Improving NLTK for Processing Portuguese

João Ferreira
Department of Informatics Engineering of the University of Coimbra, Portugal
jdcoelho@student.dei.uc.pt

Hugo Gonçalo Oliveira[1]
Centre for Informatics and Systems of the University of Coimbra, Portugal
Department of Informatics Engineering of the University of Coimbra, Portugal
hroliv@dei.uc.pt

Ricardo Rodrigues
Centre for Informatics and Systems of the University of Coimbra, Portugal
College of Education of the Polytechnic Institute of Coimbra, Portugal
rmanuel@dei.uc.pt

Abstract

Python has a growing community of users, especially in the AI and ML fields. Yet, Computational Processing of Portuguese in this programming language is limited, in both available tools and results. This paper describes NLPyPort, a NLP pipeline in Python, primarily based on NLTK, and focused on Portuguese. It is mostly assembled from pre-existent resources or their adaptations, but improves over the performance of existing alternatives in Python, namely in the tasks of tokenization, PoS tagging, lemmatization and NER.

2012 ACM Subject Classification Computing methodologies → Natural language processing

Keywords and phrases NLP, Tokenization, PoS tagging, Lemmatization, Named Entity Recognition

Digital Object Identifier 10.4230/OASIcs.SLATE.2019.18

Funding This work was funded by FCT's INCoDe 2030 initiative, in the scope of the demonstration project AIA, "Apoio Inteligente a Empreendedores (*Chatbots*)". We also thank Fábio Lopes for his help with NER based on CRF.

1 Introduction

Nowadays, the amount of information that can be accessed is virtually infinite, and we may have every answer that is known by mankind at the palm of our hand. Yet, we often seem to be unable to find, understand or use it for other purposes. This happens because most information is presented with a fair amount of what can be considered noise – for instance, utterances or contextualization text that are not always relevant – and in text format, which forces us to read large blocks of text to get simple answers.

Natural Language Processing (NLP) addresses such issues. It comprises tasks that range from tokenization to Named Entity Recognition (NER), and applications such as Information Retrieval (IR), Information Extraction (IE) and Automatic Summarization.

Computational systems that deal with human language are complex, and have improved over time, but, in many respects, are still somewhat sub-optimal. Furthermore, results are generally worse when we consider languages other than English, such as Portuguese, because there are many more researchers working on and, thus, more tools developed for English. This worsens with language specificity, which may stop an otherwise global system for performing every NLP task in the same way for English and Portuguese.

[1] Corresponding author

© João Ferreira, Hugo Gonçalo Oliveira, and Ricardo Rodrigues;
licensed under Creative Commons License CC-BY

In order to get better outcomes in NLP, we looked at the foundations of such systems – the set of NLP tools that process information. A typical NLP toolkit is formed by several modules that operate in a pipeline – that is, one module usually feeds the next and more information is added in each stage. Some of the baseline modules include tokenization, Part-of-Speech (PoS) Tagging, Lemmatization, Syntactic Parsing, and Named Entity Recognition (NER)

Several toolkits provide the previous tools. However, for Portuguese, there are only a few, and, to the best of our knowledge, none with a full pipeline (e.g., including lemmatization) in Python. Due to its growing use in the Artificial Intelligence and Data Processing fields, and its ease of use, many are using Python – as such, we chose it as the programming language of the pipeline to assemble. Therefore, the first steps were to understand which NLP pipelines already exist and how they could be improved.

Our main reference for the process was the NLPPort toolkit [14], also focused on improving the processing of Portuguese in OpenNLP [9], a toolkit in Java. Given that we were using Python, some experiments were performed, leading us to select NLTK [1] as the base pipeline, on top of which improvements were to be done or new modules plugged. Despite covering the Portuguese language, results with NLTK are limited and far from perfect. Following the approach of NLPPort, a post-processing step was added to NLTK's tokenizer to handle clitics and contractions. Further, in opposition to the default model for Portuguese, the NLTK PoS tagger was trained in manually annotated corpora. Based on the outputs of the tokenizer and the PoS tagger, a lemmatizer was developed, once again inspired by NLPPort. In NLTK, such a module was unavailable for Portuguese. Lastly, a Conditional Random Fields (CRF) NER tool was trained and added, towards easier training and better results in identifying named entities. These changes allowed for a better processing of Portuguese text. As we show, using NLPyPort, rather than the default NLTK pipeline, leads to more reliable results in the baseline tasks, which should have a positive impact on higher level tasks and applications. Although one option would be to make contributions directly to NLTK, for the time being, we chose to adapt it to our needs by wrapping NLTK and adding features: testing different models for already existing modules and changing or adding other modules.

