


NLPPort: A Pipeline for Portuguese NLP

Ricardo Rodrigues

CISUC / ESEC, Polytechnic Institute of Coimbra, Portugal


rmanuel@dei.uc.pt

 <https://orcid.org/0000-0002-6262-7920>

Hugo Gonalo Oliveira

CISUC / Department of Informatics Engineering, University of Coimbra, Portugal


hroliv@dei.uc.pt

 <https://orcid.org/0000-0002-5779-8645>

Paulo Gomes

CISUC, University of Coimbra, Portugal

pgomes@dei.uc.pt

 <https://orcid.org/0000-0002-4122-9018>

Abstract

Although there are tools for some the most common natural language processing tasks in Portuguese, there is a lack of available cross-platform tools specifically targeted for Portuguese, from end to end, namely for integration in projects developed in Java. To address this issue, we have developed and tweaked, over the last half-dozen years, NLPPORT, a set of tools that can be used in a pipelined fashion, which we have made publicly available. In this paper, we present the major features of such set of tools.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases natural language processing, tools, Portuguese

Digital Object Identifier 10.4230/OASIS.SLATE.2018.18

Category Short Paper

1 Introduction

Many high-level natural language processing (NLP) tasks rely heavily on some kind of language-specific pre-processing. Texts must be split into sentences and sentences into tokens (words and punctuation), and words often have to be further analysed. For instance, when searching for specific words, as happens in information retrieval (IR), their inflections must be considered to broaden the results. Likewise, for information extraction (IE), tools are also required for processing text and classifying its contents. Each of these tasks must be addressed, mostly in a sequential way, with the output of one tool serving as the input of the next, keeping them modular and easy to adapt and maintain.

Around 2010, in the early development stages of a question-answering system [10], in Java, we felt the need for such kind of tools for Portuguese. Although at the time there were not so many tools as those currently available, some did exist, but with several limitations, specifically when it came to language-specific knowledge. We originally relied on OpenNLP, but soon noticed that some of the tools underperformed when specific constructs of the Portuguese language were not addressed, as is the case of contractions and clitics, that conceal part of their constituents. Also missing were some essentials, like the lemmatizer. Since then, we have been developing NLPPort, with OpenNLP being used as a starting



© Ricardo Rodrigues and Hugo Gonalo Oliveira and Paulo Gomes;
licensed under Creative Commons License CC-BY

7th Symposium on Languages, Applications and Technologies (SLATE 2018).

Editors: Pedro Rangel Henriques, Jose Paulo Leal, Antonio Leitao, and Xavier Gomez Guinovart

Article No. 18; pp. 18:1–18:9



OpenAccess Series in Informatics

OASIS Schloss Dagstuhl – Leibniz-Zentrum fur Informatik, Dagstuhl Publishing, Germany

point for some of the tools – however with new models trained and other tweaks added, some of which based on the manual compilation of language-specific knowledge –, alongside MaltParser, and including other tools developed from scratch.

After a brief overview on related work, the next section present the tools of the NLPPORT suite, where key features and settings are described, together with some examples.

2 Related Work

Though some include other tasks, NLP suites of tools typically address: tokenization, part-of-speech (POS) tagging, chunking, and named entity recognition (NER) or classification. Well-established open suites, include Apache OpenNLP,¹ Stanford CoreNLP [7],² NLTK [6],³ FreeLing [9],⁴ spaCy,⁵ and LinguaKit [3].⁶ Most of them have a pipeline including the most common NLP tasks. Still, although most of these suites support or can be trained for many languages, available models do not fully support Portuguese, missing some essential tools (for instance, OpenNLP and NLTK don't include a lemmatizer for Portuguese), or are limited when it comes to using language-specific knowledge, that is not always available or is harder to compile without human intervention.

As for the application programming interfaces (API), LinguaKit uses Perl, NLTK and spaCy use Python, FreeLing uses C++, and OpenNLP uses Java. Moreover, LinguaKit and spaCy are recent – at least in the sense that when our work started, they were not available. All things considered, even though it could be possible to develop an interface between APIs in different programming languages, there was still a need for a suite of NLP tools in Java for a more streamlined integration process.

