Importance of Frameworks in Language Technology - Case of Arabic

Karim Bouzoubaa
Mohammadia School of Engineers, Mohammed V University of Rabat, Morocco
karim.bouzoubaa@emi.ac.ma

Abstract

Arabic Language Technology has known a significant progress during the last years. As a result, several tools, resources and applications have been developed such as tokenizers, Part Of Speech taggers, morphological analyzers, syntactic parsers, etc. However, most of these tools are heterogeneous and can hardly be reused in the context of other projects without modifying their source code. This problem is known to be common to all rich-resourced languages, that is why some advanced frameworks have emerged such as GATE and UIMA. These frameworks have significantly changed the way Language Technology applications are designed and developed. They provide homogenous structures for applications, better reusability and faster deployment. In this paper, (i) we present a comparative study of frameworks in order to specify which ones can suitably deal with Arabic; and (ii) report on best practises to be applied for low-resourced languages.

Keywords: Language Technology, Natural Language Processing, Rich and Low resourced languages, Arabic, Framework

Résumé

Cette technologie de langage arabe a connu un progrès significatif ces dernières années. En conséquence, de nombreuses outils, ressources et applications ont été développés tels que des tokenizer, des taggers de partie de discours, des analyseurs morphologiques, des analyseurs sémantiques, etc. Cependant, la plupart de ces outils sont hétérogènes et ne peuvent qu'avec de la modification de leurs codes être réutilisés dans le cadre d'autres projets. Ce problème est bien connu pour les langues ressources riches, pour cela des frameworks avancés ont émergé tels que GATE et UIMA. Ceux-ci ont profondément transformé la façon dont sont conçues les applications de technologie du langage. Ils fournissent des structures homogènes aux applications, une meilleure reutilisabilité et une délivrance plus rapide. Dans cette étude, (i) nous présentons une étude comparative de frameworks en vue de spécifier ceux qui peuvent convenir avec l'arabe; et (ii) nous reportons les meilleures pratiques à appliquer pour les langues ressources basses.

Keywords: Technique du langage, Traitement du langage naturel, Riches et basses ressources, Arabes, Framework

1. Introduction

One of the main issues to consider when developing any natural language processing system is choosing the most appropriate tool. For the case of Arabic language, many interesting development tools already exist: morphological analyzers to define the structure of words, stemmers to group words that have the same root or stem, syntactic parsers that determine the structures of sentences, and so on.

Most of the time, when it comes to different levels of a language (morphology, syntax, semantics and pragmatics) and because each level has its own specificity, many tools are needed at the same time. This usually leads to problems such as: integration of different technologies, more difficult maintenance of the system, a larger code in terms of the number of lines, a tedious search for appropriate solutions, etc. Thus, to avoid such problems, it is recommended to have a single integrated framework allowing researchers to develop the different aspects of the language, and which proposes:

- Basic modules of automatic processing of the Arabic language such as morphological, syntactic and semantic analysis
- Free resources (dictionaries, corpora, lexical database, etc.) and modules for comparison and evaluation
- Applications for advanced processing such as sentiment analysis, fraud detection, plagiarism, and so on.

In our view, frameworks represent an efficient way for standardization, optimization of efforts, collaboration and acceleration of developments in the field of NLP. Since our experience has mainly focused on the Arabic language, we briefly show how the Software Architecture for ARabic (SAFAR) framework can handle different levels of Arabic language.

This article is not a thorough presentation about SAFAR. The interested reader can refer to many aspects of the framework from Jaafar et al. 2018, Jaafar and Bouzoubaa 2018, Namly et al. 2016. Herein, we aim reporting on two main issues (i) understand the importance of providing a framework for any language to boost the development of language technology in that language (ii) understand how to do it by considering the Arabic as the language of study and by benchmarking all existing frameworks and their integration of tools and resources.

