction from Scientific Papers by Mining Linguistic Information". In: *Proc. of the Workshop Mining Scientific Papers: Computational Linguistics and Bibliometrics, 15th International Society of Scientometrics and Informetrics Conference (ISSI), Istanbul, Turkey: http://ceur-ws. org*. 2015.

[5] Kazi Saidul Hasan and Vincent Ng. "Automatic keyphrase extraction: A survey of the state of the art". In: *Proceedings of the Association for Computational Linguistics (ACL), Baltimore, Maryland: Association for Computational Linguistics* (2014).

[6] Anette Hulth. "Improved Automatic Keyword Extraction Given More Linguistic Knowledge". In: *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*. EMNLP '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 216–223. DOI: 10.3115/1119355.1119383.

[13] https://github.com/mjockers/syuzhet

[7] Zhiyuan Liu et al. “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.

[8] 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*. Association for Computational Linguistics. 2010, pp. 248–251.

[9] Christopher D. Manning et al. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2014, pp. 55–60.

[10] Olena Medelyan, Eibe Frank, and Ian H. Witten. “Human-competitive Tagging Using Automatic Keyphrase Extraction”. In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*. EMNLP ’09. Singapore: Association for Computational Linguistics, 2009, pp. 1318–1327. ISBN: 978-1-932432-63-3.

[11] Dario De Nart and Carlo Tasso. “A Keyphrase Generation Technique Based upon Keyphrase Extraction and Reasoning on Loosely Structured Ontologies”. In: *Proceedings of the 7th International Workshop on Information Filtering and Retrieval co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 6, 2013*. 2013, pp. 49–60.

[12] Roberto Navigli and Simone Paolo Ponzetto. “BabelNet: The Automatic Construction, Evaluation and Application of a Wide-Coverage Multilingual Semantic Network”. In: *Artificial Intelligence* 193 (2012), pp. 217–250.

[13] Nirmala Pudota et al. “Automatic keyphrase extraction and ontology mining for content-based tag recommendation”. In: *International Journal of Intelligent Systems* 25.12 (2010), pp. 1158–1186.

[14] Stuart Rose et al. “Automatic keyword extraction from individual documents”. In: *Text Mining* (2010), pp. 1–20.

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

[16] Ian H Witten et al. “KEA: Practical automatic keyphrase extraction”. In: *Proceedings of the fourth ACM conference on Digital libraries*. ACM. 1999, pp. 254–255.