Besides the tools commonly expected in a NLP pipeline, a tool that is increasingly important is a fact extractor. Although there are some reference tools, mainly for English (e.g., ReVerb, OLLIE, ExtrHech, ClausIE) [2], for Portuguese, the scenario is still mostly barren, with the exception of LinguaKit.

That led to a decision to develop a set of tools in Java, although borrowing from already proven tools whenever possible, as is the case of the OpenNLP suite, and the MaltParser⁷ dependency parser.

3 Tools

NLPPORT includes a set of tools for Portuguese NLP that are either tweaked versions of OpenNLP's tools and MaltParser (including the creation of new models and compilation of rules) or tools entirely developed by us. These go from a sentence splitter to a fact extractor and can be used in a pipeline. In Figure 1, we can observe an overview of the pipeline of tools and how they interact.

¹ *Apache OpenNLP*: <http://opennlp.apache.org/> [Accessed: April 2018].

² *Stanford CoreNLP*: <http://stanfordnlp.github.io/> [Accessed: April 2018].

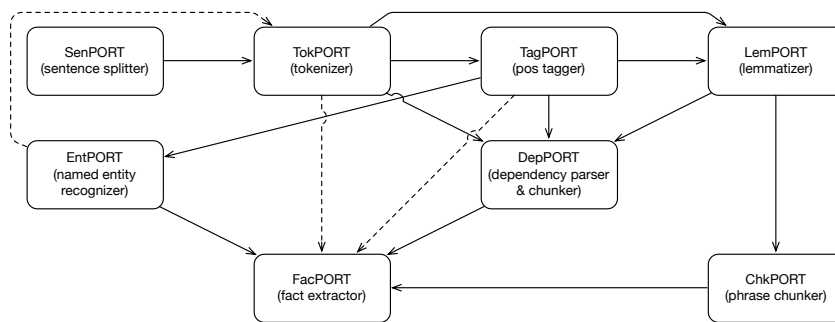
³ *NLTK*: <http://www.nltk.org/> [Accessed: April 2018].

⁴ *FreeLing*: <http://nlp.lsi.upc.edu/> [Accessed: April 2018].

⁵ *spaCy*: <http://www.spacy.io/> [Accessed: April 2018].

⁶ *LinguaKit*: <http://linguakit.com/> [Accessed: April 2018].

⁷ *MaltParser*: <http://www.maltparser.org/> [Accessed: April 2018].



■ **Figure 1** NLPPORT's Pipeline Overview.

```

<replacement target="q.b."></replacement>
<replacement target="q.e.d."></replacement>
<replacement target="q.g."></replacement>
  
```

■ **Figure 2** Examples of abbreviations used in the sentence splitter.

3.1 SenPORT

SENPORT splits text into sentences, enabling their individual processing by other tools. It is based on OpenNLP's sentence detector, the `SentenceDetectorME` class,⁸ with two major tweaks on its input and output:

- A list of abbreviations is used for avoiding splitting sentences on periods that commonly occur in abbreviations. It is applied after sentence splitting: whenever a sentence ends with an abbreviation, it is reattached to the following sentence, if any. Figure 2⁹ shows a few examples from the compiled abbreviation list.
- An option for line breaks always resulting in a new sentence – arguably true for many corpora. It is performed before applying OpenNLP's sentence detector and is addressed with a regular expression for the purpose: $(\backslash n\backslash r?) | (\backslash r\backslash n?)$.

For sentence splitting, OpenNLP's sentence detection model available for Portuguese (`pt-sent.bin`) is used¹⁰. This model was trained on *CoNLL-X Bosque 8.0* [1] data.