The rest of this article is as follows. Section 2 describes the related works concerning frameworks. Section 3 is dedicated to benchmarking these frameworks. Section 4 discusses main findings from the benchmarking. Finally, we conclude the paper.
2. Related Works

In this section we present some known and commonly used NLP architectures that support partially/entirely Arabic, namely: GATE, UIMA, LIMA, Ling-Pipe, OpenNLP, NLTK, NooJ, ATKS, AraNLPIP, MADAMIRA and SAFAR. Since we want to focus mainly on Arabic language, we break these architectures into two main categories: (1) Independent language NLP architectures that support many NLP tasks and handle many languages including Arabic, (2) ANLP architectures that are specifically designed for processing Arabic.

2.1 Independent language NLP Architectures

We give general presentations of some independent language NLP architectures without focusing on Arabic language. These general presentations introduce each architecture and get a clear idea about its structure and functionalities. This way, we will focus later on benchmarking these architectures regarding to their support of Arabic.

GATE\(^1\) (Cunningham et al. 2011) is an infrastructure for the development and deployment of components for the processing of natural language. Developed since 1995 at the University of Sheffield, it is widely used for text mining and information extraction. GATE offers an architecture, a Java framework (including many modules) and an integrated development environment. However, it has some limitations: GATE components are too abstract, and do not offer a specification in terms of API and output for Arabic NLP components.

NooJ\(^2\) (Silberztein et al. 2012) is a language development environment for building, testing and maintaining formalized descriptions with wide coverage of natural languages (in the form of dictionaries and electronic grammars), and developing language processing applications. However, it adopts a unique formalism, based on automata, and is based on a pipeline architecture to form complex processes.

UIMA\(^3\) (Ferrucci and Lally 2004) is a software architecture for the development and deployment of unstructured information analysis tools. Its purpose is to describe the processing steps of a text, image or video document in order to automatically extract structured information. The very general aim of this environment makes it a relatively low-level abstract architecture that does not offer, as such, any automatic language processing analysis module that can be used immediately. The implementation of processes for a given task thus remains the responsibility of the designer, who must be provided with analysis components developed by himself or by third parties.

OpenNLP\(^4\) (Ingersoll et al. 2013) is a Java machine learning toolkit for the processing of natural language. The main goal of the OpenNLP project is to create a mature toolkit for the most common NLP tasks: tokenization, sentence detection, part-of-speech (POS) tagging, named entity recognition, parsing, chunking, and coreference resolution. An additional goal is to provide a large number of pre-built model files for the aforementioned tasks.

NLTK\(^5\) (Bird et al. 2008) is a platform for natural language processing and text analytics. The NLTK has originally been designed to support teaching in NLP and closely related areas. However, the NLTK has not only been used successfully for teaching, but also for prototyping and building Python programs to work with natural language data.

LingPipe\(^6\), is a Natural Language Processing tool kit developed in Java by Alias-I\(^\text{I}\) company. It is designed to be effective, extensible, reusable and robust. LingPipe performs tasks such as tokenization, Part-Of-Speech tagging, named entity recognition, clustering, database text mining etc.

LIMA\(^7\) (Besançon et al. 2010), a multilingual framework for linguistic analysis and linguistic resources development and evaluation, is an NLP framework developed by the LVIC laboratory of CEA LIST.

2.2 Arabic NLP Architectures

We introduce Arabic NLP architectures that are intended to process only Arabic. We present their internal structures, components and functionalities. It should be noted that we have found only few number of such architectures for Arabic.

ATKS\(^8\), Arabic Toolkit Service, is a set of NLP components targeting Arabic language that can be exploited as web services. It was developed by Microsoft within the Advanced Technology Lab in Cairo and consists of eight ANLP components: a morphological analyzer (SARF), a spell-checker, an auto corrector, a diacritizer, a named entity recognizer (NER), a colloquial to Arabic converter, a parser and a part-of-speech (POS) tagger. It should be noted also that these components are integrated into several Microsoft products and services such as Windows, Office and Bing.