The remaining of the paper is organized as follows. Section 2 briefly analyses and compares some of the existing toolkits. Section 3 describes how the pipeline was assembled, explaining what was changed from existent elements and what was created from scratch. Section 4 presents the results of testing the new pipeline and the comparison of these to previously existing results. Finally, Section 5 presents the conclusion extracted from our results and explain how the pipeline can be further improved in the future.

2 Related Work

There is already a handful of NLP tools and toolkits, some of which freely available for use, and supporting Portuguese. Without getting into much detail, we believe the following toolkits should be considered peers to the pipeline we are developing, and therefore with results that we should aim to obtain.

In Java, one of the most known, is the OpenNLP toolkit, and from which a more use-ready version for Portuguese was assembled – the NLPPort toolkit. Another well-known Java-based toolkit is the Stanford CoreNLP [7], which does not support Portuguese out-of-the-box, but has enough resources for training models of any language.

In Python, there are two main alternatives: spaCy[2] whose utilization was considered (see Section 3); and the aforementioned NLTK [1], upon which the current pipeline is developed.

Among other toolkits that support Portuguese, in other programming languages, it is worth mentioning Freeling [11], developed in C++, and LinguaKit [5], developed in Perl.

3 The NLPyPort Pipeline

When choosing the base pipeline, several aspects were considered. First, due to the aforementioned reasons, our goal was to develop a pipeline in the Python programming language. This limited the choice of tools to NLTK and spaCy. In order to choose between them, some shallow testing was made to determine which tool was better-suited for the purpose, i.e., which was easier to use, change and adapt. We came to the conclusion that, even though spaCy is more friendly from a user perspective, as it allows to process text through the whole pipeline with a single function, NLTK allowed for a better manipulation of each stage of the pipeline, individually, making the changes more obvious.

For this reason, NLPyPort is based on the NLTK toolkit, with some layers and modules added. NLTK already provides a series of resources that can be used for Portuguese. It has a generic tokenizer, a PoS tagger trained on the "Floresta Sintá(c)tica" [3] corpus, and a general Named Entity Recognizer. With these, the base results were obtained and then changes were made, towards better results. Changes are described in detail in the following subsections. Briefly, a layer was added to the tokenizer, the PoS tagger was re-trained, and a lemmatizer was implemented based on LemPort. Lastly, a new NER, based on a CRF, was trained and integrated in the pipeline. The assembled pipeline will provide the user all the results specified, but can be changed to output only the desired data from a given task.

To illustrate the results of NLPyPort, Fig. 1 depicts the output of the initial pipeline, respectively for the simple NLTK and the NLPyPort pipeline, for the sentence: "*O António Costa deu um passeio no Porto.*"

Token	*PoS*
O,	art
António,	prop
Costa,	nprop
deu,	v-fin
um,	art
passeio,	n
no,	adp
Porto,	nprop
.	punc

Token	*Lemma*	*PoS*	*Entities*
O,	o,	art,	O
António,	antónio,	prop,	B-PESSOA
Costa,	costa,	nprop,	I-PESSOA
deu,	dar,	v-fin,	O
um,	um,	art,	O
passeio,	passeio,	n,	O
em,	em,	prp,	O
o,	o,	art,	O
Porto,	porto,	n,	B-LOCAL
.	.,	punc,	O

Figure 1 Output of simple NLTK (left) and of the NLPyPort pipeline (right).

3.1 Improving the Tokenizer

One of the critical elements of any pipeline is the tokenizer. It is responsible for splitting text into the smallest units that represent information – tokens. Since this is, for most cases, the starting point of NLP, its errors will propagate throughout the whole pipeline and thus impact performance. For this reason, it is essential to get the best possible results out of it.

[2] https://spacy.io

We first need to understand where tokenization may fail for the specific language, or where it could be better handled, towards more useful outputs. Upon exploring NLTK, we concluded that many of the errors obtained were due to language specific elements not handled properly; this was expected because NLTK's tokenizer is operating in a language-independent manner. The main problem found was not splitting tokens of two distinct types, clitics and contractions, which, depending on the application, results in loss of information.

A **clitic** is a word form generated by joining and concatenating, by means of an hyphen, a verb and a personal pronoun into a single word – for instance, "*[eu] comprei-o*" ↦ "*[eu] comprei ele*" (I have bought it). A correct separation will allow for the PoS tagger to better identify the verb and the personal pronoun as such. A **contraction** is generated when a preposition and a pronoun are joined and concatenated into a single word – for instance, "*na*" ↦ "*em a*" (in the). Similarly to the clitics, proper separation into two words should improve PoS tagging.