3.2 TokPORT

TOKPORT is our tokenizer, which relies on OpenNLP's `TokenizerME` class, the pre-trained model for Portuguese (`pt-token.bin`), but includes some tweaks for considering contractions and clitics when pre-processing the sentences, and for considering abbreviations in post-processing. More precisely, after sentences are tokenized, tokens are optionally checked for the presence of contractions and clitics, in order to better address POS tagging later, as expanding clitics in tokens, clearly separating the verb from the personal pronouns, makes it easier to identify the latter. For example, using one of the rules in Figure 3, we get the form

⁸ The ME suffix in some of OpenNLP's classes denotes the use of a maximum entropy model.

⁹ In this figure and in others that follow, excerpts of an XML based format are shown, where all "rules" are defined by means of a *target*, used to find matches in text, and of an eventual *replacement*, that, depending on the task at hand, may be optional.

¹⁰ This and other OpenNLP's pre-trained models used in this suite of tools can be downloaded from <http://opennlp.sourceforge.net/models-1.5/> [Accessed: April 2018].

```
<replacement target="-me-emos">emos a mim</replacement>
<replacement target="-me-ia">ia a mim</replacement>
<replacement target="-me"> a mim</replacement>
```

■ **Figure 3** Examples of clitics processed by the tokenizer.

```
<replacement target="à">a a</replacement>
<replacement target="ao">a o</replacement>
<replacement target="aos">a os</replacement>
```

■ **Figure 4** Examples of contractions processed by the tokenizer.

```
<replacement target="em abono de"></replacement>
<replacement target="em bloco"></replacement>
<replacement target="em breve"></replacement>
```

■ **Figure 5** Examples of token groups used in the tokenizer.

dar-me-ia – in English, “[*it*] would give me” –, that would be POS tagged just as a verb, to *daria a mim* (even if not strictly in line with the structure of Portuguese), yielding as tags a verb, a preposition and a pronoun, respectively for the three resulting tokens. The use of one option over the other may also have implications in the classifications of the other tokens by the POS tagger, by changing the entropy.

The reason for processing contractions is similar to that of clitics. In this case, prepositions and pronouns are broken apart, as shown in Figure 4. For example, *aos* (preposition) is changed into *a os* (preposition and pronoun).

Again, the abbreviation list is used for coupling the period with the respective abbreviation (and classified together) instead of being addressed as punctuation and leading to incorrect POS tags. For abbreviation examples, please refer back to Figure 2. Abbreviations with multiple periods that may have been split by the tokenizer are also put back together (e.g., *q. b.* back to *q.b.*).

We have also opted for grouping tokens during the tokenization process: proper nouns are combined in an “unbreakable” token, to be processed together (feeding back the resulting entities from NER to the tokenizer); and adverbial expressions have their elements grouped together, with some examples presented in Figure 5.

3.3 EntPORT

The named entity (NE) recognizer, ENTPORT, is based on OpenNLP’s `NameFinderME` class, and is used straight out-of-the box. Yet, as there was no pre-trained model for Portuguese available among the OpenNLP models, one had to be trained, for which we used the *Floresta Virgem* [1] treebank in the format *árvores deitadas*. Entities are thus classified as one of the following: *abstract*, *artprod* (article or product), *event*, *numeric*, *organization*, *person*, *place*, *thing*, or *time*. The trained model achieves a precision of 81.9%, a recall of 76.8% and an *F*-measure of 79.3% over the same treebank.

As stated before, the entities recognized by this tool are fed back to TOKPORT in order to bundle together the tokens that compose the entities, so that they can be identified and processed as such. This way, when the tokens of a multiword NE get to the POS tagger, they

get tagged together – for instance, as a proper noun in the case of the name of a person (e.g., {*José da Silva*} for {*José*} {*da*} {*Silva*}) – instead of being individually tagged, with benefits in the POS tagging process itself and later in other tasks that depend on it.