MADAMIRA\(^9\) (Pasha et al. 2014) is a toolkit for morphological analysis and disambiguation of Arabic and its dialects [5]. This toolkit provides seven ANLP processing tasks: Tokenization, morphological disambiguation for full range of morphological features, Part-of-Speech tagging, lemmatization, diacritization, named entity recognition and base phrase chunking. It should be noted that MADAMIRA is considered in our point of view as a fixed pipeline rather than a flexible

---

1 https://gate.ac.uk/
2 http://www.nooj-association.org/
3 https://uima.apache.org/
4 https://opennlp.apache.org/
5 http://www.nltk.org
6 http://www.alias-i.com/lingpipe/
7 https://github.com/aymara/lima
9 https://camel.abudhabi.nyu.edu/madamira/
toolkit that separates components from each other and give at the same time the possibility to create other types of pipelines.

AraNLP\textsuperscript{10} (Althobaiti et al. 2014) is a Java-based architecture that contains several Arabic text basic tools [4]. AraNLP attempts to bring together most of the vital Arabic text preprocessing tools into one single structure that can be accessed easily by end users. So far, AraNLP includes a sentence detector, tokenizer, light stemmer, root-based stemmer, part-of-speech tagger (POS-tagger), word segmenter, normalizer, and a punctuation and diacritic remover. Users can build customized pipelines by piping output from one tool into the next. For example, a customized pipeline can forward output from the sentence detector to the word segmenter into the POS tagger.

SAFAR\textsuperscript{11} is a Java-based framework dedicated to Arabic. It brings together all layers of ANLP: resources, preprocessing, morphology, syntax and semantics. The general idea behind SAFAR is to gather, within a single homogeneous architecture, the available set of Arabic tools that are already developed, and develop new ones if necessary. Application builders can then realize many benefits by reusing components and avoid problems of interoperability as long as all components within the framework share the same architecture. All tools within SAFAR are standardized according to several Java interfaces. Therefore, users can add new implementations of any family tools just by implementing the appropriate interface. They could also easily create customizable pipelines where the output of one tool is the input of another. SAFAR outputs can be either memory objects or output files. So far, SAFAR contains several tools and resources such as morphological analyzers, stemmers, parsers, utilities, etc.

3. Benchmarking NLP Architectures

For this benchmark, we have selected several criteria that we have grouped into four features: 1) Arabic integrated tools, 2) Arabic integrated resources, 3) flexibility of exploitation and 4) maintenance and support. It should be noted that this benchmark concerns the Arabic side within each architecture and not all its aspects. For example, UIMA has too many published articles, but we are interested only on those that concern Arabic, the same goes for integrated tools and the other criteria. That is to say, tables below aim to present how these architectures are concerned by Arabic and how they handle it.

As it is shown in table 1, NLP architectures dedicated to Arabic exceed largely independent language architectures in terms of Arabic integrated tools. This is obvious because ANLP architectures are intended to contain only Arabic tools unlike others. Indeed, the ANLP community is not encouraged to integrate its works within such language-independent architectures. This justifies the lack of Arabic language processing components within these architectures. SAFAR framework comes in the first place since it implements various tools within all its layers.

Unlike tools, resources come with fewer numbers. This is because they are time consuming when developing them comparing to some tools such as tokenizers and light stemmers. Resources also require the cooperation of computer scientists as well as linguists, which complicates the task especially for huge resources. Table 1 shows that each of LIMA and NooJ provide only one resource which are respectively a lexicon and corpora. SAFAR is the only architecture that provides many Arabic resources. Some architectures provide no Arabic resource, while others provide resources (such as clitics, roots, etc.) but used in the context of their programs and it is up to the programmer to understand the workflow of the program and extract the corresponding resource.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Tools</th>
<th>Resources</th>
<th>Flexibility</th>
<th>Maint.</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIMA</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>GATE</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>LIMA</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>NooJ</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>SAFAR</td>
<td>23</td>
<td>8</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>AraNLP</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>OpenNLP</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>NLTK</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>LingPipe</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>ATKS</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>MADAMIRA</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Architectures according to four features

The flexibility score is calculated from four different metrics which are (1) The number of possible data formats available to processes resources; (2) The possibility or not to extend the architecture and program pipelines; (3) The possibility to exploit it as an API and/or via web services; and (4) the portability or not of the architecture. UIMA, GATE and SAFAR are getting the highest score since they provide all the above mentioned features.