The implementation of these modules was rather simple, mainly because the most work-intensive part had already been done – the creation of lists of clitics and contractions. These were part of NLPPort, and available in a standard XML format, which allowed their easy integration in Python. In order to apply these, the default NLTK's tokenizer is used and both lists are checked to find word matches with any of the known contractions or clitics. Such cases are changed to the expanded from, replacing the original word by simpler words.

3.2 Improving the PoS tagger

As mentioned earlier, improving tokenization would improve PoS tagging. This happens because most PoS tagging approaches are based on models for automatic sequence labelling. So, when both clitics and contractions are not handled, both new tokens (e.g., *lo*, *no*) and PoS tags for these (e.g., *ADP* instead of *PREP+DET*) are introduced, with a negative impact on the generalization power of the tagger. Not to mention that some of those tokens may increase ambiguity (e.g., "*nos*" can either be the clitic for "*a nós*" or the contraction of the preposition "*em*" with the determinant "*os*").

NLTK provides a few different models for PoS tagging, one of which is based on Maximum Entropy. In theory, it can learn from any PoS tagged text and it is language independent. Yet, the quality of the training text used is a determinant factor in the results. Its base implementation uses not only the current word, but also some of the previous and of the following words. In fact, it is not limited to words and uses other features, such as PoS tags, if available. This means that, even if there was no change in the current word, the fact that the previous word had been split into two other words with two other tags will influence the tag of the current word.

The default NLTK PoS tagging model for Portuguese is learned from the "Floresta Sintá(c)tica" corpus [3]. Even though it contains a large number of annotated documents, not all were manually reviewed. So, in order to improve the PoS tagging model, we decided to use only "Bosque", a smaller, but manually annotated subset of "Floresta Sintá(c)tica". The reasoning behind this is that a manually annotated resource is less likely to have errors, and thus a system that learns from it should do its job better. In addition to "Bosque", the PoS tagger was also trained with the "Mac-Morpho" [2] corpus, this too manually annotated, yielding a combined number of 35 PoS tags.

3.3 Integrating a Lemmatizer

In order to normalize text, NLTK includes the RSLP stemmer for Portuguese [10]. This tool is based on the removal of suffixes from Portuguese words and, while it may be worth for some text mining tasks, the result is often not an existing word. An alternative would be a lemmatizer that transforms words in their dictionary form, thus easing their linking with lexicons or other resources. However, NLTK does not include a Portuguese lemmatizer.

Due to its wide range of applications, we chose to implement a Portuguese lemmatizer and added it to the pipeline. The code and design for this module was largely inspired in LemPort, a part of NLPPort. This allowed for using the same resources as LemPort, and apply them exactly in the same situations.

The lemmatizer is defined by two modules: the first relies on a lexicon; the second is a set of predefined hierarchical rules that will perform transformations on the token, in case there is no match with lexicon words.

For the first module, the process is straightforward: each word with a valid tag is searched on LABEL morphology lexicon [13]. If the lexicon contains the word, its lemma is retrieved and the lemmatization process ends. If, on the other hand, the word is not in lexicon, rules corresponding to the tag are applied and repeated until there is a match against the lexicon. This process must be repeated after each rule is applied, as there are words that must go through multiple normalizations – for instance, gender and number. If no rules match the word and it is not found in the lexicon, it is assumed to be already in its lemmatized form.

It is also important to note that each word may have a ranking, used when there is more than one lemmatization possibility. This ranking was determined by the word's frequency in the frequency lists of the AC/DC project [15].

The second module is based on a hierarchical set of rules, that are applied if no match was found in the lexicon. In order to make the correct lemmatization, a set of handcrafted rules perform it in a progressive order. This way, some normalizations are always made before others, which allows for a reduction of the rules on the following normalizations. To better understand this, lets take the example of a noun of feminine gender and plural form. If we applied gender normalization before number normalization, the result would not match an existing lemma. By executing the number normalization first, we only need the masculine form of the noun (if applicable). Considering this simplification, the following order of rules was defined: (1) Manner (adverb) normalization; (2) Number normalization; (3) Superlative normalization; (4) Augmentative normalization; (5) Diminutive normalization; (6) Gender normalization; (7) Verb normalization (for regular and irregular verbs).