3.4 TagPORT

Given that the previous processing in TOKPORT already addressed most of the issues that could affect the outcome of the tagging process, OpenNLP’s POS tagger was also used straight out-of-the-box. Specifically, we use the `POSTaggerME` class with the available model for Portuguese (`pt-pos-maxent.bin`). Only a wrapper was created to ease the integration with the other NLPOR tools, yielding TAGPORT. For reference, the available POS tags are “*adjectivo*,” “*advérbio*,” “*artigo*,” “*nome*,” “*numeral*,” “*nome próprio*,” “*preposição*” and “*verbo*” – and, if considered as such, “*pontuação*”¹¹.

3.5 LemPORT

For lemmatization, we have developed LEMPORT, extensively described elsewhere [11]. Briefly, it allows for *manner*, *number*, *superlative*, *augmentative*, *diminutive*, *gender* and *verb* normalization of words, using two complementary approaches: a lexicon and rules. Among other data, the lexicon contains word inflections, their dictionary form and respective morphological classification, and is used first for attempting normalization of a given word. When that is not possible, the rules, including transformations and classes to which the transformations should be applied to, are introduced iteratively until a match in the lexicon is found or the rules are exhausted. LEMPORT currently achieves an accuracy over 98% against *Bosque 8.0*.

3.6 ChkPORT

CHKPORT uses OpenNLP’s phrase chunker, the class `ChunkerME`, out-of-the-box. Yet, as it happens for NER, no prebuilt model for Portuguese was available. Once more, we have used *Bosque 8.0*, both for training and testing the model, yielding an accuracy of 95%, recall of 96%, and *F*-measure of 95%. The inputs of CHKPORT are the tokens, their POS tags, and the lemmas. Chunks can be classified as nominal (NP), verbal (VP), prepositional (PP), adjectival (ADJP) or adverbial (ADVP) phrases. Again, except for minor aspects related to the presentation of results (e.g., including the use of the lemmas in the description of the chunks), addressed by a wrapper, the results are used directly in the pipeline.

3.7 DepPORT

For dependency parsing, we have resorted to MaltParser as the basis of both our dependency parser and, inherently, *dependency chunker*, bundled together as DEPPORT. Instead of using just the dependencies *per se*, they are used for aggregating tokens from a sentence in chunks. For example, we want to group all the tokens related to the noun identified as the subject of the sentence, rather than just get the noun itself.

The model for MaltParser was also trained with *Bosque 8.0*, after conversion to the CoNLL-X format. Resulting from the application of that model, a token can be assigned

¹¹In English: *adjective*, *adverb*, *article*, *noun*, *numeral*, *proper noun*, *preposition*, *verb*, and *punctuation*, respectively.

[tokens]					
<i>id</i>	<i>form</i>	<i>lemma</i>	<i>pos</i>	<i>head</i>	<i>dependency</i>
1	Mel_Blanc	mel_blanc	prop	21	SUBJ
2	era	ser	v-fin	0	ROOT
3	alérgico	adj	adj	21	SC
4	a	a	prp	22	A<
5	cenouras	cenoura	n	23	P<
6	.	.	punc	21	PUNC

↳

[chunks]			
<i>id</i>	<i>head</i>	<i>function</i>	<i>tokens</i>
1	2	SUBJ	[Mel_Blanc]
2	0	ROOT	[era]
3	2	SC	[alérgico a cenouras]

■ **Figure 6** Dependency parsing and chunking example.

one of many grammatical functions. Of those, only the following can be selected as direct dependents of the *root* token in the next processing step (except for the *root* token itself and *punctuation* tokens):

- (Predicate) Auxiliary Verb (PAUX);
- (Predicate) Main Verb (PMV);
- Adjunct Adverbial (ADVL);
- Adjunct Predicative (PRED);
- Auxiliary Verb (AUX);
- Complementizer Dependent (>S);
- Dative Object (DAT);
- Direct Object (ACC);
- Focus Marker (FOC);
- Main Verb (MV);
- Object Complement (OC);
- Object Related Arg. Adverbial (ADVO);
- Passive Adjunct (PASS);
- Predicator (P);
- Prepositional Object (PIV);
- Punctuation (PUNC);
- Root (ROOT);
- Statement Predicative (S<);
- Subject (SUBJ);
- Subject Complement (SC);
- Subject Related Arg. Adverbial (ADVS);
- Top Node Noun Phrase (NPHR);
- Topic Constituent (TOP);
- Vocative Adjunct (VOC).