The score of maintenance is calculated by attributing one point for each architecture that has a release within the last five years, one point if it has documentation and one point for every Arabic published article. Table 5 shows also that most of the presented architectures are up to date regarding their releases. However, SAFAR is the only architecture that provides many published articles concerning Arabic. Each one of these articles addresses one or many aspects of processing Arabic within SAFAR. Other architectures do not provide any articles or provide few ones focusing on Arabic language as for LIMA, NooJ, AraNLP and MADAMIRA. Concerning the documentation, UIMA, GATE, SAFAR and NLTK have extensive ones. This can be very helpful to get started and be familiar with these architectures with a minimum effort from end users. Indeed, less documentation leads to more effort to discover how to manipulate it, and vice versa.

\textsuperscript{10}https://sites.google.com/site/mahajalthobaiti/resources
\textsuperscript{11}http://arabic.cki.ac.ma/safar/
4. Discussion

We can classify all architectures into four main categories namely zone “a”, “b”, “c” and “d” (Figure 1). Zone “1” represents all architectures which are dedicated to specific processing including but not limited to Arabic. Zone “2” represents all advanced architectures used to handle complex processing either for Arabic or other languages. Zone “3” concerns all architectures which are dedicated specifically to Arabic.

Figure 1: Architectures according to researchers needs

These three zones are general and can be intersected to produce new zones for more detailed and specific needs. For example, zone “a” is equal to zone “1” minus zone “b”, which represents all architectures for specific processing that are not dedicated to Arabic namely: NooJ, OpenNLP, NLTK and LingPipe. The same goes as well for zone “b”, which is the intersection of zone “1” and zone “3” and which represents all architectures for specific processing that are dedicated specifically to Arabic, namely: AraNLP, ATKS and MADAMIRA. In the other side, zone “c” is equal to zone “2” minus zone “d”, which represents all advanced architectures that are not dedicated to Arabic, namely UIMA, GATE and LIMA. Finally, zone “d” is the intersection of zone “2” and zone “3” and which represents processing for advanced processing that are dedicated specifically to Arabic, namely SAFAR.

Hence, we conclude that all the presented architectures can be used according to researches needs and specific contexts. However we believe that frameworks are an important step towards standardization, the resolution of interoperability issues, reusability and integration of all development efforts in the field of language processing.

From the above study and comparison between frameworks for the specific case of Arabic, we learn:

- Rich-resource languages have plenty of frameworks (many others exist in addition to those presented in this paper) to consider when developing for language technology purposes
- For low-resourced language, either no frameworks exist such as the Amazigh language (local language spoken mainly in Maghreb region), or very few components (tools and resources) are integrated in known and large frameworks
- For low-resourced languages, we recommend first of all to compile all existing and available components and then group them in a framework. This way, researchers would find a central point of development and would avoid develop redundant components. As a consequence, this will boost language technology development in that language.

5. Conclusion

In this paper, we presented a comparative study of Natural Language Processing architectures that can handle Arabic language. Indeed these architectures represent for us a way to standardize the various aspects shared by Arabic processing tools in order to promote interoperability. For this, we have highlighted several architectures among which we cite UIMA, GATE, AraNLP and SAFAR.

6. References


Y. Jaafar, M. Nasri, K. Bouzoubaa, "Semantic Analysis of Arabic Texts within SAFAR Framework ", 5th International IEEE Congress on Information Science and Technology CIST’18, Marrakech, Morocco, October 2018