To illustrate this, lets consider the word "*certinhas*". It will first be processed by the number normalizer ("*certinhas*" $\mapsto$ "*certinha*"), then the diminutive normalizer ("*certinha*" $\mapsto$ "*certa*"), and lastly the gender normalizer ("*certa*" $\mapsto$ "*certo*"). The final lemmatized form would be the much simple word "*certo*".

3.4 Adding a new Named Entity Recognizer

The most recent addition to the pipeline was a Named Entity Recognizer (NER) module. The default NLTK's NER tool offers many challenges when training on a new model or a different set of categories. There is no way to train it out-of-the-box. Even if the user can understand how to do it, they need to previously know for which entity categories it is being trained, and cannot get them from the training data. For this reason, we considered an alternative and ready-to-use NER module, with easier training, while maintaining or even improving the baseline NLTK's results. The NER tags utilize were those of the Second

HAREM collection[4]. The integrated NER module is based on CRFsuite, an implementation of Conditional Random Fields (CRF) [6], which, in a not so distant past, have been used for NER with relative success. Although originally available by Naoaki Okazaki [8], we used a wrapper for the scikit-learn toolkit[3], easier to train and test than NLTK NER module.

Entities were identified with the BIO notation. This is a way to indicate if a token is the beginning (B), inside (I), or outside (O) of an entity. This letter is then followed by the type of entity. An example of this can be seen in Figure 1.

4 Measuring Improvements

We recall that our goal was to achieve better results on the processing of the Portuguese language, using Python. So, in order to quantify the improvement over NLTK, we compared our results to the base results of using NLTK out-of-the-box.

4.1 Improvements in Tokenization

To measure how much the tokenizer has improved, we used it to tokenize the documents in the manually-reviewed Bosque corpus. Achieved results were compared to those obtained with the original NLTK's tokenizer. For both tokenizers, Table 1 reports on the proportion of correct tokens over all the tokens in the corpus.

Table 1 Percentage of correct tokens.

Toolkits	Tokenization Accuracy(%)
NLTK	83
NLPyPort	90

Numbers confirm that the changes made in the tokenization module lead to an improvement of 7% over NLTK's baseline. As mentioned earlier, this is due to the separation of clitics and contraction. An example of this can be seen in Fig. 1, where the word "*no*" is split into two simpler words: "*no*" $\mapsto$ "*em o*" .

4.2 Improvements in PoS tagging

To assess the second change in the pipeline, the PoS tagger was fed with the Bosque tokenization, one per line. The previous tokens were then tagged and the result compared to the tags in Bosque. Per token accuracy for both the original PoS tagging model of NLTK and the Bosque trained NLPyPort PoS tagger are reported in Table 2.

Table 2 Percentage of correct tags.

Tookits	PoS tagging(%)
NLTK	60
NLPyPort	86

Figures show significant improvements on PoS tagging after the changes made, namely of 26% over the base pipeline. We recall that this is due to training in different corpora.

[3] `https://github.com/TeamHG-Memex/sklearn-crfsuite`

An example of PoS tagging can also be seen in Fig. 1. The word "no", previously classified as *adp*, is now split and its parts differently classified ($\text{no}_{\#adp} \mapsto \text{em}_{\#prp} + \text{o}_{\#art}$).

4.3 Improvements in Tokenization followed by PoS tagging

Improvements over the original NLTK's pipeline were confirmed for the tokenizer and the PoS tagger, individually. Since each module feeds the next one, we can assume that these improvements are also reflected on the pipeline. To test whether this is true, we compared the results of tokenization followed by PoS tagging, using the original NLTK and using NLPyPort. Table 3 shows the per-token accuracy of PoS tagging, after the improvements in the tokenizer and in the PoS Tagger. Again, Bosque was used as our gold reference.

Table 3 Percentage of correct tags after Tokenization.

Tool	PoS tagging after Tokenization(%)
NLTK	58
NLPyPort	82

Results show that there is a substantial improvement, of 24 points, when the new tokenizer and PoS tagger are used together, one after the other, thus confirming the improvements achieved with NLPyPort. Differences of these tasks against the original NLTK are also seen in Figure 1.

With NLPyPort, complex tokens are now split into simpler tokens, and more of the tokens are correctly classified. For example, we can see that there has been a division of the word "*no*", as defined earlier, into the to simpler forms "*em*" and "*o*", which then leads to a better classification of the tags. NLTK also fails to identify the punctuation in the sentence.