Once a sentence is processed by the dependency parser, tokens are grouped in what we call *dependency chunks*. These chunks are formed by selecting the tokens whose head is the *root* of the sentence, and then by aggregating each of those tokens together with all of their dependents. Please refer to Figure 6 for an example of *dependency chunking*. In the same figure, we can observe the results of the tokenizer, the lemmatizer and the POS tagger, alongside dependency parsing and chunking.

3.8 FacPORT

For fact extraction, FACPORT uses NEs and phrase or dependency chunks, combined with a set of rules. Facts are, for all purposes, triples with extra information or metadata, which are composed of:

- an **identifier** – a unique identifier of a fact in relation to each sentence, auto-incremented;
- a **subject** – the subject of the fact, usually a NE or some thing related to the object;
- a **predicate** – the predicate of the fact, typically a verb;

```

<replacement target="[NP] [NP]">is a</replacement>
<replacement target="[NP] [em:PP]">is in</replacement>
<replacement target="[NP] [de:PP]">is part of</replacement>

```

■ **Figure 7** Examples of rules for extracting facts of adjacent phrase chunks.

- an **object** – the object of the fact, usually something related to the subject or a NE, reverse mirroring the contents of the subject;
- a **sentence identifier** – a unique identifier of a sentence in relation to each document, auto-incremented;
- a **document identifier** – a unique identifier for each document (e.g., its filename).

Rules for fact extraction from sentences are only used in the case of adjacent *phrase chunks*. For *dependency chunks*, we use directly the dependency classification of the chunks, as in $\text{SUBJ} + \text{ROOT} + \text{OBJ} \rightarrow \text{subject} + \text{predicate} + \text{object}$ ¹². Figure 7 shows some rules for extracting facts from adjacent phrase chunks.

When using phrase chunks, the fact extractor checks them for the presence of NEs. When a match occurs, adjacency relations between chunks are used to extract facts. For instance, if a NP chunk contains a person NE and is immediately followed by another NP chunk, it is highly probable that the second chunk is a definition or specification of the first, thus yielding the fact $NP_n \text{ is a } NP_{n+1}$. The rules specify the classification of the adjacent chunks and also some elements that the chunks may or must contain, appended using a prefix or a suffix with a colon (:).

For dependency chunks, the *subject* and *object* chunks are checked for the presence of entities, and a triple is built using the corresponding predicates and objects or subjects, accordingly. Subject, predicate, and object chunks are then transposed into the corresponding fields of a newly created fact.

Fact extraction is currently limited to the presence of NEs (or proper nouns, when no NEs are available or recognized as such) in both the phrase and dependency chunks. It does not have to be like that, but it is a form of reducing the extraction of spurious facts. We do intend to revise this option, devising a way of selecting facts that may be meaningful even without NEs or proper nouns in them.

In Figure 8, we present examples of facts extracted from a sentence, including an erroneous fact identified with an asterisk. Nevertheless, it can be easily acknowledged that facts do summarize the key information bits in the sentence.

4 Conclusions and Future Work

We have presented NLPPORT, a set of tools that provide cross-platform (using Java) end-to-end Portuguese NLP. It borrows from OpenNLP and MaltParser, but includes multiple tweaks and compiled knowledge that allow for improving the output of its tools, considering constructs such as clitics and contractions.

The tools can be used individually or in a pipeline, where the output of one is the input of the next. A major feature is their modularity, allowing for changes and improvements individually in each of them (eventually improving the overall results of the pipeline). As a

¹² Actually, objects (OBJ) are defined, for instance, as dative objects (DAT) or direct objects (ACC), or are not objects at all, as is the case of subject complements (SC), but the structure maintains.

```

Mel Blanc, o homem que deu a sua voz a o coelho mais famoso de o mundo, Bugs
Bunny, era alérgico a cenouras.

↳

Fact: {subject=[Bugs_Bunny], predicate=[ser], object=[a sua voz a o coelho
mais famoso de o mundo]}*
+
Fact: {subject=[Mel_Blanc o homem que deu a sua voz a o coelho mais famoso de
o mundo Bugs_Bunny], predicate=[ser], object=[alérgico a cenouras]}
+
Fact: {subject=[Mel_Blanc], predicate=[ser], object=[o homem que deu a sua voz
a o coelho mais famoso de o mundo Bugs_Bunny]}
+
Fact: {[subject=[Mel_Blanc], predicate=[ser], object=[o homem]}
+
Fact: {[subject=[Mel_Blanc], predicate=[ser], object=[alérgico a cenouras]}

```

■ **Figure 8** Example of facts extracted from a sentence.

whole or individually, some of these tools have been used in the past both inside our research group (e.g., [4, 10, 5, 8]) and outside (e.g., [12]).

Although some of the tools have a only narrow margin for improvement, namely the dependent from external ones, others can greatly benefit from increased interest and use, as is the case of FACPORT, which remains to be properly evaluated against existing solutions.

Finally, a missing tool that is being considered for future development and integration in the suite is one for coreference resolution, which could be easily exploited by FACPORT.

The NLPPORT suite of tools is freely available for download, as a whole or each of the tools individually, at GitHub¹³.

References

- 1 Cláudia Freitas, Paulo Rocha, and Eckhard Bick. Floresta Sintá(c)tica: Bigger, thicker and easier. In *8th Conference on Computational Processing of the Portuguese Language (PROPOR)*, pages 216–219, 2008.
- 2 Pablo Gamallo. An overview of open information extraction. In *3rd Symposium on Languages, Applications and Technologies (SLATE)*, pages 13–16, 2014.
- 3 Pablo Gamallo and Marcos Garcia. LinguaKit: Uma ferramenta multilingue para a análise linguística e a extracção de informação. *Linguamática*, 9(1):19–28, 2017.
- 4 Hugo Gonçalo Oliveira. *Onto.PT: Towards the Automatic Construction of a Lexical Ontology for Portuguese*. PhD thesis, University of Coimbra, 2013.
- 5 Hugo Gonçalo Oliveira, Diogo Costa, and Alexandre Pinto. Automatic generation of Internet Memes from Portuguese news headlines. In *12th Conference on Computational Processing of the Portuguese Language (PROPOR)*, volume 9727, pages 340–346, 2016.
- 6 Edward Loper and Steven Bird. NLTK: the natural language toolkit. In *Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics (ETMTNLP)*, pages 63–70, 2002.
- 7 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *52nd Annual Meeting of the Association for Computational Linguistics*, pages 55–60, 2014.

¹³<http://github.com/rikarudo/>

- 8 Ana Oliveira Alves, Ricardo Rodrigues, and Hugo Gonçalo Oliveira. ASAPP: Alinhamento semântico automático de palavras aplicado ao Português. *Linguamática*, 8(2):43–58, 2016.
- 9 Lluís Padró, Miquel Collado, Samuel Reese, Marina Lloberes, and Irene Castellón. FreeLing 2.1: five years of open-source language processing tools. In *7th Language Resources and Evaluation Conference (LREC)*, pages 931–936, 2010.
- 10 Ricardo Rodrigues. *RAPPort: A Fact-Based Question Answering System for Portuguese*. PhD thesis, University of Coimbra, 2017.
- 11 Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. LemPORT: a high-accuracy cross-platform lemmatizer for Portuguese. In *3rd Symposium on Languages, Applications and Technologies (SLATE)*, pages 267–274, 2014.
- 12 Derry Tanti Wijaya and Tom Mitchell. Mapping verbs in different languages to knowledge base relations using web text as interlingua. In *15th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 818–827, 2016.