4.4 Improvements in NER

A study of Portuguese NER with NLTK has already been made [12], and the results obtained were used as a base for our comparison. To ensure that the tests are fair, the same train and test splits used to assess the default NLTK NER module were used for CRF NER, as well as the testing approach – ten-fold cross-validation with 4 repeats on a BIO conversion[4] of the Second HAREM [4] golden collection, where named entities were manually annotated.

The metrics used for the comparison were Precision, Recall and F1-score, which were compared to those presented by Pires [12]. The obtained results are reported in Table 4.

Table 4 Comparison of Precision, Recall and F-Measure of NLTK NER and CRF NER.

NER	Precision(%)	Recall(%)	F-Measure(%)
Baseline	30.58	31.38	30.97
CRF	57.05±2.45	46.56±1.29	51.28±1.49

Results are far from perfect and some points below the 0.59 best F1 in Second HAREM [4]. Yet, this comparison confirmed that the CRF NER outperforms the default NLTK's NER for Portuguese, and has therefore been a worthwhile addition to the assembled pipeline.

[4] For the NER datasets used, check `https://github.com/arop/ner-re-pt/`.

5 Conclusions and Future Work

We have presented NLPyPort, a new NLP pipeline implemented in Python and based on NLTK, specifically focused on Portuguese processing, for which improvements were achieved. The current version of NLPyPort is freely available from: `https://github.com/jdportugal/NLPyPort`.

In the near future, we will keep with the continuous improvement of the different modules of the pipeline and, similarly to NLPPort, we will add a feedback loop for sending the identified entities to the PoS tagging module, hopefully leading to better results. Another possible and needed addition to this pipeline is a more user friendly interface that allows the user to change the pipeline as desired. Given their potential utility for more advanced applications, we will also consider the addition of other modules, namely a phrase chunker and a relation extractor.

References

1 Steven Bird and Edward Loper. NLTK: The Natural Language Toolkit. In *Proceedings of ACL 2004 on Interactive poster and demonstration sessions*, page 31. ACL, 2004.

2 Erick Rocha Fonseca and João Luís G Rosa. Mac-Morpho Revisited: Towards Robust Part-of-Speech Tagging. In *Proceedings of 7th Brazilian Symposium in Information and Human Language Technology*, 2013.

3 Cláudia Freitas, Paulo Rocha, and Eckhard Bick. Um mundo novo na Floresta Sintá(c)tica — o treebank do Português. *Calidoscópio*, 6(3):142–148, 2008.

4 Cláudia Freitas, Paula Carvalho, Hugo Gonçalo Oliveira, Cristina Mota, and Diana Santos. Second HAREM: advancing the state of the art of named entity recognition in Portuguese. In *Proceedings of 7th International Conference on Language Resources and Evaluation*, LREC 2010, La Valleta, Malta, May 2010. ELRA.

5 Pablo Gamallo and Marcos Garcia. LinguaKit: uma ferramenta multilingue para a análise linguística e a extração de informação. *Linguamática*, 9(1):19–28, 2017.

6 John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. 18th International Conference on Machine Learning*, ICML '01, pages 282–289. Morgan Kaufmann, 2001.

7 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. ACL Press, 2014.

8 Naoaki Okazaki. CRFsuite: a Fast Implementation of Conditional Random Fields (CRFs), 2007. URL: `http://www.chokkan.org/software/crfsuite/`.

9 Apache OpenNLP. Apache software foundation. *URL http://opennlp. apache. org*, 2011.

10 Viviane Moreira Orengo and Christian Huyck. A Stemming Algorithm for the Portuguese Language. In *Proceedings of 8th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 183—-193, Laguna de San Raphael, Chile, 2001.

11 Lluís Padró and Evgeny Stanilovsky. FreeLing 3.0: Towards wider multilinguality. In *Proceedings of 8th International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2473–2479, Istanbul, Turkey, May 2012. ELRA.

12 André Ricardo Oliveira Pires. Named Entity Extraction from Portuguese Web Text. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2017.

13 Elisabete Ranchhod, Cristina Mota, and Jorge Baptista. A Computational Lexicon of Portuguese for Automatic Text Parsing. In *Proceedings of SIGLEX99 Workshop: Standardizing Lexical Resources*. ACL Press, 1999.

14 Ricardo Rodrigues, Hugo Gonçalo Oliveira, and Paulo Gomes. NLPPort: A Pipeline for Portuguese NLP (Short Paper). In *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

15 Diana Santos and Eckhard Bick. Providing Internet access to Portuguese corpora: the AC/DC project. In *Proceedings 2nd International Conference on Language Resources and Evaluation*, LREC 2000, pages 205–210, 2000.